

Xi Wang
ZhiWu Li

Scheduling and Reconfiguration of Real-Time Systems

A Supervisory Control Approach

 Springer


Scheduling and Reconfiguration of Real-Time Systems


Xi Wang • ZhiWu Li

Scheduling and Reconfiguration of Real-Time Systems

A Supervisory Control Approach

 Springer

Xi Wang 
School of Electro-Mechanical Engineering
Xidian University
Xi'an, China

ZhiWu Li 
Institute of Systems Engineering
Macau University of Science and
Technology
Tapei, Macau SAR, China

ISBN 978-3-031-41968-3 ISBN 978-3-031-41969-0 (eBook)
<https://doi.org/10.1007/978-3-031-41969-0>

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Switzerland AG 2023

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Paper in this product is recyclable.

*In memory of my parents, Shuhua Zhao
and Fuyou Wang*

*For my family, Jun Guo and Peilin Jesse
Wang*

(XW)

*For my family, Tongling Feng and Buzi Li
(ZWL)*

Foreword

This monograph provides a newly identified research topic, namely real-time scheduling and reconfiguration based on supervisory control theory (SCT). SCT, also known as the R-W method, initiated by Ramadge and Wonham in the 1980s, offers a systematic treatment to the modelling, analysis, and control of discrete-event systems (DES) that are a technical generalization of many contemporary technological systems, and explores generic principles to a wide range of application domains. Generally, any system can be viewed as a real-time system (RTS) if it performs real-time application functions and behaves correctly depending on given logical activities and satisfying specified deadlines for the activities. In the past decades, as a popular and topical research topic, real-time scheduling and reconfiguration have far-reaching influences in academia and industrial implementations.

This monograph provides broad views and detailed introductions to SCT and its application in real-time scheduling and reconfiguration. Based on three popular SCT modelling frameworks, DES, timed DES (TDES), and state-tree structures (STS), the authors provide RTS modelling frameworks; thereafter, SCT is used to find their safe execution sequences. For STS, as a newly identified hierarchical modelling framework, in Chap. 2, the authors explain their semantics, notions, and concepts in a straightforward and easy to understand way with plenty of examples. Along with the deepening of understanding of the RTS modelling problem, the authors provide three RTS modelling methodologies based on DES, TDES, and STS in Chaps. 4, 5, and 7, respectively. Finally, the main differences among the three SCT modelling frameworks are vividly illustrated in Chap. 8, which also show that the cores of all the presented modelling approaches are identical.

Intuitively, it is trivial to find out that “time” is a critical criterion and vital attribute to both TDES and RTS. Hence, as the first step, the authors provide a TDES-based RTS modelling, scheduling, and reconfiguration formalism in Chap. 4 for RTS running on uni-processors. In order to schedule and reconfigure RTS in a uniformed model, a multi-period periodic RTS task is modelled by a TDES. Initially, a task is assigned with the shortest period. By implementing SCT, the multi-periods of tasks are used to reconfigure an RTS when its initial safe execution sequence set

is empty. During the real-time scheduling/reconfiguration process, the supervisor proposes all the safe execution sequences.

An RTS modelling approach is presented in Chap. 5, which uses (untimed) events to represent the execution and preemption of each individual RTS task. This modelling formalism brings the possibilities to model the preemptions of tasks' executions. Furthermore, in some cases, priorities cannot be assigned to real-time tasks. In order to solve this problem, a matrix-based priority-free conditional-preemption (PFCP) relation is provided in Chap. 5, which generalizes fixed-priority (FP) RTS scheduling. By defining the preemption relation among any two tasks running in a processor, a preemption matrix is utilized to describe all the possible FP preemption relations and other user-specified preemption relations. By SCT, the synthesized supervisor provides all the safe real-time execution sequences. As a natural extension, a generalized modular modelling framework is proposed in Chap. 6 to model the task parameters instead of the global real-time task. The modular models are taken to be generic entities, which also considers the exact execution time of real-time tasks.

STS are undoubtedly recognized as a computationally efficient SCT framework which manages the state explosion problem significantly. Clearly, the synthesized supervisors based on the modelling methodologies stated in Chaps. 4, 5, and 6 can provide all the possible safe execution sequences of an RTS. As a natural extension, it is theoretically significant and practically invaluable to find a few best sequences. To this end, Chap. 7 uses STS to model RTS and assigns dynamic priorities as specified optimality criteria such that a set of safe execution sequences is selected. This provides the possibilities for large-scale industrial implementations.

This monograph is a comprehensive and solid work on SCT-based real-time scheduling and reconfiguration. The two authors have been working in this area for years. I am pleasantly surprised to state that there are several highlights in this monograph; in particular, “untimed” events are used to represent the “timing” behaviors of RTS tasks. This insight gives vitality to this interdisciplinary research topic and brings it to a new horizon.

February 2023

Maria Pia Fanti
Politecnico di Bari

Preface

Adequate responses to predictable and specific constraints are a critical criterion to both real-time systems (RTS) and supervisory control of discrete-event systems (DES). RTS satisfying the (predefined) specific time constraints mean that they can respond to events, such as a task's arrival/release and the start/completion/preemption of its execution, within preset time. DES are event-driven instead of clock-driven, which are asynchronous in event executions and discrete in state space. Hence, properly representing the "events" of RTS by DES frameworks is the key of the modelling methods provided in this monograph. This motivates us to investigate the RTS scheduling and reconfiguration problems from the perspective of supervisory control of DES.

Supervisory control theory (SCT, known as the R-W method), initiated by Ramadge and Wonham in the 1980s, is a methodology to automatically synthesize supervisors of DES, which restricts the behavior of a plant such that the given specifications are fulfilled in a minimally restrictive manner. Based on three popular SCT modelling frameworks: DES, timed DES (TDES), and state-tree structures (STS), uniformed frameworks to model, schedule, and reconfigure RTS, are provided in this monograph. Given an RTS modelled by any framework, user-defined RTS scheduling and reconfiguration requests are described using proper specifications. By SCT, all the safe execution sequences of an RTS are embodied in its optimal supervisor.

Based on TDES, DES, and STS frameworks, multi-period periodic real-time task models are provided for integrating real-time scheduling and reconfiguration in uniformed models. A multi-period is a period set varying between a lower bound and an upper bound. The default period for a task is the shortest one. In the case that an RTS is non-schedulable, the multi-period is used to reconfigure the RTS automatically. By SCT, an RTS is claimed to be non-schedulable if its supervisor is empty.

Normally, in a real-time scheduling process, fixed or dynamic priorities are assigned to real-time tasks. However, in some special cases, scheduling priorities cannot be assigned to tasks properly. As a solution, based on DES and STS frameworks, by defining the preemption relation among any two tasks running in

a processor, a priority-free conditionally-preemptive (PFCP) scheduling method is provided, which generalizes priority-based preemption. In particular, based on the STS modelling and supervisory control mechanism, by assigning dynamic priorities, a set of safe execution sequences is selected, which rank at the top according to specified optimality criteria. Generally, the provided dynamic priority specifications can be combined with the PFCP specifications.

Historically, RTS scheduling and reconfiguration are associated with two problems: schedulability checking and safe execution sequence searching. In comparison, SCT-based real-time scheduling and reconfiguration do not need to check the schedulability and find safe execution sequences separately. As a general extension, all the possible (instead of only one in general) safe execution sequences are provided by the optimal supervisors.

The outline of this monograph is as follows: Chap. 1 reports the overviews of this monograph, SCT, and the real-time scheduling theory. The motivation, contribution, and the outline of this book are also provided. Chapter 2 presents the preliminaries of SCT and three DES modelling frameworks: DES, TDES, and STS. From the perspective of RTS scheduling, Chap. 3 reviews the real-time scheduling and reconfiguration of periodic RTS. Chapter 4 recalls the seminal work on SCT-based real-time scheduling. Thereafter, based on it, in order to dynamically reconfigure RTS, a new formalism is presented to assign multi-periods to periodic tasks. Chapter 5 points out that using DES to model RTS is more general than using TDES, and provides the possibility of preemptive SCT-based scheduling of RTS, where a DES-based periodic real-time task modelling method is presented. The timing constraints of RTS tasks are represented by different events. In the light of the multi-period reported in Chap. 4 and building on Chap. 5, a DES version modular multi-period is presented in Chap. 6. A task is represented by an automaton synchronized by the required modular models, in which a multi-period task is assigned with a set of possible periods between a minimum period and a maximum period. The only difference of a task's model before and after its reconfiguration is the upper bound of its multi-period. Based on nonblocking supervisory control of STS, a hierarchical RTS model is presented in Chap. 7, where both conditionally-preemptive and dynamic priority scheduling are addressed in the real-time scheduling. The proposed modular models are taken to be generic entities, which are utilized to model a problem domain such as "hard real-time manufacturing and reconfigurations" and manage its manufacturing production process. Finally, Chap. 8 concludes the contributions of this monograph and proposes some possible future extensions. The main differences among the three SCT modelling frameworks are also discussed in this chapter.

The primary audiences are the researchers and practitioners in SCT or RTS, equipped with the basic SCT knowledge. Also, they should be familiar with the SCT synthesis tools TCT, TTCT, STS, or other related ones. This book will appeal specifically to a reader concerned with how to model, schedule, and reconfigure an RTS.

Xi'an, China
Macau SAR, China
February 2023

Xi Wang
ZhiWu Li

Acknowledgments

We would like to take this opportunity express our sincere gratitude and appreciation to Professor W. M. Wonham, Department of Electrical and Computer Engineering, University of Toronto. Since 2013, we have been collaborating in supervisory control of DES, TDES, and STS and their implementation on real-time scheduling and reconfiguration. We are very grateful to Professor Thomas Moor, Lehrstuhl für Regelungstechnik, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany. Since 2018, we have been jointly working in supervisory control of DES and STS and their application in real-time scheduling and reconfiguration.

We extend very special thanks to our colleagues Dr. Chan Gu, School of Electrical and Control Engineering, Shaanxi University of Science and Technology (China), and Dr. Deguang Wang, School of Electrical Engineering, Guizhou University (China). The continuing interaction and stimulating discussions with them have been a constant source of encouragement and inspiration.

This monograph was in part supported by the National Natural Science Foundation of China under Grant Nos. 61703322, 61374068, 61673309, and 61603285, the Fundamental Research Funds for the Central Universities under Grant Nos. ZYTS23016, XJS17071, JBX170413, JB160401, and JB160416, the Science and Technology Development Fund, Macau Special Administrative Region (MSAR), under Grant No. 0012/2019/A1, the Natural Sciences and Engineering Research Council of Canada (NSERC) under Grant DG480599, and Alexander von Humboldt Foundation.

Finally, we are deeply in debited to Jun Guo (the first author's wife) and Tongling Feng (the second author's wife), for their care and encouragement throughout the writing of this monograph.

Contents

1	Introduction	1
1.1	Overview of This Monograph	1
1.2	Supervisory Control Theory	2
1.3	Real-Time Scheduling Theory	3
1.4	Motivation and Contribution	7
1.4.1	RTS Modelling Methods	7
1.4.2	RTS Scheduling and Reconfiguration	10
1.5	Monograph Outline	13
	References	14
2	Preliminaries of Supervisory Control Theory	17
2.1	Discrete-Event Systems	17
2.2	Timed Discrete-Event Systems	24
2.3	State-Tree Structures	31
2.3.1	Superstates	31
2.3.2	Holons	34
2.3.3	State-Trees	38
2.3.4	State-Tree Structures	42
2.3.5	Predicates	47
2.3.6	State Feedback Control	48
2.3.7	Compact Representation of Predicates	51
2.4	Real-Time Scheduling/Reconfiguration Based on Supervisory Control	53
	References	53
3	Real-Time Scheduling and Reconfiguration	55
3.1	Real-Time Systems	55
3.2	Fixed Priority Scheduling	58
3.3	Dynamic Priority Scheduling	60
3.3.1	Earliest Deadline First Scheduling	62
3.3.2	Least Laxity First Scheduling	64
3.4	Elastic Period Model for Reconfiguration	67
	References	68

4 Non-Preemptive Scheduling/Reconfiguration Based on Supervisory Control of TDES	71
4.1 Introduction	71
4.2 RTS Modelled by Timed Discrete-Event Systems	73
4.2.1 Multi-Period RTS Task Model	73
4.2.2 Real-Time Tasks Modelled by Timed Discrete-Event Systems	74
4.2.3 Global RTS Execution Model	80
4.2.4 Timed Discrete-Event System Generators	81
4.3 Dynamic Scheduling and Reconfiguration of Multi-Period RTS	84
4.4 Case Study: Supervisor Synthesis of Motor Network	85
4.4.1 Real-Time Scheduling	86
4.4.2 Dynamic Reconfiguration	88
4.4.3 Multi-Periods in the Safe Execution Sequences	91
4.5 Conclusion	92
References	92
5 Priority-Free Conditionally-Preemptive Real-Time Scheduling Based on R-W Method	95
5.1 Introduction	95
5.2 Task Models and Preemption Policies	97
5.2.1 Task Model	97
5.2.2 Priority-Free Real-Time Scheduling	98
5.2.3 Preemption Matrices	99
5.2.4 Task-Centered Conditional-Preemption Constraints	100
5.2.5 Response Time Constraints	101
5.3 Tasks Modelled by Discrete-Event Systems	101
5.4 Specifications Modelled by Discrete-Event Systems	106
5.4.1 Nonblocking Specifications	107
5.4.2 Matrix-Based Priority-Free Conditional-Preemption Specifications	108
5.4.3 Task-Centered Specifications	109
5.4.4 Response Time Constraint Specifications	111
5.5 Case Study I: Supervisor Synthesis of Motor Network	112
5.5.1 Work Plan I	113
5.5.2 Work Plan II	118
5.6 Case Study II: Supervisor Synthesis of Manufacturing Cell	123
5.7 Conclusion	125
References	129
6 Modular Scheduling/Reconfiguration with Exact Execution Time Based on R-W Method	131
6.1 Introduction	131
6.2 Modular RTS Models	133
6.2.1 RTS Tasks	134

- 6.2.2 Periodic/Sporadic Task Execution Time Models 136
- 6.2.3 Non-Repetitive Execution Time Models 138
- 6.2.4 Deadline Models 139
- 6.2.5 Release and Multi-Period Models 141
- 6.3 Global RTS Execution Models 143
 - 6.3.1 Approach I 143
 - 6.3.2 Approach II 145
 - 6.3.3 Global RTS Behavior 148
- 6.4 Scheduling Based on Supervisory Control 151
- 6.5 Case Study: Manufacturing Cell 153
 - 6.5.1 Task Models with Worst Case Execution Time 153
 - 6.5.2 Task Models with Exact Execution Time 159
- 6.6 Conclusion 160
- References 161
- 7 Scheduling/Reconfiguration Based on Supervisory Control of STS 163**
 - 7.1 Introduction 163
 - 7.2 RTS Modelled by State-Tree Structures 164
 - 7.2.1 RTS Tasks 165
 - 7.2.2 Execution Time Models 166
 - 7.2.3 Deadline Models 168
 - 7.2.4 Release Time and Period Models 170
 - 7.2.5 Task Models 171
 - 7.2.6 Global RTS Execution Models 173
 - 7.3 Conditionally-Preemptive Specifications 173
 - 7.3.1 Matrix-Based Conditional-Preemption Specifications 174
 - 7.3.2 Task-Centered Specifications 175
 - 7.4 Dynamic Specifications 176
 - 7.4.1 Earliest-Deadline First Task Selection at Arrival 176
 - 7.4.2 Partially Preemptive Earliest-Deadline First Scheduling 177
 - 7.4.3 Partially Non-Preemptive Earliest-Deadline First Scheduling 177
 - 7.5 Supervisor Synthesis with a Case Study: Manufacturing Cell 178
 - 7.5.1 Compact Encoding of the Manufacturing Cell 179
 - 7.5.2 Conditionally-Preemptive Scheduling 180
 - 7.5.3 Preemptive and Non-Preemptive Earliest-Deadline First Scheduling 182
 - 7.5.4 Non-Preemptive Earliest-Deadline First Scheduling Sequences 183
 - 7.6 Large RTS Example 184
 - 7.7 Conclusion 185
 - References 186
- 8 Conclusion and Future Work 189**
 - 8.1 Conclusion 189
 - 8.1.1 RTS Modelling Methods 189

8.1.2 An Overview of Specifications Describing RTS	
Scheduling Requirements	195
8.2 Future Work	197
References	197
Glossary	199
Index	201

Acronyms

ATG	Active transition graph
BCET	Best-case execution time
BDD	Binary decision diagram
B-W	Brandin-Wonham
DES	Discrete-event system
DM	Deadline monotonic
DP	Dynamic priority
EDF	Earliest deadline first
FP	Fixed priority
FSM	Finite state machine
HFSM	Hierarchical finite state machine
LLF	Least laxity first
LM	Release map
MNSC	Marking nonblocking supervisory control
NCA	Nearest common ancestor
NSC	Nonblocking supervisory control
OS	Other user-defined specification
PFCP	Priority-free conditional-preemption
PS	Priority-free conditional-preemptive specification
RM	Rate-monotonic
RTS	Real-time system
R-W	Ramdge-Wonham
SCT	Supervisory control theory
SM	Scheduling map
STS	State-tree structure
TDES	Timed discrete-event system
TS	Task-centered conditional-preemption specification
TTG	Timed transition graph
WCET	Worst-case execution time
WCRT	Worst-case response time

Chapter 1

Introduction



1.1 Overview of This Monograph

Initiated by Ramadge and Wonham in the 1980s [38, 48], *supervisory control theory* (SCT) of *discrete-event systems* (DES) aims to provide a general treatment for modelling and control of a wide class of man-made systems that nowadays are usually computer-integrated, including many contemporary information extensive infrastructures of human society, such as manufacturing systems, smart city traffic management, communication protocols, computer networks, etc. In general, timing constraints are a vital attribute and a critical criterion to operate these systems, whose technical generalization leads to the notion of *real-time systems* (RTS). Roughly speaking, the behavioral correctness of an RTS does not depend only on logical activities occurring in it, but also on the satisfiability of specified deadlines for the activities, i.e., the correct functioning of an RTS is attributed to the hard or soft timing requirements on the tasks to be executed. To this end, *real-time scheduling and reconfiguration* are treated as an effective vehicle for an RTS to carry out the tasks such that a correct function is guaranteed.

In 1973, the seminal work by Liu and Layland [28] touched upon the real-time scheduling problem for multi-program on a single processor, showing that a fixed priority scheduler possesses an upper bound to processor utilization. It is also shown that full processor utilization can be achieved by dynamically assigning priorities in terms of the current deadlines of tasks to be executed. Over the past half century, there has been a fair amount of significant work from researchers and practitioners on real-time scheduling and reconfiguration of RTS [16, 37, 40, 49, 51]. Interestingly and surprisingly, a suite of documented methodologies on the modelling, control, and scheduling of these systems [12–15, 21, 22, 34–36, 43–47] are from the perspective of supervisory control of DES [4, 29, 30, 38, 48].

As an interdisciplinary approach, SCT-based real-time scheduling and reconfiguration are a newly-identified research topic. In this monograph, an RTS is modelled by the SCT modelling frameworks and user-defined scheduling/reconfiguration

requests are described by the corresponding specifications. Currently, there are three popular SCT modelling frameworks: *DES* [38, 48] (known as the R-W method), *timed DES* (TDES) [4], and *state-tree structures* (STS) [29, 30]. In this monograph, RTS are modelled by all the three frameworks. Users are invited to propose scheduling or reconfiguration requirements at will. These scheduling/reconfiguration requirements, whether priority-based or not, from the perspective of either processor or individual task, are converted into formal SCT *specifications* offline. With such specifications assigned, all the safe execution sequences of an RTS are embodied in its optimal supervisor. In particular, as stated in Chap. 7, by assigning *dynamic priorities* (*DP*) to RTS tasks, a few safe execution sequences are selected based on the STS modelling and supervisory control mechanism, which rank at the top according to specified optimality criteria.

Historically, in the real-time scheduling research area, both the RTS scheduling and reconfiguration are associated with two problems [39]: *schedulability* checking and safe execution sequences searching. Usually, if an RTS is schedulable, one random safe execution sequence is provided by a scheduling algorithm for its processor. In comparison, SCT-based real-time scheduling and reconfiguration do not need to check the schedulability and find safe execution sequences separately. As a general extension, all the possible (instead of only one) safe execution sequences are provided by the optimal supervisors.

1.2 Supervisory Control Theory

This section provides a brief overview of the basic principles of SCT. Interested readers are suggested to refer to monographs [48] and [29] for a systematic understanding. The detailed preliminaries of SCT with examples are presented in Chap. 2. Unlike in [29], we introduce STS by starting from *superstates* and *holons*. The latter are natural extensions of DES diagrams.

The DES framework is language-based. Generally, a *plant* is modelled as a DES, denoted by \mathbf{G} , which can be considered as a *dynamic system* equipped with a discrete *state space* and a *state-transition structure*. The *system behavior* is described as a *formal language* $L(\mathbf{G})$ that is generated by an automaton over an *alphabet* (or *event set*, usually finite) denoted by Σ , which models the plant. Generally, event set Σ is partitioned into two disjoint subsets: *controllable events*, denoted by Σ_{con} , that can be disabled by external controllers, and *uncontrollable events*, denoted by Σ_{unc} , that cannot be disabled. With this control mechanism, a *controller* (or *supervisor*) can be used to restrict the system behavior only by disabling controllable events.

Given the desired behavior of a plant \mathbf{G} , i.e., a specification, represented by a formal language E , a supervisor can be designed to restrict \mathbf{G} 's behavior such that the controlled system behavior is a sublanguage of E . Many efforts have been made to synthesize an optimal nonblocking supervisor for DES. Here “*optimal*” means “*minimally restrictive*”.

By incorporating timing features, the notion of *timed DES* (TDES) is proposed in [4]. The *active transition graphs* (ATG), identical with DES diagrams, are assigned with modelling enhancements like program variables, transformations, and transition guards; the *timed transition graphs* (TTG) of TDES are the detailed global transition graphs. The nature of temporal logic is addressed in the languages describing TDES' behavior. For instance, the concept "eventuality", i.e., liveness in the long run, over sets of *infinite strings*, is considered in the supervisory control of TDES, which is also integrated into the TTG construction process starting from ATG.

As a top-down modelling approach, STS are proposed in [42] for the purpose of incorporating the *hierarchical (vertical)* and *concurrent (horizontal) structures* of a complex DES into a compact and natural model. In STS, as a modelling tool, *statecharts* [20] are used to provide a *compact representation* of the *hierarchical transition structure*. The system *transition structures* on successive *layers* are represented by holons (based on [23] and [19]). Thereafter, the STS framework is extended in [29] and [30], which is well-developed to manage the *state explosion problem*. The STS model and its specifications are represented by *predicates*. Prior to calculating the predicate representing the optimal controlled behavior, maximally permissive predicates are presented for disabling controllable events in the transition structure. For controllable events, their *control logics* are represented by *binary decision diagrams* (BDD) [1, 5] that represent a boolean function. A BDD is a *directed acyclic graph* that has two *terminal nodes* 1 and 0 representing *true* and *false*, respectively.

1.3 Real-Time Scheduling Theory

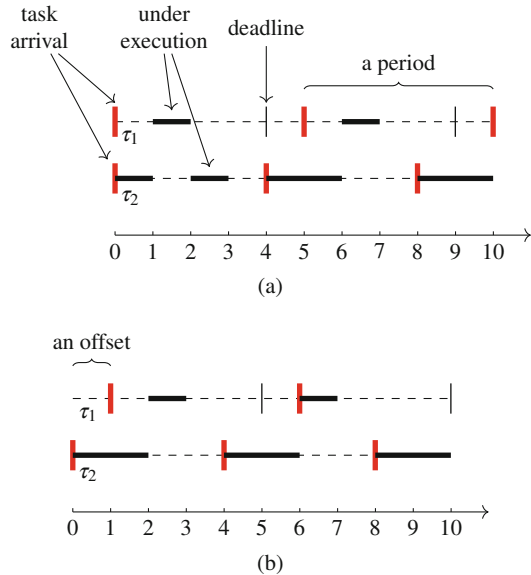
The seminal work of real-time scheduling was proposed by Liu and Layland in the 1970s. Some basic concepts are introduced in this section, and the preliminaries of classical RTS scheduling algorithms and examples are detailed in Chap. 3. A preliminary *real-time task* is time triggered, which satisfies the following facts [39]:

- all the tasks are periodic;
- all the tasks are released at the beginning of periods and have deadlines equal to their periods;
- all the tasks are *independent*, i.e., they have no resource or precedence relationships;
- all the tasks have fixed *execution time*, or at least fixed upper bounds on their computation time, which is less than or equal to their periods;
- no task may voluntarily suspend itself;
- all the tasks are fully preemptable;
- all the overheads are assumed to be zero;
- there is only one processor.

In reality, it is possible that a task has an *offset*, between the *system start-up* and the first release of its *job*. A set of real-time tasks is *synchronous* if they have the same offsets. Otherwise, the tasks in this task set are *asynchronous* [39]. A *periodic task* is associated with a regular *period*, which means that the time interval between any two successive jobs is constant. Every job has a *deadline* that assigns the latest time of its execution to be completed. A task is associated with a *hard deadline* if every job must meet its corresponding deadline. Otherwise, it is associated with a *soft deadline*. A deadline can be expressed relative to the release time (i.e., *relative deadline*) or as an absolute time (i.e., *absolute deadline*). Throughout this monograph, we only consider hard deadlines, i.e., deadlines which must be kept unconditionally. Generally, a task's processing time is between its *best-case execution time* (BCET) and *worst-case execution time* (WCET) [40]. Nassor and Bres propose a task model in [33], with a deadline less than or equal to its period but greater than the WCET. In the theoretical analysis of real-time scheduling, researchers focus on the WCET to guarantee the feasibility of real-time scheduling. The modelling and scheduling of RTS based on the exact execution time is studied in [46]. A uni-processor system is nonschedulable in the case that there is an *overload* [39].

Suppose that two periodic tasks τ_1 and τ_2 are running in an RTS, both are released at the *system start-up*, i.e., time $t = 0$, and their WCETs are equal to one and two, respectively. In the analysis, normally, only WCETs are considered to guarantee the feasibility. The deadline and period for processing τ_1 are four and five, respectively. In parallel, the deadline and period for τ_2 are both four. Two possible schedulings are visualized in Fig. 1.1. Since the tasks depicted in Fig. 1.1a have no offsets, the

Fig. 1.1 Real-time scheduling of periodic tasks.
(a) Tasks without offsets. (b) Task τ_1 with an offset



scheduling is synchronous. Both of τ_1 's relative deadline and absolute deadline are four time units. The deadline of τ_2 is equal to its period, i.e., $D_2 = T_2$.

Suppose that the offset of τ_1 is one time unit, i.e., it is released at time $t = 1$. A possible asynchronous scheduling is shown in Fig. 1.1b. It represents that after the system starts up for one time unit, task τ_1 releases its first job. With its offset assigned, the absolute deadline for the first job of τ_1 is changed from four to five time units.

RTS may also contain tasks that do not have periodic releases or predictable execution time, and may have large variations in the sequential release times or execution time, including *aperiodic tasks* [39], *sporadic tasks* [39], or *non-repetitive tasks* [46]. They are formally introduced in [32] and differ from periodic tasks only in the release time. Clearly, it will be impractical to assign enough processor capacity/resource to guarantee a feasible scheduling. Generally, the *lowest priorities* are assigned to such tasks. Typical application cases are low-priority background operations such as *garbage collection*. The absence of a deadline may lead to *unboundedly postponed execution*.

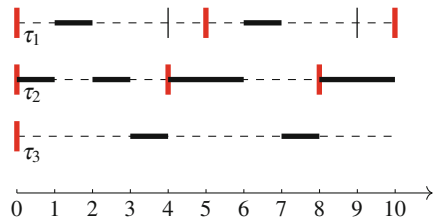
As depicted in Fig. 1.2, which is built on Fig. 1.1a, a non-repetitive task τ_3 with its WCET equal to two is added to the RTS. Task τ_3 is assigned with the lowest priority, i.e., it is under execution only if the executions of the current jobs of τ_1 and τ_2 are complete and before their next arrivals.

According to [17] and [9], *preemptive* and *non-preemptive* real-time scheduling do not depend on any assumed scheduling algorithm. Given a set of real-time tasks, if the execution of a job (belonging to a real-time task) is allowed to be preempted by other (real-time tasks') jobs before its execution finishes, the scheduling is *preemptive*; otherwise, it is *non-preemptive*. The only assumption is that the execution cannot be preempted by *idle operations*.

For example, the real-time scheduling in Fig. 1.1a is preemptive and that in Fig. 1.1b is non-preemptive. At time $t = 1$ in Fig. 1.1b, the execution of task τ_2 is preempted by task τ_1 . However, as illustrated in Fig. 1.1b, no job of task τ_1 is under execution in the time interval [5, 6), and it is illegal to preempt the execution of task τ_2 by idle operations at time $t = 5$.

Generally, the well-known priority-based real-time scheduling algorithms can be divided into two categories: *fixed priorities* (FP) and *dynamic priorities* (DP). The real-time scheduling fails when no schedulable sequence can be found. The priorities of a real-time task set are commonly used to order their accesses to the processor and other shared resources.

Fig. 1.2 Real-time scheduling with a non-repetitive task



In parallel with the seminal work of real-time scheduling, the FP scheduling is proposed in [28]. As a classical fixed priority scheduling algorithm, *rate-monotonic* (RM) scheduling requires that the deadline of a real-time task should be equal to its period. The task with a short period is assigned with the *highest priority*. Later *deadline monotonic* (DM) scheduling is proposed in [27] to schedule RTS that may contain periodic tasks with deadlines less than their periods. A DM policy assigns a higher priority to the tasks with shorter deadlines.

The *feasibility analysis* of periodic task sets, such as *earliest deadline first* (EDF) scheduling, is also proposed in [28], which is usually regarded as the most widely-used DP real-time scheduling algorithm. Under the same simplified assumptions used for RM scheduling, a set of periodic tasks is schedulable by the EDF algorithm, if and only if the processor utilization is less than or equal to one (the processor is fully occupied). In [11], the author shows that, among all preemptive scheduling algorithms, EDF is optimal. If there exists a feasible scheduling for a task set, then the scheduling produced by EDF is also feasible. Under the EDF scheduling, the analysis of periodic tasks with the deadlines less than their periods is proposed in [3].

For example, the real-time scheduling in Fig. 1.1b is EDF scheduling. At time $t = 1$, the absolute deadline of task τ_1 equals five time units and that of τ_2 is four time units. Hence, task τ_2 is currently with a higher priority than τ_1 .

Least laxity first (LLF), as another optimal scheduling algorithm, is proposed by Mok in [32], which assigns the processor to the active task with the smallest laxity. However, LLF has a larger overhead than EDF due to a larger number of context switches caused by laxity changes at run time. Detailed examples can be found in Chap. 3.

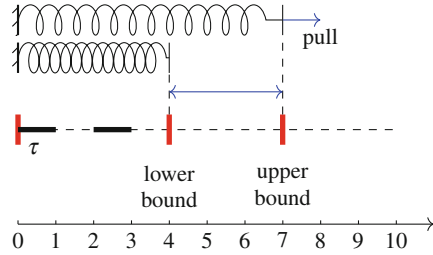
Real-time reconfiguration is of critical importance to RTS. A reconfiguration scenario can be the addition/removal/update of the tasks at run-time in order to save the whole system when *random disturbances* occur. There has been a fair amount of significant work from academia and industry [16, 37, 40, 49, 51] for real-time reconfiguration, which are based on the RM/EDF scheduling with preemptive/non-preemptive schemes. In principle, two types of reconfiguration policies can be identified:

- *fixed policies*: applied *offline* before any system's *cold start* [2], and
- *dynamic policies*: applied at *run-time* [52].

As an RTS reconfiguration approach, *job skipping reconfiguration* [24] can be utilized by an RTS to execute “occasionally skippable” tasks, such as video reception, telecommunications, packet communication, and aircraft control. However, industrial production lines should avoid job skipping since it will increase the manufacturing costs. Another approach, called the elastic scheduling model [6–8, 31], is utilized to guarantee that no deadline is missed during a manufacturing process in industrial applications [18].

In [6–8, 31], an elastic period task model is proposed to handle the overload of an RTS by decreasing its processor utilization via adapting the tasks' periods. As illustrated in Fig. 1.3, the period of a task is assumed with a lower bound and an

Fig. 1.3 Real-time scheduling with an elastic period



upper bound. In the case that an RTS is non-schedulable, a reconfiguration approach is used to enlarge the periods of the tasks with *elastic periods*. This reconfiguration approach deals with such tasks as *springs*. Hence, enlarging the periods is just like pulling a spring. In this monograph, if necessary, we follow a similar approach to reconfigure an RTS.

Several *model checkers*, such as Cheddar¹ [41], Kronos [50], PRISM [25], have been developed to model an RTS and check its schedulability or other performance criteria. This monograph focuses on the real-time scheduling of periodic and sporadic tasks. As Cheddar matches most closely the scheduling problems discussed in this monograph, in what follows, we choose Cheddar as the RTS model checker and scheduler.

1.4 Motivation and Contribution

SCT-based real-time scheduling and reconfiguration [10, 22, 34, 43–47] are a newly-identified research topic. Based on three popular SCT modelling frameworks, TDES [10, 22, 34, 43], DES [44], and STS [47], this monograph provides SCT-based real-time scheduling and reconfiguration mechanisms.

1.4.1 RTS Modelling Methods

The study in the seminal work [10] proposes a TDES-based task model and an SCT-based non-preemptive scheduling scheme. Supervisory control of TDES is able to model and schedule two types of RTS processing:

- resource-sharing tasks, i.e., resources are available to execute multiple tasks concurrently; or
- independent periodic tasks with their deadlines less than or equal to the corresponding periods.

¹ <http://beru.univ-brest.fr/cheddar/>.

Even though the resource-sharing tasks are not as popular as independent tasks, the supervisory control of TDES can provide all the possible safe executions for both in a uniformed approach. An RTS is claimed to be non-schedulable if the supervisor is empty. Thereafter, for the independent periodic tasks, SCT-based real-time scheduling is extended to:

- the scheduling of RTS preemptively or non-preemptively [22],
- the scheduling of RTS processing sporadic tasks [34], and
- the dynamic reconfiguration of RTS when no safe execution sequences are found [43].

In the past two decades, fruitful works on real-time scheduling and reconfiguration are reported, in which RTS are mainly modelled by:

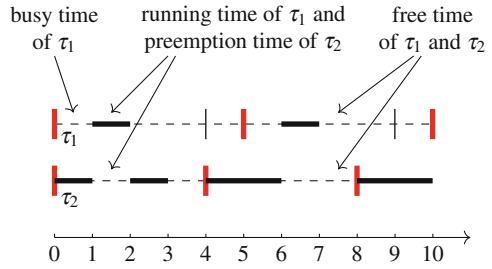
- TDES [12–15, 21, 22, 34–36, 43]: The timing constraints of RTS tasks are represented by time bounds of active events, representing task release/arrival, starting, and finishing. As a consequence, the timing for the idle operations of processors and for the execution of any real-time task is represented by the global clock *tick* (represented by event t).
- DES [44–46]: The execution of each individual RTS task and idle operations are represented by different events in the DES models. Hence, in the DES modelling mechanism, the execution of different tasks will always happen asynchronously. This model is more realistic, i.e., it can describe the sequential execution of real-time tasks in a uni-processor properly. As a natural extension, preemptive real-time scheduling of RTS is studied in [44–47].
- STS [47]: RTS tasks are modelled hierarchically by STS, which are built on a DES model. The task release/arrival, starting, and finishing are on the higher level, and the task execution is on the lower level. This modelling mechanism provides the possibility of assigning DP to the real-time tasks under execution.

We assume that a set of n RTS tasks processed by a uni-processor RTS is represented by a task set $\mathbb{S} = \{\tau_1, \tau_2, \dots, \tau_i, \dots, \tau_n\}$. For periodic real-time scheduling, the *hyper-period* [26] of a set of periodic tasks equals the *least common multiple* of their periods. For simplicity, we consider traditional RTS tasks only. As defined in [44], let $i \in \mathbf{n} := \{1, 2, \dots, n\}$. From the perspective of each individual task τ_i , in each hyper-period, all the processor time units are partitioned into:

- *busy time*: the processor is occupied by other tasks, and thus τ_i cannot be executed;
- *running time*: τ_i is in process;
- *preemption time* (if any): after τ_i has started, its execution is interrupted by (a subset of) other tasks; and
- *free time*: the execution of τ_i is completed or it has not arrived yet. These processor time units can be idle or utilized to execute other tasks.

As depicted in Fig. 1.4, such partitions of time units can be applied to the scheduling shown in Fig. 1.1a as follows:

Fig. 1.4 Processor time units partition



- time interval $[0, 1)$, i.e., time $t = 0$ and the following time unit, is the busy time of task τ_1 and the running time of task τ_2 ;
- time interval $[1, 2)$ is the running time of task τ_1 and preemption time of task τ_2 ;
- time interval $[2, 3)$ is the busy time of task τ_1 and the running time of task τ_2 ;
- time intervals $[3, 4)$ and $[7, 8)$ are the free time of both τ_1 and τ_2 ;
- time intervals $[4, 6)$ and $[8, 10)$ are both the running time of task τ_2 and the busy time of task τ_1 ; and
- time interval $[6, 7)$ is the running time of τ_1 and busy time of task τ_2 .

The preemption scheduling addressed in this monograph is on two levels: from the perspective of the processor and individual task. Generally, a *processor-preemption relation* assigns the priority to tasks without considering their execution details. Once a task is under execution, all the tasks with higher priorities can preempt its execution. However, from the perspective of an individual real-time task, a *task-preemption relation* is depicted by DES specifications directly in order to assign the exact preemption relation at the exact relative execution time. The presented two general conditional-preemption specifications are utilized to customize scheduling and preemption requirements.

The research in [44] and [45] shows that, on the processor level, both preemptive and non-preemptive priority scheduling policies may be conservative. As a solution, a matrix-based *priority-free conditionally-preemptive* (PFCP, detailed in Chap. 5) scheduling policy is developed, which generalizes the priority-based preemption, such as:

- non-preemptive [10, 12–15, 21, 22, 35, 36, 43],
- preemptive [13, 34], and
- both non-preemptive and preemptive [44–47].

In other words, a priority-free scheduling policy can be utilized to schedule all the periodic tasks randomly, i.e., for a real-time task, its busy time and preemption time can be occupied by any other tasks.

On the processor level, most studies of SCT-based real-time scheduling focus on FP or PFCP scheduling. To the best of our knowledge, the work in Chap. 7 is the first attempt to schedule RTS with DP. This means that the preemption relation defined in Chap. 7 is two-fold:

- The developed scheduling framework can find the optimal behavior (all the safe execution sequences) of an RTS by the supervisory control of STS.
- A few sequences are selected, which rank at the top according to some specified optimality criteria. For example, such an optimality criterion could be defined according to EDF: at any time, only the sequences executing tasks with the shortest deadlines are retained.

A controller for each controllable event of the STS is obtained by the supervisory control of STS, which provides the expected safe execution sequences.

For the purpose of dynamic RTS reconfiguration, multi-period periodic task models are proposed in Chaps. 4, 6, and 7, which can be utilized to model RTS tasks with periods varying between a lower bound and an upper bound. In the light of the *elastic period model*, the main idea of a multi-period is to assign all the possible periods between its lower bound and upper bound. For the real-time task shown in Fig. 1.3, in the model of TDES, DES or STS, we assign four possible periods to it: four, five, six, and seven. The default period for a task is the shortest one. In the case that the RTS is non-schedulable, based on nonblocking supervisory control, the multi-period is used to reconfigure the RTS automatically. Generally, a traditional RTS task is viewed as special cases of the corresponding multi-period models. The only difference of a task's model before and after its reconfiguration is the upper bound of its multi-period.

1.4.2 RTS Scheduling and Reconfiguration

For an RTS represented by TDES, DES, or STS, its optimal supervisor is synthesized, which provides all the possible scheduling/reconfiguration sequences. Any of the scheduling plans embodied in the supervisor can be utilized to schedule/reconfigure the RTS. In these sequences, all the possible DP or FP sequences are included. If the supervisor is empty, the RTS is claimed to be non-schedulable. Otherwise, the users can choose any sequence to schedule the RTS. In particular, as stated in Chap. 7, based on the supervisory control of STS, a few sequences are selected, which rank at the top according to some specified (dynamic) optimality criteria.

In this monograph, a reconfigurable real-time task τ_i is assigned with a multi-period. By modelling such RTS tasks using TDES, DES, or STS, with dynamic reconfiguration integrated, all the safe execution sequences (possible reconfiguration scenarios) are found by supervisory control.

The SCT-based reconfiguration process of RTS in this monograph is illustrated in Fig. 1.5, which is a two-step approach. At the first step, we select the RTS task models running in the same processor and their corresponding specifications (denoted by “execution spec”), which are followed by calculating the synchronous products using procedure **sync** (introduced in [48]); thereafter, the supervisor is calculated by **supcon** (introduced in [48]). If the supervisor is empty and the

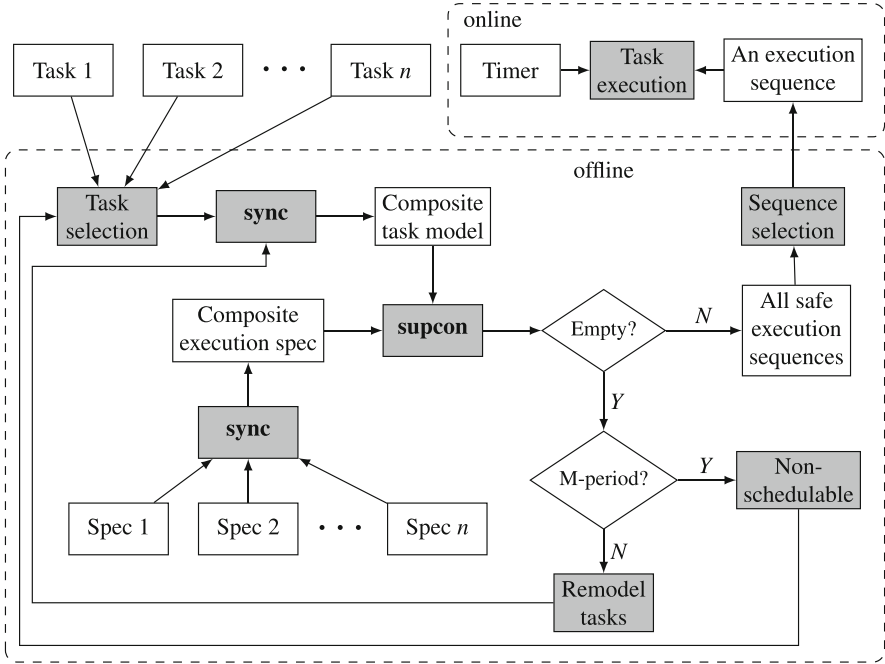


Fig. 1.5 Procedures for real-time scheduling

tasks are not running in the multi-period model (denoted by “M-period”), we will reconfigure the tasks and repeat the scheduling process at the second step. Finally, the users can select a safe execution sequence from the supervisor to schedule the RTS online.

Suppose that in every scheduling plan only a subset of tasks executed by an RTS enters the uni-processor for execution. Initially the tasks are running in the periodic version with lower bounds. If no safe execution sequence can be found at the highest processor utilization, SCT is utilized to provide all the possible safe execution sequences by offline supervisory control. Hence, during the reconfiguration process, the exact processor utilization of any task assigned with multi-period lies between its lower and upper bounds.

By SCT, all possible safe execution sequences are found, resulting in a decrease of processor utilization. The users should take the responsibility to provide the tolerable lowest processor utilization. Consequently, any safe execution sequence in the supervisor can be selected as a guide to schedule the RTS by dynamically reconfiguring the period of a task. If the supervisor is still empty, we claim that the system is non-schedulable.

For industrial production lines or manufacturing processes, the technique presented in this monograph reconfigures an RTS that executes a set of tasks with the

same task scale studied in [6–8, 10, 22, 31]. We suggest that the users predefine an acceptable processor utilization interval for every task.

For SCT, a method for speeding up the calculation is to reduce the number of states in the plant and specification. The presented synthesis speeding up approach can be applied to Chaps. 5 and 6. In this book, we mainly divide the calculations into three steps. Each step considers different specifications as follows.

- Step 1: Spec 1 (S1), from the perspective of processors, *PFCP specifications* are touched upon;
- Step 2: Spec 2 (S2), from the perspective of individual tasks, *task-centered conditional-preemption specifications* are considered; and
- Step 3: Spec 3 (S3), other user defined specifications are taken into account.

As the commutative diagram shown in Fig. 1.6, PS, TS, and OS represent PFCP specifications, task-centered conditional-preemption specifications, and other user defined specifications, respectively. In Fig. 1.6, the synthesis steps represented by thick lines can speed up the synthesis process. According to [29] and [30], the STS framework (rooted in BDD) is well-developed to manage the state explosion problem. Hence, the presented “speeding up” approach is not so necessary for STS-based real-time scheduling and reconfiguration.

According to [48], the software packages² TCT and TTCT are developed to create DES and TDES generators, respectively. Moreover, STS can also be synthesized in a software package STSLib³, which utilizes BDD as the basis for efficient computation. The procedures TCT/TTCT utilized in this monograph can be found in [48]. All the operations and the generated files are recorded in an annotated file MAKEIT.TXT.

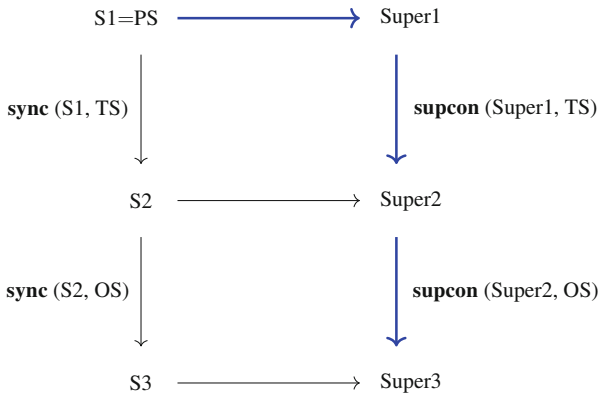


Fig. 1.6 A commutative diagram

² <http://www.control.utoronto.ca/DES>.

³ <https://github.com/chuanma/STSLib>.

1.5 Monograph Outline

The remainder of this monograph is organized as follows. Chapter 2 reports the preliminaries of SCT and three DES modelling frameworks: DES, TDES, and STS. In this monograph, they are all utilized to model the behavior of RTS. By SCT, their expected safe execution sequences are synthesized with user-defined specifications taken into account.

From the perspective of RTS scheduling, Chap. 3 reviews the real-time scheduling and reconfiguration of periodic RTS. Moreover, basic concepts such as classical DP scheduling, FP scheduling, and elastic period task models are presented. These models and scheduling algorithms will be represented and handled by DES, TDES, and STS in the rest of this monograph.

Chapter 4 reviews the seminal work on SCT-based real-time scheduling. Thereafter, based on it to dynamically reconfigure RTS, a new formalism is presented to assign multi-periods to periodic tasks. For a periodic task, its release/arrival, starting, and finishing are represented by active events, and their timing constraints are converted to the time bounds of such events. Thus, the dynamics of a real-time task is depicted in a TTG. A multi-period task is assigned with a set of possible periods between a minimum period and a maximum period. Initially, a task is assigned with the shortest period, which can be viewed as a special case of the corresponding multi-period. By implementing SCT, an RTS is dynamically reconfigured when its initial safe execution sequence set is empty. During the real-time scheduling/reconfiguration process, the supervisor proposes all the safe execution sequences.

Chapter 5 points out that DES are more general for modelling RTS than TDES, and provide the possibility of preemptive SCT-based scheduling of RTS. A DES-based periodic real-time task modelling method is presented. The timing constraints of RTS tasks are represented by different events. A preemption policy, namely conditional-preemption, is presented. On the processor level, the task preemption relations are described by preemption matrices. Thereafter, DES specifications are designed accordingly. On the task level, the task preemption relations are depicted by DES specifications directly. The presented preemption relation generalizes priority-based preemption. In fact, for some real-time scheduling requirements, priorities cannot be assigned to real-time tasks. Considering the processor behavior related to each individual task's execution and the user-defined specifications, by implementing supervisory control of DES, we synthesize a supervisor which provides all the safe real-time execution sequences. Based on this idea, by considering the exact execution time of real-time tasks, a general modular DES model representing RTS tasks is presented in Chap. 6.

In the light of the multi-period reported in Chap. 4 and building on Chap. 5, a DES version modular multi-period is presented in Chap. 6. For the purpose of integrating real-time scheduling and reconfiguration into a uniformed framework, a multi-period model is presented, which contains a set of possible periods. A task is represented by an automaton synchronized by the required modular models, in

which a multi-period task is assigned with a set of possible periods between a minimum period and a maximum period. The only difference of a task's model before and after its reconfiguration is the upper bound of its multi-period. The DES model depicting the RTS is synchronized by the DES representing these tasks. As a consequence, we introduce the main contributions without distinguishing real-time scheduling and reconfiguration.

A hierarchical RTS model is presented in Chap. 7, based on nonblocking supervisory control of STS, where both conditionally-preemptive and DP scheduling are addressed in the SCT-based real-time scheduling. This chapter reports on a unified STS-based framework to model and schedule RTS by addressing PFCP and DP. A formal constructive method is presented to model an RTS that processes multi-period and sporadic tasks, in which a multi-period task is assigned with a set of possible periods between a minimum period and a maximum period. The proposed modular models are taken to be generic entities, which are utilized to model a problem domain such as "hard real-time manufacturing and reconfigurations" and manage its manufacturing production process.

Finally, Chap. 8 concludes the contributions of this monograph and proposes some possible future extensions. The main differences among the three SCT modelling frameworks are also discussed in Chap. 8. Through an RTS example, this chapter shows that the core of all the presented modelling approaches is identical: the real-time tasks' behavior is represented by formal languages that are generated by TDES, DES, or the holons in STS. Thereafter, for either real-time scheduling or dynamic reconfiguration, SCT is utilized to find out the safe execution sequences.

References

1. Andersen, H.R.: An Introduction to Binary Decision Diagrams. Lecture Notes, IT University of Copenhagen (1997). http://web.archive.org/web/20140222052815/http://configit.com/configit_wordpress/wp-content/uploads/2013/07/bdd-eap.pdf
2. Angelov, C., Sierszecki, K., Marian, N.: Design models for reusable and reconfigurable state machines. In: International Conference on Embedded and Ubiquitous Computing, pp. 152–163. Springer, Berlin (2005)
3. Baruah, S.K., Rosier, L.E., Howell, R.R.: Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Syst.* **2**(4), 301–324 (1990)
4. Brandin, B.A., Wonham, W.M.: Supervisory control of timed discrete-event systems. *IEEE Trans. Autom. Control* **39**(2), 329–342 (1994)
5. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.* **100**(8), 677–691 (1986)
6. Buttazzo, G., Abeni, L.: Adaptive workload management through elastic scheduling. *Real-Time Syst.* **23**, 7–24 (2002)
7. Buttazzo, G.C., Lipari, G., Abeni, L.: Elastic task model for adaptive rate control. In: *IEEE Real-Time Systems Symposium*, pp. 286–295 (1998)
8. Buttazzo, G.C., Lipari, G., Caccamo, M., Abeni, L.: Elastic scheduling for flexible workload management. *IEEE Trans. Comput.* **51**(3), 289–302 (2002)

9. Buttazzo, G.C., Bertogna, M., Yao, G.: Limited preemptive scheduling for real-time systems: a survey. *IEEE Trans. Ind. Inf.* **9**(1), 3–15 (2013)
10. Chen, P.C.Y., Wonham, W.M.: Real-time supervisory control of a processor for non-preemptive execution of periodic tasks. *Real-Time Syst.* **23**, 183–208 (2002)
11. Dertouzos, M.L.: Control robotics: the procedural control of physical processes. In: *IF IP Congress*, pp. 807–813 (1974)
12. Devaraj, R., Sarkar, A., Biswas, S.: Fault-tolerant preemptive aperiodic RT scheduling by supervisory control of TDES on multiprocessors. *ACM Trans. Embed. Comput. Syst.* **16**(3), 1–25 (2017)
13. Devaraj, R., Sarkar, A., Biswas, S.: Fault-tolerant scheduling of non-preemptive periodic tasks using SCT of timed DES on uniprocessor systems. *IFAC-PapersOnLine* **50**(1), 9315–9320 (2017)
14. Devaraj, R., Sarkar, A., Biswas, S.: Real-time scheduling of non-preemptive sporadic tasks on uniprocessor systems using supervisory control of timed DES. In: *American Control Conference*, pp. 3212–3217. *IEEE* (2017)
15. Devaraj, R., Sarkar, A., Biswas, S.: Supervisory control approach and its symbolic computation for power-aware RT scheduling. *IEEE Trans. Ind. Inf.* **15**(2), 787–799 (2018)
16. Gaujal, B., Navet, N.: Dynamic voltage scaling under EDF revisited. *Real-Time Syst.* **37**(1), 77–97 (2007)
17. George, L., Voluteau, D.D., France, B.L.C.C.: Preemptive and non-preemptive real-time uniprocessor scheduling. *INRIA Res. Rep.* **2966** (1996)
18. Girbea, A., Suci, C., Nechifor, S., Sisak, F.: Design and implementation of a service-oriented architecture for the optimization of industrial applications. *IEEE Trans. Ind. Inf.* **10**(1), 185–196 (2014)
19. Gou, L., Hasegawa, T., Luh, P.B., Tamura, S., Oblak, J.M.: Holonic planning and scheduling for a robotic assembly testbed. In: *International Conference on Computer Integrated Manufacturing and Automation Technology*, pp. 142–149. *IEEE* (1994)
20. Harel, D.: Statecharts: a visual formalism for complex systems. *Sci. Comput. Program.* **8**(3), 231–274 (1987)
21. Janarthanan, V., Gohari, P.: Multiprocessor scheduling in supervisory control of discrete-event systems framework. *Control Intell. Syst.* **35**(4), 360 (2007)
22. Janarthanan, V., Gohari, P., Saffar, A.: Formalizing real-time scheduling using priority-based supervisory control of discrete-event systems. *IEEE Trans. Autom. Control* **51**(6), 1053–1058 (2006)
23. Koestler, A.: *The Ghost in the Machine*. Henry Regnery, Washington (1989)
24. Koren, G., Shasha, D.: Skip-over: algorithms and complexity for overloaded systems that allow skips. In: *IEEE Real-Time Systems Symposium*, pp. 110–117 (1995)
25. Kwiatkowska, M., Norman, G., Parker, D.: *PRISM 4.0: verification of probabilistic real-time systems*. In: *International Conference on Computer Aided Verification*, pp. 585–591. Springer, Berlin (2011)
26. Leung, J.Y.T., Merrill, M.L.: A note on preemptive scheduling of periodic, real-time tasks. *Inf. Process. Lett.* **11**(3), 115–118 (1980)
27. Leung, J.Y.T., Whitehead, J.: On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Perform. Eval.* **2**(4), 237–250 (1982)
28. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM* **20**(1), 46–61 (1973)
29. Ma, C., Wonham, W.M.: *Nonblocking Supervisory Control of State Tree Structures*, vol. 317. Springer, Berlin (2005)
30. Ma, C., Wonham, W.M.: Nonblocking supervisory control of state tree structures. *IEEE Trans. Autom. Control* **51**(5), 782–793 (2006)
31. Marinoni, M., Buttazzo, G.: Elastic DVS management in processors with discrete voltage/frequency modes. *IEEE Trans. Ind. Inf.* **3**(1), 51–62 (2007)

32. Mok, A.K.: Fundamental design problems of distributed systems for the hard-real-time environment. Ph.D. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA (1983)
33. Nasser, E., Bres, G.: Hard real-time sporadic task scheduling for fixed priority schedulers. In: International Workshop on Responsive Systems, pp. 44–47 (1991)
34. Park, S.J., Cho, K.H.: Real-time preemptive scheduling of sporadic tasks based on supervisory control of discrete event systems. *Inf. Sci.* **178**, 3393–3401 (2008)
35. Park, S.J., Cho, K.H.: Supervisory control for fault-tolerant scheduling of real-time multiprocessor systems with aperiodic tasks. *Int. J. Control* **82**(2), 217–227 (2009)
36. Park, S.J., Yang, J.M.: Supervisory control for real-time scheduling of periodic and sporadic tasks with resource constraints. *Automatica* **45**(11), 2597–2604 (2009)
37. Quan, G., Hu, X.S.: Minimal energy fixed-priority scheduling for variable voltage processors. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **22**(8), 1062–1071 (2003)
38. Ramadge, P.J., Wonham, W.M.: Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.* **25**(1), 206–230 (1987)
39. Sha, L., Abdelzaher, T., Cervin, A., Baker, T., Burns, A., Buttazzo, G., Caccamo, M., Lehoczky, J., Mok, A.K.: Real time scheduling theory: a historical perspective. *Real-Time Syst.* **28**(2), 101–155 (2004)
40. Shin, Y., Choi, K.: Power conscious fixed priority scheduling for hard real-time systems. In: Design Automation Conference, pp. 134–139. IEEE, New Orleans (1999)
41. Singhoff, F., Legrand, J., Nana, L., Marcé, L.: Cheddar: a flexible real time scheduling framework. *ACM SIGAda Ada Lett.* **4**, 1–8 (2004)
42. Wang, B.: Top-down design for RW supervisory control theory. Master’s Thesis, Department of Electrical and Computer Engineering, University of Toronto (1996)
43. Wang, X., Li, Z., Wonham, W.M.: Dynamic multiple-period reconfiguration of real-time scheduling based on timed DES supervisory control. *IEEE Trans. Ind. Inf.* **12**(1), 101–111 (2016)
44. Wang, X., Li, Z., Wonham, W.M.: Optimal priority-free conditionally-preemptive real-time scheduling of periodic tasks based on DES supervisory control. *IEEE Trans. Syst. Man Cybern. Syst.* **47**, 1082–1098 (2017)
45. Wang, X., Li, Z., Wonham, W.M.: Priority-free conditionally-preemptive scheduling of modular sporadic real-time systems. *Automatica* **89**, 392–397 (2018)
46. Wang, X., Li, Z., Moor, T.: SCT-based priority-free conditionally-preemptive scheduling of modular real-time systems with exact task execution time. *Discrete Event Dyn. Syst. Theory Appl.* **29**, 501–520 (2019)
47. Wang, X., Li, Z., Wonham, W.M.: Real-time scheduling based on nonblocking supervisory control of state-tree structures. *IEEE Trans. Autom. Control* **66**(9), 4230–4237 (2021)
48. Wonham, W.M., Cai, K.: Supervisory Control of Discrete-Event Systems. Monograph Series Communications and Control Engineering, Springer, Berlin (2018)
49. Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. In: Annual Symposium on Foundation Computer Science, pp. 374–382 (1995)
50. Yovine, S.: Kronos: a verification tool for real-time systems. *Int. J. Softw. Tools Technol. Transf.* **1**(1–2), 123–133 (1997)
51. Yun, H., Kim, J.: On energy-optimal voltage scheduling for fixed-priority hard real-time systems. *ACM Trans. Embed. Comput. Syst.* **2**(3), 393–430 (2003)
52. Zhang, J., Khalgui, M., Li, Z., Mosbahi, O., Alahmari, A.M.: R-TNCES: a novel formalism for reconfigurable discrete event control systems. *IEEE Trans. Syst. Man Cybern. Syst.* **43**(4), 757–772 (2013)

Chapter 2

Preliminaries of Supervisory Control Theory



2.1 Discrete-Event Systems

In the *language-based Ramdge-Wonham (R-W) framework* [13, 17], *discrete-event systems (DES)* are represented by automata. A finite state DES *plant* is a *generator*

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

where

- Q is the *finite state set*,
- Σ is the *finite event set (alphabet)*, partitioned into the disjoint *controllable event* subset Σ_{con} and the *uncontrollable event* subset Σ_{unc} , i.e.,

$$\Sigma = \Sigma_{con} \dot{\cup} \Sigma_{unc},$$

- $\delta : Q \times \Sigma \rightarrow Q$ is the *partial state transition function*,
- q_0 is the *initial state*, and
- $Q_m \subseteq Q$ is the subset of *marker states*.

In accordance with [17], Σ^+ denotes the set of all *finite sequences* that consists of the events in Σ . By adjoining the *empty string* ϵ , the set of finite strings over the alphabet Σ is written as Σ^* , i.e.,

$$\Sigma^* = \Sigma^+ \cup \{\epsilon\}.$$

The operation of *catenation* of strings

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

is defined as

$$\text{cat}(\epsilon, s) = \text{cat}(s, \epsilon) = s, s \in \Sigma^*$$

and

$$\text{cat}(s, t) = st, s, t \in \Sigma^+.$$

Operation $\text{cat}(\cdot, \cdot)$ is associative, i.e.,

$$\text{cat}(\text{cat}(s, t), u) = \text{cat}(s, \text{cat}(t, u)), s, t, u \in \Sigma^+.$$

Function δ can be extended to

$$\delta : Q \times \Sigma^* \rightarrow Q$$

by defining $\delta(q, \epsilon) = q$ and $\delta(q, s\sigma) = \delta(\delta(q, s), \sigma)$, where $q \in Q$ is a state and $s \in \Sigma^*$ is a string. Write $\delta(q, s)!$ if $\delta(q, s)$ is defined. The *length* of a string $s \in \Sigma^*$, denoted by $|s|$, is defined below.

$$|s| = \begin{cases} 0, & \text{if } s = \epsilon \\ k, & \text{if } s = \sigma_1\sigma_2\cdots\sigma_k \in \Sigma^+ \end{cases}$$

For $t \in \Sigma^*$, we say $s \in \Sigma^*$ is a *prefix* of t . Write $s \leq t$, if $t = su$ for some $u \in \Sigma^*$. Clearly $\epsilon \leq t$ and $t \leq t$ for all $t \in \Sigma^*$. A *language* over Σ is any subset of Σ^* , i.e., an element of the *power set* $Pwr(\Sigma^*)$. The *closed behavior* of \mathbf{G} is represented by

$$L(\mathbf{G}) := \{s \in \Sigma^* \mid \delta(q_0, s)!\} \quad (2.1)$$

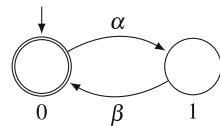
and the *marked behavior* is represented by

$$L_m(\mathbf{G}) := \{s \in L(\mathbf{G}) \mid \delta(q_0, s) \in Q_m\} \subseteq L(\mathbf{G}). \quad (2.2)$$

The (*prefix*) *closure* of $L_m(\mathbf{G})$ is denoted by $\overline{L_m(\mathbf{G})}$. A DES is *nonblocking* if $L(\mathbf{G}) = \overline{L_m(\mathbf{G})}$. In the *transition graph* describing a DES, the initial state is labelled with an *entering arrow*, and a marker state is represented by a *double circle*.

Example A DES generator $\mathbf{G}_1 = (Q_1, \Sigma_1, \delta_1, q_{0,1}, Q_{m,1})$ is depicted in Fig. 2.1, in which

- $Q_1 = \{0, 1\}$,
- $\Sigma_1 = \{\alpha, \beta\}$ with $\Sigma_{1,con} = \{\alpha\}$ and $\Sigma_{1,unc} = \{\beta\}$,
- $\delta_1(0, \alpha) = 1$ and $\delta_1(1, \beta) = 0$,
- state 0 is the initial state, and
- state set $\{0\}$ is the marker state subset.

Fig. 2.1 DES generator \mathbf{G}_1 

The closed behavior of \mathbf{G}_1 is

$$L(\mathbf{G}_1) = \{\epsilon, \alpha, \alpha\beta, \alpha\beta\alpha, \dots\} = \{(\alpha\beta)^*, (\alpha\beta)^*\alpha\},$$

where

$$(\alpha\beta)^* = \{\epsilon, (\alpha\beta)^1, (\alpha\beta)^2, \dots\},$$

and the marked behavior of \mathbf{G}_1 is

$$L_m(\mathbf{G}_1) = \{\epsilon, \alpha\beta, \alpha\beta\alpha\beta, \dots\} = \{(\alpha\beta)^*\}.$$

We have $L(\mathbf{G}_1) = \overline{L_m(\mathbf{G}_1)} = \{\epsilon, \alpha, \alpha\beta, \alpha\beta\alpha, \alpha\beta\alpha\beta, \dots\}$, and the DES depicted in Fig. 2.1 is nonblocking. \square

Synchronous product [17] is a standard approach to combine a finite set of DES into a single and more complex one. Suppose that there are n languages $L_i \subseteq \Sigma_i^*$ corresponding to n DES, respectively, with

$$\Sigma = \bigcup_{i \in \mathbf{n}} \Sigma_i, \quad \mathbf{n} := \{1, 2, \dots, n\}.$$

The *natural projection* $P_i : \Sigma^* \rightarrow \Sigma_i^*$ is defined by

- $P_i(\epsilon) = \epsilon$,
- $P_i(\sigma) = \begin{cases} \epsilon, & \text{if } \sigma \notin \Sigma_i \\ \sigma, & \text{if } \sigma \in \Sigma_i \end{cases}$, and
- $P_i(s\sigma) = P_i(s)P_i(\sigma)$, $s \in \Sigma^*$, $\sigma \in \Sigma$.

The *inverse image function* of P_i is

$$P_i^{-1} : Pwr(\Sigma_i^*) \rightarrow Pwr(\Sigma^*),$$

where $Pwr(\Sigma^*)$ denotes the power set of Σ^* . For $H \subseteq \Sigma_i^*$, we have

$$P_i^{-1}(H) := \{s \in \Sigma^* \mid P_i(s) \in H\}. \quad (2.3)$$

The *synchronous product* of a family of languages L_1, L_2, \dots , and L_n , denoted by $L_1 || L_2 || \dots || L_n$, is defined as

$$L_1 || L_2 || \cdots || L_n := P_1^{-1}L_1 \cap P_2^{-1}L_2 \cap \cdots \cap P_n^{-1}L_n. \quad (2.4)$$

Example Suppose that another DES generator $\mathbf{G}_2 = (Q_2, \Sigma_2, \delta_2, q_{0,2}, Q_{m,2})$ is depicted in Fig. 2.2, in which

- $Q_2 = \{0, 1, 2\}$,
- $\Sigma_2 = \{\beta, \lambda\}$ with $\Sigma_{2,con} = \{\lambda\}$ and $\Sigma_{2,unc} = \{\beta\}$,
- $\delta_2(0, \beta) = 1$, $\delta_2(1, \beta) = 2$, and $\delta_2(2, \lambda) = 0$,
- state 0 is the initial state, i.e., $q_{0,2} = 0$, and
- state set $\{0\}$ is the subset of marker states, i.e., $Q_{m,2} = \{0\}$.

By computing the synchronous product of \mathbf{G}_1 and \mathbf{G}_2 , we have the global event set as

$$\Sigma = \Sigma_1 \cup \Sigma_2 = \{\alpha, \beta, \lambda\}.$$

The automata representing $P_1^{-1}L_1$ and $P_2^{-1}L_2$ are depicted in Fig. 2.3, in which the newly added events in the automata are represented by *selfloops*. Finally, the synchronous product of \mathbf{G}_1 and \mathbf{G}_2 is

$$L(\mathbf{G}) = L(\mathbf{G}_1 || \mathbf{G}_2) = P_1^{-1}L_1 \cap P_2^{-1}L_2.$$

The DES diagram corresponding to \mathbf{G} is depicted in Fig. 2.4. □

Fig. 2.2 DES generator \mathbf{G}_2

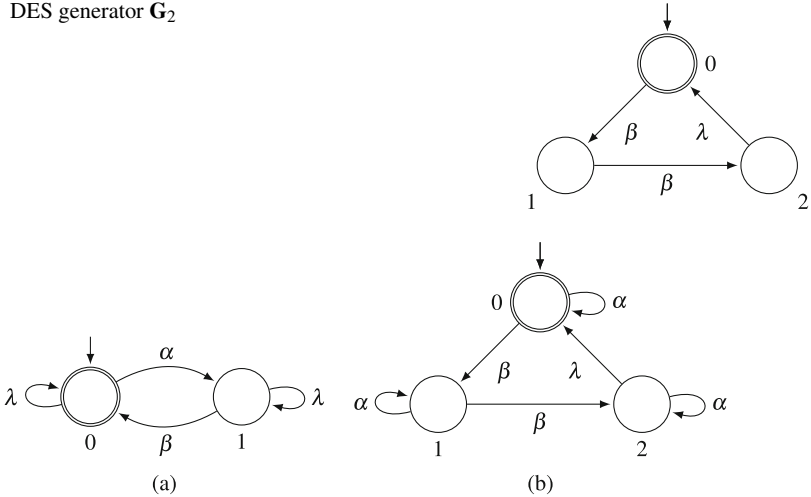


Fig. 2.3 Two DES generators with selfloops. (a) DES representing $P_1^{-1}L_1$. (b) DES representing $P_2^{-1}L_2$

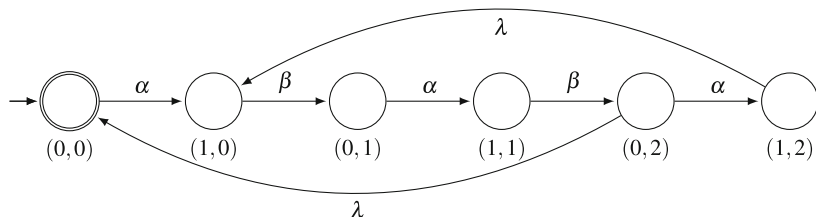


Fig. 2.4 Synchronous product $\mathbf{G} = \mathbf{G}_1 \parallel \mathbf{G}_2$

The DES synthesis tool TCT¹ is a software package that provides several procedures for DES. The synchronous product $L(\mathbf{G}) = L(\mathbf{G}_1 \parallel \mathbf{G}_2)$ can be calculated in TCT as listed below, in which events α , β , and λ are represented by integers 1, 2, and 3, respectively. The utilized procedures are introduced in [17].

$$\mathbf{G1} = \mathbf{create} (\mathbf{G1}, [\mathbf{mark} \ 0], [\mathbf{tran} \ [0, 1, 1], [1, 2, 0]]) (2, 2)$$

$$\mathbf{G2} = \mathbf{create} (\mathbf{G2}, [\mathbf{mark} \ 0], [\mathbf{tran} \ [0, 2, 1], [1, 2, 2], [2, 3, 0]]) (3, 3)$$

$$\mathbf{T1} = \mathbf{selfloop} (\mathbf{G1}, [3]) (2, 4)$$

$$\mathbf{T2} = \mathbf{selfloop} (\mathbf{G2}, [1]) (3, 6)$$

$$\mathbf{G} = \mathbf{meet} (\mathbf{T1}, \mathbf{T2}) (6, 7)$$

TCT also provides a procedure **sync** to calculate the synchronous product of up to 20 DES directly. In TCT, the synchronous product of \mathbf{G}_1 and \mathbf{G}_2 , denoted by TEST, is calculated below.

$$\mathbf{TEST} = \mathbf{sync} (\mathbf{G1}, \mathbf{G2}) (6, 7)$$

Finally, the identity of the two approaches above is verified by

$$\mathbf{true} = \mathbf{isomorph} (\mathbf{G}, \mathbf{TEST}; \mathbf{identity}),$$

which shows that the synchronous products obtained by the two approaches given above are identical. \square

Suppose that a DES model is nonempty. Under *supervisory control*, all the uncontrollable events are automatically enabled. After adjoining a particular subset of the controllable events to be enabled, a set of *control patterns* is defined as

$$\Phi = \{\phi \in Pwr(\Sigma) \mid \phi \supseteq \Sigma_{unc}\}. \quad (2.5)$$

¹ <http://www.control.utoronto.ca/DES>.

The supervisory control for \mathbf{G} is any map

$$V : L(\mathbf{G}) \rightarrow \Phi.$$

DES \mathbf{G} under the supervision of V is written as V/\mathbf{G} . The *closed behavior* of V/\mathbf{G} is defined to be $L(V/\mathbf{G}) \subseteq L(\mathbf{G})$ described as

- the empty string $\epsilon \in L(V/\mathbf{G})$,
- if $s \in L(V/\mathbf{G})$, $\sigma \in V(s)$, and $s\sigma \in L(\mathbf{G})$, then $s\sigma \in L(V/\mathbf{G})$, and
- no other strings belong to $L(V/\mathbf{G})$.

The *marked behavior* of V/\mathbf{G} is denoted by

$$L_m(V/\mathbf{G}) = L(V/\mathbf{G}) \cap L_m(\mathbf{G}). \quad (2.6)$$

The *control map* V is said to be *nonblocking* for \mathbf{G} if

$$\overline{L_m(V/\mathbf{G})} = L(V/\mathbf{G}).$$

A language $K \subseteq \Sigma^*$ is said to be *controllable* (with respect to \mathbf{G}) if

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

i.e.,

$$(\forall s \in \Sigma^*)(\forall \sigma \in \Sigma)s \in \overline{K} \ \& \ \sigma \in \Sigma_{unc} \ \& \ s\sigma \in L(\mathbf{G}) \Rightarrow s\sigma \in \overline{K}.$$

Let $K \subseteq L(\mathbf{G})$ be nonempty and closed. There exists a supervisory control V for \mathbf{G} such that

$$L(V/\mathbf{G}) = K$$

iff K is *controllable* with respect to \mathbf{G} ; this is referred to as a *nonblocking supervisory control* (NSC). Generally, if *marking* is also considered, then we select a sublanguage $M \subseteq L_m(\mathbf{G})$. A *marking NSC* (MNSC) with respect to \mathbf{G} exists if it is a map $V : L(\mathbf{G}) \rightarrow \Phi$ satisfying the behavior

$$L_m(V/\mathbf{G}) = L(V/\mathbf{G}) \cap M. \quad (2.7)$$

Suppose that a *specification language* is given by $E \subseteq \Sigma^*$. Let $\mathcal{C}(E)$ be the family of *sublanguages* of E that are controllable with respect to \mathbf{G} . $\mathcal{C}(E)$ is nonempty and is closed under arbitrary unions. Since $\emptyset \subseteq E$, the (unique) *supremal element* within $\mathcal{C}(E)$, denoted by $\sup \mathcal{C}(E)$, always exists.

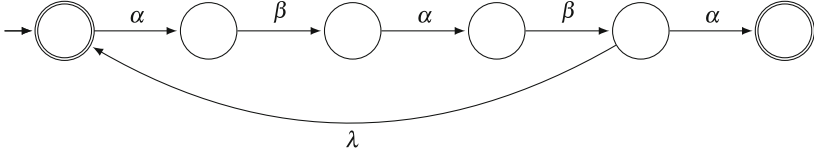


Fig. 2.5 Specification

Example For the DES \mathbf{G} portrayed in Fig. 2.4, we have

- $L(\mathbf{G}) = \{\epsilon, \alpha, \alpha\beta, \alpha\beta\alpha, \alpha\beta\alpha\beta, \alpha\beta\alpha\beta\lambda, \alpha\beta\alpha\beta\alpha, \dots\}$, and
- $L_m(\mathbf{G}) = L_m(\mathbf{G}_1) \parallel L_m(\mathbf{G}_2) = \{(\alpha\beta)^*\} \parallel \{(\beta\beta\lambda)^*\} = \{(\alpha(\beta\alpha\beta\alpha\lambda)^*\beta\alpha\beta\lambda)^*\}$.

Now we assign a specification \mathbf{S} as shown in Fig. 2.5. It satisfies $L(\mathbf{S}) = \overline{L_m(\mathbf{S})}$ and

$$L_m(\mathbf{S}) = \{(\alpha\beta\alpha\beta\lambda)^*, (\alpha\beta\alpha\beta\lambda)^*\alpha\beta\alpha\beta\alpha\}.$$

Let $E = L(\mathbf{S})$. First, we check the *controllability* of $E_1 = \{\alpha\beta\alpha\beta\alpha\} \subset E$ with $\Sigma_{unc} = \{\beta\}$ as follows:

- $\overline{E_1} = \{\epsilon, \alpha, \alpha\beta, \alpha\beta\alpha, \alpha\beta\alpha\beta, \alpha\beta\alpha\beta\alpha\}$,
- $\overline{E_1}\Sigma_{unc} = \{\beta, \alpha\beta, \alpha\beta\beta, \alpha\beta\alpha\beta, \alpha\beta\alpha\beta\beta, \alpha\beta\alpha\beta\alpha\beta\}$, and
- $\overline{E_1}\Sigma_{unc} \cap L(\mathbf{G}) = \{\alpha\beta, \alpha\beta\alpha\beta\} \subset \overline{E_1}$.

Let $K = E_1$. We say that K is controllable and there exists an NSC V such that $L(V/\mathbf{G}) = K$. Let $M = L_m(\mathbf{G})$. We have

$$L_m(V/\mathbf{G}) = L(V/\mathbf{G}) \cap M = \emptyset.$$

Clearly, the MNSC with respect to K is empty.

Second, let $K = E_2 = \{(\alpha\beta\alpha\beta\lambda)^*\alpha\beta\alpha\beta\alpha\}$. It is true that $K \subset L(\mathbf{G})$ and there does not exist $s \in K$ such that $s \in L_m(\mathbf{G})$. Hence there exists an NSC V such that

$$L(V/\mathbf{G}) = K$$

and there exists an empty MNSC with respect to K .

At the next step, we check the controllability of $E_3 = \{(\alpha\beta\alpha\beta\lambda)^*\}$:

- $\overline{E_3} = \overline{\{(\alpha\beta\alpha\beta\lambda)^*\}} = \{(\alpha\beta\alpha\beta\lambda)^*, (\alpha\beta\alpha\beta\lambda)^*\alpha, (\alpha\beta\alpha\beta\lambda)^*\alpha\beta, (\alpha\beta\alpha\beta\lambda)^*\alpha\beta\alpha, (\alpha\beta\alpha\beta\lambda)^*\alpha\beta\alpha\beta\}$,
- $\overline{E_3}\Sigma_{unc} = \{(\alpha\beta\alpha\beta\lambda)^*\beta, (\alpha\beta\alpha\beta\lambda)^*\alpha\beta, (\alpha\beta\alpha\beta\lambda)^*\alpha\beta\beta, (\alpha\beta\alpha\beta\lambda)^*\alpha\beta\alpha\beta, (\alpha\beta\alpha\beta\lambda)^*\alpha\beta\alpha\beta\beta\}$, and
- $\overline{E_3}\Sigma_{unc} \cap L(\mathbf{G}) = \{(\alpha\beta\alpha\beta\lambda)^*\alpha\beta, (\alpha\beta\alpha\beta\lambda)^*\alpha\beta\alpha\beta\} \subset \overline{E_3}$.

Clearly, E_3 is controllable. Let $K = E_3$. As depicted in Fig. 2.6, there exists a DES namely SUPER that implements V . It is easy to check that

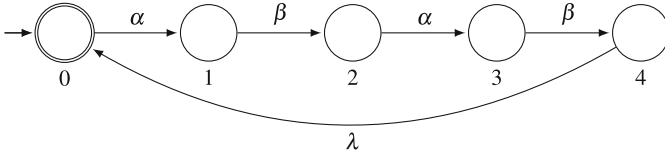


Fig. 2.6 A supervisor SUPER

$$L(V/\mathbf{G}) = K.$$

Let $M = E_3$. We conclude that $M \subseteq L_m(\mathbf{G})$ and

$$L_m(V/\mathbf{G}) = L(V/\mathbf{G}) \cap M$$

hold. We say that the supervisor illustrated in Fig. 2.6 is an MNSC. It is clear that it is also an NSC.

By repeating the same approach, we can reach a conclusion that specification $K = L(\mathbf{S})$ is an NSC but not an MNSC.

The operations in TCT to synthesize the optimal MNSC are listed below.

$$\text{SPEC} = \mathbf{edit}(\mathbf{G}, [\text{mark } +[5]], [\text{trans } -[5, 3, 1]]) (6, 6)$$

$$\text{SUPER} = \mathbf{supcon}(\mathbf{G}, \text{SPEC}) (5, 5)$$

$$\text{SUPER} = \mathbf{condat}(\mathbf{G}, \text{SUPER}) \text{ Controllable}$$

Clearly, as illustrated in Fig. 2.6, SUPER is the optimal MNSC. Following the **condat** procedure (introduced in [17]) in TCT, we find that SUPER disables event α at state 4 in Fig. 2.6. \square

2.2 Timed Discrete-Event Systems

By adjoining *time bounds* to the R-W framework (on the transitions), the *Brandin-Wonham (B-W) framework* [2] of a timed DES (TDES) is obtained. From the perspective of TDES, a DES under the R-W framework is viewed as an (untimed) *activity transition graph* (ATG) of a TDES. In other words, a TDES \mathbf{G} can be modelled starting from an untimed ATG represented by a five-tuple:

$$\mathbf{G}_{\text{act}} = (A, \Sigma_{\text{act}}, \delta_{\text{act}}, a_0, A_m)$$

that is essentially an untimed DES with its state set being replaced by an *activity set* A . The elements of A are called “activities”, usually denoted by a . Let $\mathbb{N} =$

$\{0, 1, 2, \dots\}$ and $\sigma \in \Sigma_{act}$. Every event σ in Σ_{act} is equipped with a *timer*, defined by a *lower time bound* $l_\sigma \in \mathbb{N}$ and an *upper time bound* $u_\sigma \in \mathbb{N} \cup \{\infty\}$ with respect to which the timer is “tick down”. Σ_{act} is partitioned into two subsets, which satisfies

$$\Sigma_{act} = \Sigma_{spe} \dot{\cup} \Sigma_{rem},$$

where Σ_{spe} and Σ_{rem} are the *prospective* and *remote event sets* with *finite* and *infinite* upper time bounds, respectively.

By defining a *timer interval* to tick down event σ , represented by T_σ , such that $T_\sigma = [0, u_\sigma]$ and $T_\sigma = [0, l_\sigma]$ for σ in Σ_{spe} and Σ_{rem} , respectively, a TDES state is denoted by

$$q = (a, \{t_\sigma | \sigma \in \Sigma_{act}\})$$

with $t_\sigma \in T_\sigma$, which shows that a TDES state q consists of an activity a and a tuple assigning to each event σ in Σ_{act} an integer in its timer interval T_σ . Hence, t_σ is called the *timer* of event σ in state q . Thus, the TDES state set is built as the *Cartesian product* of the activity set and the timer intervals of all the activity events appeared in Σ_{act} , i.e.,

$$Q := A \times \prod \{T_\sigma | \sigma \in \Sigma_{act}\}.$$

The initial state of a TDES is

$$q_0 := (a_0, \{t_{\sigma_0} | \sigma \in \Sigma_{act}\}),$$

where t_{σ_0} equals u_σ and l_σ for a *prospective* and a *remote* state, respectively. The *marker state set* is a user-defined subset

$$Q_m \subseteq A_m \times \prod \{T_\sigma | \sigma \in \Sigma_{act}\}.$$

The global *tick event* $tick(t)$ representing “tick of the global clock” is adjoined to Σ_{act} to form the full alphabet denoted by

$$\Sigma := \Sigma_{act} \dot{\cup} \{t\}.$$

Thus a TDES is represented by

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

Example We consider the automaton depicted in Fig. 2.1 as an ATG

$$\mathbf{G}_{act} = (A, \Sigma_{act}, \delta_{act}, a_0, A_m)$$

with

- $A = \{0, 1\}$,
- $\Sigma_{act} = \{\alpha, \beta\}$ with $\Sigma_{rem} = \{\alpha\}$ and $\Sigma_{spe} = \{\beta\}$,
- $\delta_{act}(0, \alpha) = 1$ and $\delta_{act}(1, \beta) = 0$,
- state 0 is the initial activity, i.e., $a_0 = 0$, and
- state set $\{0\}$ is the marker activity set, i.e., $A_m = \{0\}$.

We assign time bounds to α and β to be

- $l_\alpha = 1$,
- $u_\alpha = \infty$,
- $l_\beta = 1$, and
- $u_\beta = 2$,

written as $(\alpha, [1, \infty])$ and $(\beta, [1, 2])$, respectively. Then, we have a TDES

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

where

- $Q = \{0, 1\} \times \{0, 1\} \times \{0, 1, 2\}$ is the *Cartesian product* of the activity set and the non-negative integers (the timer intervals) to tick down events α and β ,
- $\Sigma = \Sigma_{act} \dot{\cup} \{t\} = \{\alpha, \beta, t\}$,
- $q_0 = (0, \{1, 2\})$ is the initial state, and
- $Q_m = \{(0, \{1, 2\})\}$ is a (user defined) marker state set.

We have size $|Q| = 12$ and we take $(0, \{1, 2\})$ and $\{(0, \{1, 2\})\}$ as the initial state and marker state set, respectively. State $q_0 = (0, \{1, 2\})$ is the initial state of the TDES, where state 0 is the initial activity, and $\{1, 2\}$ is the set of the user-defined initial (timing) labels for events α and β , respectively. The marker state set Q_m is singleton containing $q_0 = (0, \{1, 2\})$, in which state 0 is the unique marking activity, and $\{1, 2\}$ is also the set of the user-defined marking (timing) labels for events α and β , respectively. \square

In accordance with [17], an event $\sigma \in \Sigma_{act}$ is *enabled* at q if $\delta_{act}(a, \sigma)$ is defined, written as $\delta_{act}(a, \sigma)!$; it is *eligible* if its timer is also defined, i.e., $\delta(q, \sigma)!$, in accordance with the following rules:

- $\sigma = t$ and $(\forall \tau \in \Sigma_{spe}) \delta_{act}(a, \tau)! \Rightarrow t_\tau > 0$, or
- $\sigma \in \Sigma_{spe}$, $\delta_{act}(a, \sigma)!$, $0 \leq t_\sigma \leq u_\sigma - l_\sigma$, or
- $\sigma \in \Sigma_{rem}$, $\delta_{act}(a, \sigma)!$, $t_\sigma = 0$.

Formally, δ is defined as $\delta(q, \sigma) = q'$ with

$$q = (a, \{t_\tau | \tau \in \Sigma_{act}\}) \text{ and } q' = (a', \{t'_\tau | \tau \in \Sigma_{act}\}),$$

where the *entrance* q' is defined by the following rules:

1. Let $\sigma = t$.

In the entrance q' we have $a' := a$, and

- if $\tau \in \Sigma_{spe}, t'_\tau := \begin{cases} u_\tau, & \text{if not } \delta_{act}(a, \tau)! \\ t_\tau - 1, & \text{if } \delta_{act}(a, \tau)! \text{ and } t_\tau > 0 \end{cases}$, and
- if $\tau \in \Sigma_{rem}, t'_\tau := \begin{cases} l_\tau, & \text{if not } \delta_{act}(a, \tau)! \\ t_\tau - 1, & \text{if } \delta_{act}(a, \tau)! \text{ and } t_\tau > 0. \\ 0, & \text{if } \delta_{act}(a, \tau)! \text{ and } t_\tau = 0 \end{cases}$

2. Let $\sigma \in \Sigma_{act}$.

In the entrance q' we have $a' := \delta_{act}(a, \sigma)$, and

- if $\tau \neq \sigma$ and $\tau \in \Sigma_{spe}, t'_\tau := \begin{cases} u_\tau, & \text{if not } \delta_{act}(a, \tau)! \\ t_\tau, & \text{if } \delta_{act}(a, \tau)! \end{cases}$,
- if $\tau = \sigma$ and $\tau \in \Sigma_{spe}, t'_\tau := u_\sigma$,
- if $\tau \neq \sigma$ and $\tau \in \Sigma_{rem}, t'_\tau := \begin{cases} l_\tau, & \text{if not } \delta_{act}(a, \tau)! \\ t_\tau, & \text{if } \delta_{act}(a, \tau)! \end{cases}$, and
- if $\tau = \sigma$ and $\tau \in \Sigma_{rem}, t'_\tau := l_\sigma$.

In accordance with [17], only an *eligible* event can actually occur. If σ is not *enabled*, it is said to be *disabled*; if σ is not *eligible*, it is *ineligible*; an *enabled* but *ineligible* event is called a *pending* event. A TDES should satisfy *activity-loop-free*, i.e.,

$$(\forall q \in Q)(\forall s \in \Sigma_{act}^+) \delta(q, s) \neq q.$$

Example For the ATG \mathbf{G}_{act} discussed in the previous example, we have its corresponding *timed transition graph* (TTG) \mathbf{G} depicted in Fig. 2.7, which shows that:

- At state $(0, \{1, 2\})$ where items 0, 1, and 2 are respectively the activity 0 in \mathbf{G}_{act} , the lower time bound $l_\alpha = 1$ for event α , and the upper time bound $u_\beta = 2$ for event β :

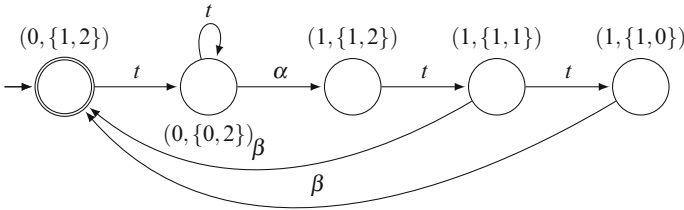


Fig. 2.7 A TTG

- event α is enabled but not eligible to occur since $t_\alpha = 0$ is violated, i.e., it is pending,
- event β is disabled since $\delta_{act}(0, \beta)!$ is not satisfied, and
- event t is eligible to occur since $\delta_{act}(0, \beta)! \Rightarrow t_\beta > 0$ holds.
- At state $(0, \{0, 2\})$:
 - event α is enabled and eligible to occur since $\delta_{act}(0, \alpha)!$ and $t_\alpha = 0$ hold,
 - event β is disabled, and
 - event t is eligible to occur since $\delta_{act}(0, \beta)! \Rightarrow t_\beta > 0$ holds.
- At state $(1, \{1, 2\})$:
 - event α is disabled,
 - event β is pending since $\delta_{act}(1, \beta)!$ and $t_\beta = 2$ violates $0 \leq t_\beta \leq u_\beta - l_\beta = 1$, and
 - event t is eligible to occur since $\delta_{act}(0, \beta)! \Rightarrow t_\beta > 0$ holds.
- At state $(1, \{1, 1\})$:
 - event α is disabled,
 - event β is enabled and eligible to occur since $\delta_{act}(1, \beta)!$ and $t_\beta = 1$ satisfies $0 \leq t_\beta \leq u_\beta - l_\beta = 1$, and
 - event t is eligible to occur since $\delta_{act}(0, \beta)! \Rightarrow t_\beta > 0$ holds.
- At state $(1, \{1, 0\})$:
 - event α is disabled,
 - event β is enabled and eligible to occur, and
 - event t is disabled since $\delta_{act}(0, \beta)! \Rightarrow t_\beta > 0$ is violated.

After the occurrence of event α at state $(0, \{0, 2\})$, the system arrives state $(1, \{1, 2\})$ which resets the timer for event α to be $l_\alpha = 1$. Event β is eligible to occur at both states $(1, \{1, 1\})$ and $(1, \{1, 0\})$. After the occurrence of event β , the system returns to the initial state with the timer for event β defined as $u_\beta = 2$.

By using the procedure **timed_graph** procedure in the TDES synthesis tool TTCT,² a TTG \mathbf{G} can be generated from the corresponding ATG \mathbf{G}_{act} , which is shown in Fig. 2.7. \square

The *closed behavior* of a TDES \mathbf{G} is represented by language

$$L(\mathbf{G}) := \{s \in \Sigma^* \mid \delta(q_0, s)!\}. \quad (2.8)$$

In addition, the *marked behavior* of \mathbf{G} is

$$L_m(\mathbf{G}) := \{s \in L(\mathbf{G}) \mid \delta(q_0, s) \in Q_m\}. \quad (2.9)$$

² <http://www.control.utoronto.ca/DES>.

A TDES \mathbf{G} is *nonblocking* if $L_m(\mathbf{G})$ satisfies

$$\overline{L_m(\mathbf{G})} = L(\mathbf{G}),$$

where $\overline{L_m(\mathbf{G})}$ denotes the (prefix) closure of $L_m(\mathbf{G})$. An ATG can be converted into a TTG by incorporating the *tick* (represented by event t) transition explicitly.

A subset $\Sigma_{hib} \subseteq \Sigma_{act}$ represents the *prohibitible event set*, in which each event can be disabled by a supervisor. Furthermore, another subset $\Sigma_{for} \subseteq \Sigma_{act}$, namely *forcible event set*, is defined, which can preempt the occurrence of event t . For example, suppose that at a state q , several *forcible* events and t are eligible. By SCT, event t can be effectively erased from the current event list. There is no particular relation postulated *a priori* between Σ_{for} and any of Σ_{hib} , Σ_{rem} , or Σ_{spe} . In particular an event in Σ_{rem} might be both *forcible* and *prohibitible*.

In a TDES plant \mathbf{G} , the eligible event set $Elig_{\mathbf{G}}(s) \subseteq \Sigma$ at a state q corresponding to a string $s \in L(\mathbf{G})$ is defined by

$$Elig_{\mathbf{G}}(s) := \{\sigma \in \Sigma \mid s\sigma \in L(\mathbf{G})\}. \quad (2.10)$$

For an arbitrary language $K \subseteq L(\mathbf{G})$, let $s \in \overline{K}$,

$$Elig_K(s) := \{\sigma \in \Sigma \mid s\sigma \in \overline{K}\}. \quad (2.11)$$

The language K is *controllable* w.r.t. \mathbf{G} if for any string $s \in \overline{K}$,

$$Elig_K(s) \supseteq \begin{cases} Elig_K(s) \cap (\Sigma_{unc} \cup \{t\}), & \text{if } Elig_K(s) \cap \Sigma_{for} = \emptyset \\ Elig_K(s) \cap \Sigma_{unc}, & \text{if } Elig_K(s) \cap \Sigma_{for} \neq \emptyset \end{cases}. \quad (2.12)$$

Thus an event $\sigma \in \Sigma$ is eligible to occur w.r.t. K if σ is *eligible* in \mathbf{G} and

- σ is *uncontrollable*, or
- $\sigma = t$ and no *forcible* event is currently *eligible* in K .

Obviously, the significance of “controllable” differs from the definition in (untimed) DES: even though event *tick* is controllable, it can be preempted only by a forcible event that is eligible to occur. The set of all controllable sublanguages of K is denoted by $\mathcal{C}(K)$ that is nonempty (the empty set belongs to it) and closed under arbitrary set unions. Hence, a unique supremal (i.e., largest) element exists, denoted by $\sup \mathcal{C}(K)$.

Suppose that a *specification language* is represented by $E \subseteq \Sigma^*$. Let

$$K \subseteq L(\mathbf{G}) \cap E$$

be nonempty and closed. There exists a supervisory control V for \mathbf{G} such that

$$L(V/\mathbf{G}) = K$$

iff K is *controllable* with respect to \mathbf{G} ; this is referred to as an NSC. Formally, a supervisory control is a map

$$V : L(\mathbf{G}) \rightarrow 2^\Sigma$$

such that, for all $s \in L(\mathbf{G})$,

$$V(s) \supseteq \begin{cases} \Sigma_{unc} \cup \{\{t\} \cap Elig_K(s)\}, & \text{if } V(s) \cap Elig_K(s) \cap \Sigma_{for} = \emptyset \\ \Sigma_{unc}, & \text{if } V(s) \cap Elig_K(s) \cap \Sigma_{for} \neq \emptyset \end{cases}. \quad (2.13)$$

The closed behavior of \mathbf{G} under the supervision of V , denoted by $L(V/\mathbf{G})$, is defined as:

- $\epsilon \in L(V/\mathbf{G})$,
- if $s \in L(V/\mathbf{G})$, $\sigma \in V(s)$, and $s\sigma \in L(\mathbf{G})$, then $s\sigma \in L(V/\mathbf{G})$, and
- no other strings belong to $L(V/\mathbf{G})$.

Generally, if *marking* is also considered, then we select a sublanguage $M \subseteq L_m(\mathbf{G})$. A marking nonblocking supervisory control (MNSC) with respect to \mathbf{G} exists, which is a map $V : L(\mathbf{G}) \rightarrow \Phi$ with the behavior

$$L_m(V/\mathbf{G}) = L(V/\mathbf{G}) \cap M. \quad (2.14)$$

Example The TTG depicted in Fig. 2.7 is nonblocking, which is considered as a plant with a forcible event α . A user-defined specification \mathbf{S} is depicted in Fig. 2.8. Let $E = L(\mathbf{S})$. We have a TTG representing $K = L(\mathbf{G}) \cap E$ shown in Fig. 2.9.

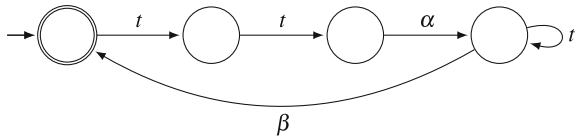
Let $s_1 = tt$, $s_2 = ttat$, and $s_3 = ttatt$. We have

- $s_1 \in \overline{K}$, and $Elig_{\mathbf{G}}(s_1) = Elig_K(s_1) = \{\alpha\}$,
- $s_2 \in \overline{K}$, and $Elig_{\mathbf{G}}(s_2) = Elig_K(s_2) = \{\beta\}$, and
- $s_3 \in \overline{K}$, and $Elig_{\mathbf{G}}(s_3) = Elig_K(s_3) = \{\beta\}$.

Strings s_1 , s_2 , and s_3 are controllable since Eq. (2.12) is satisfied. Furthermore, it is easy to check that K is controllable since all the strings s in \overline{K} satisfy Eq. (2.12). Hence K is controllable. More precisely, it is both an NSC and an MNSC.

This conclusion can also be verified by using the TTCT procedure **supcon**. Considering the TTG depicted in Figs. 2.7 and 2.8 as plants and specifications, respectively, as shown in Fig. 2.9, a TTG supervisor, denoted by SUPER, is obtained. The operations in TTCT to synthesize the optimal MNSC are listed below:

Fig. 2.8 A TDES specification



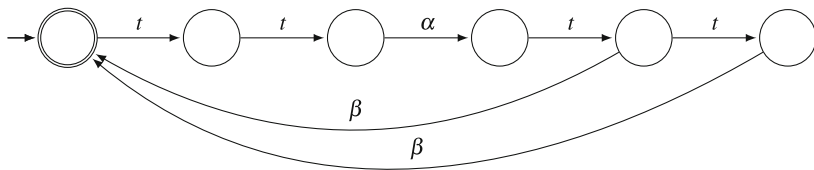


Fig. 2.9 A TDES supervisor

$G = \mathbf{acreate}$ (G1, [mark 0], [time bounds [1, 1, inf], [2, 1, 2]], [forcible 1], [tran [0, 1, 1], [1, 2, 0]]) (2, 2)

$G = \mathbf{timed_graph}$ (G) (5, 7)

$\text{SPEC} = \mathbf{create}$ (SPEC, [mark 0], [tran [0, 0, 1], [1, 0, 2], [2, 1, 3], [3, 0, 3], [3, 2, 0]], [forcible 1]) (4, 5)

$\text{SUPER} = \mathbf{supcon}$ (G, SPEC) (6, 7)

□

2.3 State-Tree Structures

Similar to the hierarchical organizations in the real world, state-tree structures (STS) are proposed in [14] for the purpose of incorporating the *hierarchical* and *concurrent structures* of complex DES (or *finite state machines*, FSM) into a compact and natural model. Thereafter, it is completed in [10] and [11]. STS are viewed as *hierarchical finite state machines* (HFMSM) [6, 7, 12]. In other words, an STS is equally considered as a set of DES with multi-levels. In this monograph, we introduce STS by starting from *superstates* as defined in *statecharts* [8]. A superstate, similar to a hierarchical organization or hierarchy, is generally made of several subordinates that may also be hierarchical organizations.

2.3.1 Superstates

A *superstate*, as known as the *abstraction* of a system or a sub-system, is an *aggregation* (or abstraction) of its components [8, 10]. Let X be a finite collection of sets that are called *states* of a system. Given a state $x \in X$ and a non-empty set

$$Y = \{x_1, x_2, \dots, x_n\} \subsetneq X$$

with $x \notin Y$, i.e., Y is a proper subset of X that does not contain x . As stated below, x is said to be a *superstate* in X expanded by Y if x can be obtained by one of the two expansions.

- **OR expansion:** x is the *disjoint union* of the states in Y , i.e.,

$$x = \dot{\bigcup}_{x_i \in Y} x_i.$$

In this case, x is called an **OR superstate** of X and x_i is called an **OR-component** of x in X . Disjointness means that the semantics of x is the *exclusive-or* of x_i , i.e., a system at state x implies that it is at exactly one state of Y .

- **AND expansion:** x is the *Cartesian product* of states in Y , i.e.,

$$x = (x_1, x_2, \dots, x_n).$$

For simplification, write

$$x = \prod_{x_i \in Y} x_i$$

or

$$x = x_1 \times x_2 \times \dots \times x_n.$$

In this case, x is called an **AND superstate** and x_i ($i \in [1, n] = \{1, 2, \dots, n\}$) is called an **AND-component** of $x \in X$. The semantics of an **AND superstate** x means that a system at state x is at all the states of Y simultaneously.

If $x \in X$ is not a superstate, it is said to be a *simple* state, denoted by **SIM**, i.e., there does not exist a non-empty set $Y = \{x_1, x_2, \dots, x_n\} \subsetneq X$ that expands x .

Formally, given a state set X , the *type function*

$$\mathcal{T} : X \rightarrow \{\text{AND, OR, SIM}\}$$

is defined by

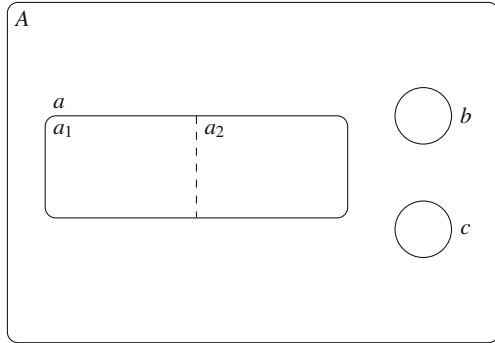
$$\mathcal{T}(x) := \begin{cases} \text{AND,} & \text{if } x \text{ is an AND superstate} \\ \text{OR,} & \text{if } x \text{ is an OR superstate} \\ \text{SIM,} & \text{otherwise} \end{cases} \quad (2.15)$$

Moreover, the *expansion function*

$$\mathcal{E} : X \rightarrow 2^X$$

is defined by

Fig. 2.10 States in statecharts



$$\mathcal{E}(x) := \begin{cases} Y, & \text{if } \mathcal{T}(x) \in \{\text{AND, OR}\} \\ \emptyset, & \text{if } \mathcal{T}(x) = \text{SIM} \end{cases} \quad (2.16)$$

with $x \in X$, $\emptyset \subseteq Y \subset X$, and $x \notin Y$, which is, for $x \in X$ with $\mathcal{T}(x) \neq \text{SIM}$, there exists a set $Y \subset X$ such that $\mathcal{E}(x) = Y$; for $x \in X$ with $\mathcal{T}(x) = \text{SIM}$, $\mathcal{E}(x) = \emptyset$.

Intuitively, a simple state has no children. An OR superstate has several children, and the system is only allowed to stay at exactly one child at a time. An AND superstate also has several children, but the system must stay at all of its children simultaneously.

Example The diagram depicted in Fig.2.10 has a state collection $X = \{A, a, b, c, a_1, a_2\}$, in which

- state A is an OR superstate expanded by states a, b , and c ;
- state a is an AND superstate expanded by states a_1 , and a_2 ;
- states b and c are two simple states without children.

As presented in Fig. 2.10, a superstate is represented by a box with round corners and a simple state is depicted by a circle. Generally, the components of a superstate are on the adjacent lower-level. Superstate A is expanded by three states a, b , and c , i.e., $\mathcal{E}(A) = \{a, b, c\}$, in which the AND superstate a is further expanded by two OR superstates a_1 and a_2 , i.e., $\mathcal{E}(a) = \{a_1, a_2\}$. The dashed line between the two boxes labelled with a_1 and a_2 represents that they are the expansions of superstate a . Based on a top-down modelling approach, the expansions of superstates are built inside the boxes iteratively. The state set X is continually growing during the modelling of an STS. We require that any state in X should only appear once.

Clearly, from the perspective of superstate A , the system must be at exactly one state of a, b , or c ; from the perspective of superstate a , the system must be at states a_1 and a_2 simultaneously. Holons defined below describe the *internal structures* of superstates a_1 and a_2 . □

After building the local transitions among the OR components, *holons* [10, 11] are created. Automatically, a set of superstates (or holons) structured in this way is nested.

2.3.2 Holons

Both the hierarchy and horizontal transition relations of an STS are described in a family of holons. A holon consists of an internal structure and a (possibly empty) *external structure*. The internal structure of a holon that matches an OR superstate x is denoted by H^x . Its internal state set X_I^x is equal to the expansion of superstate x . Formally, $\mathcal{L}(x) = X_I^x$ is true.

The external structure of a holon is defined in the adjacent higher level to build transitions with other states. Hierarchically, a holon H is defined as a five-tuple

$$H := (X, \Sigma, \delta, X_0, X_m).$$

Here,

- X is the nonempty state set, structured as the disjoint union of the (possibly empty) *external state set* X_E and the *nonempty internal state set* X_I , i.e.,

$$X = X_E \dot{\cup} X_I.$$

- Σ is the event set, structured as the disjoint union of the *boundary event set* Σ_B and the *internal event set* Σ_I , i.e.,

$$\Sigma = \Sigma_B \dot{\cup} \Sigma_I.$$

- the transition structure

$$\delta : X \times \Sigma \rightarrow X$$

is a partial function. In accordance with DES and TDES, we write $\delta(x, \sigma)!$ if $\delta(x, \sigma)$ is defined. δ is the disjoint union of two transition structures, the *internal transition structure* $\delta_I : X_I \times \Sigma_I \rightarrow X_I$ and the *boundary transition structure* δ_B which is again the disjoint union of two transition structures: $\delta_{BI} : X_E \times \Sigma_B \rightarrow X_I$ (*incoming boundary transitions*) and $\delta_{BO} : X_I \times \Sigma_B \rightarrow X_E$ (*outgoing boundary transitions*).

- $X_0 \subseteq X_I$ is the initial state set, where X_0 has exactly the target states of incoming boundary transitions if δ_{BI} is defined; otherwise X_0 is a nonempty subset of X_I selected for convenience.
- $X_m \subseteq X_I$ is the *terminal state set*, where X_m has exactly the *source states* of the outgoing boundary transitions if δ_{BO} is defined; otherwise X_m is a selected nonempty subset of X_I .

For a holon H , its event set Σ can also be partitioned into the disjoint union of *controllable events* Σ_{con} and *uncontrollable events* Σ_{unc} , i.e.,

$$\Sigma = \Sigma_{con} \dot{\cup} \Sigma_{unc}.$$

Fig. 2.11 Three superstates

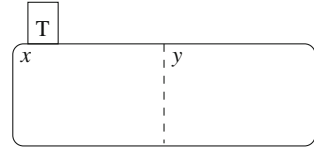


Fig. 2.12 Two DES generators. (a) DES G^x . (b) DES G^y

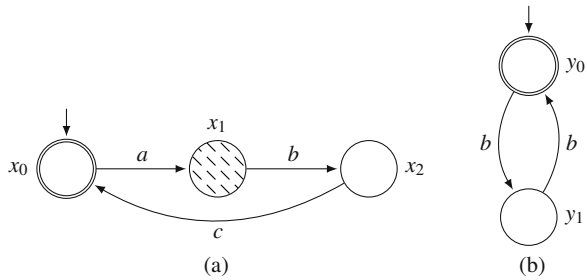
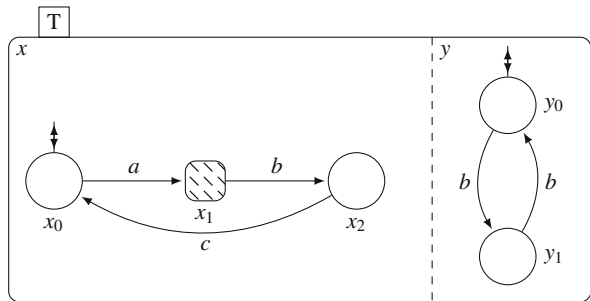
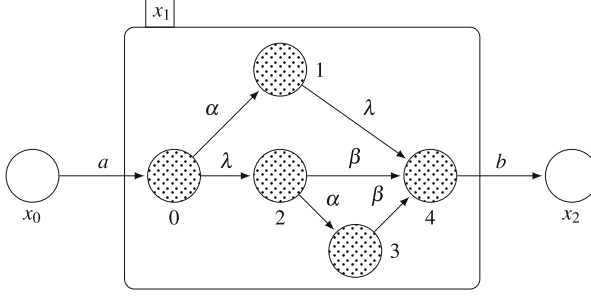
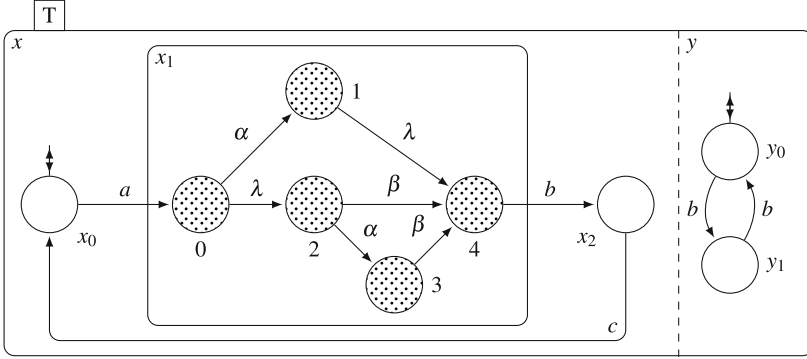


Fig. 2.13 A set of two holons



A holon with an empty external structure is identical with the DES proposed in [17]. The initial (resp., terminal) states of the holons without external states are marked by *incoming* (resp., *outgoing*) arrows. A family of holons is denoted by \mathcal{H} .

Example Given an HFSM G^T as the synchronous product of an HFSM x and an FSM y , which can be viewed as three superstates structured in Fig. 2.11. State T is an AND superstate and it is expanded by two superstates x and y . Suppose that the inner behavior of superstates x and y are respectively identical with two DES generators G^x and G^y , as depicted in Fig. 2.12. In Fig. 2.12a, superstate x_1 is dashed by north west lines, which represents that its inner structure can be constructed afterwards. As a consequence, x and y are OR superstates, and HFSM G^T is reformed as the two holons H^x and H^y illustrated in Fig. 2.13. In particular, we consider that G^x shown in Fig. 2.12a (isomorphic with holon H^x in Fig. 2.13) is hierarchical. Superstate x_1 illustrated in Fig. 2.13 is represented by a square dashed with north west lines. Note that superstate T has internal structures. In this monograph, as illustrated in Fig. 2.10, such a superstate is represented by a square with round corners.

Fig. 2.14 Holon H^{x_1} Fig. 2.15 Monolithic dynamic structure of G^T (1)

On the one hand, suppose that superstate x_1 is an OR superstate, and its internal behavior is depicted by holon H^{x_1} as shown in Fig. 2.14. Then, by plugging H^{x_1} into superstate x_1 appeared in Fig. 2.13, we obtain the monolithic dynamic structure of G^T as illustrated in Fig. 2.15. The set of the holons describing the dynamics of G^T is denoted by $\mathcal{H}^T = \{H^x, H^y, H^{x_1}\}$. Holon H^{x_1} shown in Fig. 2.14 is with internal and external structures, i.e.,

- X^{x_1} is a nonempty state set structured as the disjoint union of the external state set $X_E^{x_1} = \{x_0, x_2\}$ and internal state set $X_I^{x_1} = \{0, 1, 2, 3, 4\}$. Formally, $X^{x_1} = X_E^{x_1} \dot{\cup} X_I^{x_1} = \{x_0, x_2, 0, 1, 2, 3, 4\}$ with $X_E^{x_1} \cap X_I^{x_1} = \emptyset$;
- Σ^{x_1} is the event set, structured as the disjoint union of the boundary event set $\Sigma_B^{x_1}$ and the internal event set $\Sigma_I^{x_1}$ with $\Sigma_B^{x_1} = \{a, b\}$ and $\Sigma_I^{x_1} = \{\alpha, \beta, \lambda\}$. Formally, $\Sigma^{x_1} = \Sigma_B^{x_1} \dot{\cup} \Sigma_I^{x_1} = \{a, b, \alpha, \beta, \lambda\}$ with $\Sigma_B^{x_1} \cap \Sigma_I^{x_1} = \emptyset$;
- there exist an incoming boundary transition $\delta_{B'I}^{x_1}(x_0, a) = 0$ and an outgoing boundary transition $\delta_{B'O}^{x_1}(4, b) = x_2$;
- $X_0 = \{0\}$ is the initial state set;
- $X_m = \{4\}$ is the terminal state set.

Fig. 2.16 Holons describing internal behavior of superstate x_1

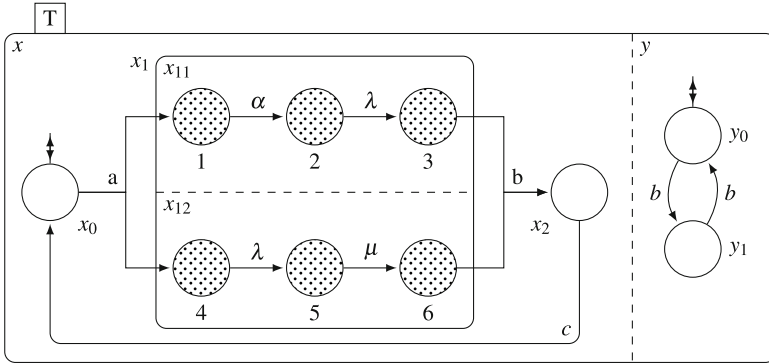
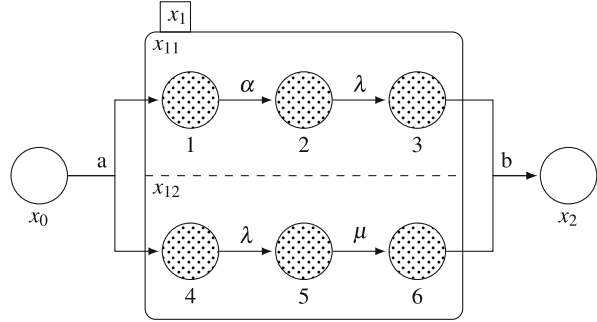


Fig. 2.17 Monolithic dynamic structure of G^T (2)

On the other hand, suppose that superstate x_1 shown in Fig. 2.13 is an AND superstate, and its internal behavior is depicted by holons $H^{x_{11}}$ and $H^{x_{12}}$ as shown in Fig. 2.16. Then, by plugging the holons into superstate x_1 in Fig. 2.13, we obtain the monolithic dynamic structure of G^T as illustrated in Fig. 2.17. The set of holons describing the dynamics of G^T is denoted by $\mathcal{H}^T = \{H^x, H^y, H^{x_{11}}, H^{x_{12}}\}$. \square

Generally, for a holon H^x , its external state set X_E^x belongs to X_I^y of holon H^y on the adjacent higher level. The occurrence of $\sigma \in \Sigma_B^x$ leads the system from H^x to H^y or vice versa. We say that superstate x satisfies $x \in X_I^y$, i.e., a lower level holon H^x is considered as an internal state of H^y . We require that $\Sigma_I^x \cap \Sigma_I^y = \emptyset$ should hold. In this monograph, for a holon with a nonempty external state set, say H^{x_1} , its internal states are graphically dashed with crosshatch dots.

Example Holon H^{x_1} illustrated in Fig. 2.14 is with $X_E^{x_1} = \{x_0, x_2\}$. The state set of holon H^x shown in Fig. 2.13 is $H^x = \{x_0, x_1, x_2\}$. Clearly, H^{x_1} is viewed as an internal state of H^x . Moreover, with event sets $\Sigma_B^{x_1} = \{a, b\}$ and $\Sigma_I^x = \{a, b, c\}$ assigned, we have that $\Sigma_E^{x_1} \subset \Sigma_I^x$ and $\Sigma_I^{x_1} \cap \Sigma_I^x = \emptyset$ hold. \square

2.3.3 State-Trees

Both the *hierarchical* and *horizontal transition relations* in an STS are described by a family of holons. The internal structure of a holon matches an **OR** superstate x and its expansion, and the external structure of a holon connects its internal behavior with the *exosystem* (outside world) [5, 16] that is on the adjacent higher level. The global state space of a set of holons is represented by a *state-tree* that is hierarchical. Note that the holons with the same *state space* (and possibly different transition relations) match the same state-tree.

Given a structured state set X , the *reflexive and transitive closure* of \mathcal{E} is written as

$$\mathcal{E}^* : X \rightarrow 2^X.$$

Consequently, given a superstate x , the *unfolding* of $\mathcal{E}(x)$ is denoted by

$$\mathcal{E}^+(x) = \mathcal{E}^*(x) - \{x\}.$$

Recursively, a state-tree is a four-tuple

$$\mathbf{ST} = (X, x_0, \mathcal{T}, \mathcal{E}),$$

where X is a finite state set with $X = \mathcal{E}^*(x_0)$ and $x_0 \in X$ is the *root state*. $\mathbf{ST} = (X, x_0, \mathcal{T}, \mathcal{E})$ is a state-tree satisfying:

1. (terminal case) $X = \{x_0\}$ represents that X contains only one simple state, or
2. (recursive case) $(\forall y \in \mathcal{E}(x_0)) \mathbf{ST}^y = (\mathcal{E}^*(y), y, \mathcal{T}_{\mathcal{E}^*(y)}, \mathcal{E}_{\mathcal{E}^*(y)})$ is also a state-tree, where

$$(\forall y, y' \in \mathcal{E}(x_0))(y \neq y' \Rightarrow \mathcal{E}^*(y) \cap \mathcal{E}^*(y') = \emptyset)$$

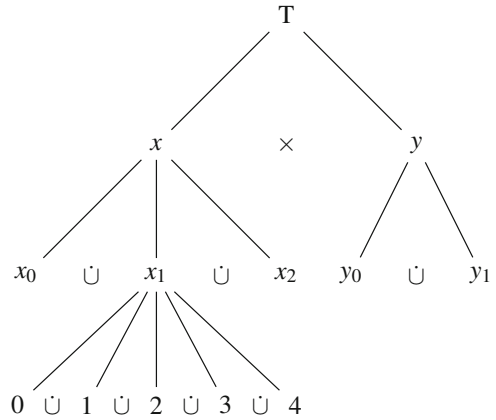
and

$$\dot{\bigcup}_{y \in \mathcal{E}(x_0)} \mathcal{E}^*(y) = \mathcal{E}^+(x_0).$$

Example The holons shown in Fig. 2.15 match the state-tree \mathbf{ST}^T depicted in Fig. 2.18. In a state-tree, the symbol \times (resp., $\dot{\cup}$) is placed between any two adjacent **AND** (resp., **OR**) components. In state-tree \mathbf{ST}^T , we have

- state collection: $X^T = \{T, x, y, x_0, x_1, x_2, y_0, y_1, 0, 1, 2, 3, 4\}$,
- type function: $\mathcal{T}(T) = \mathbf{AND}$,
- type functions: $\mathcal{T}(x) = \mathcal{T}(y) = \mathcal{T}(x_1) = \mathbf{OR}$,
- type functions: $\mathcal{T}(x_0) = \mathcal{T}(x_2) = \mathcal{T}(y_0) = \mathcal{T}(y_1) = \mathcal{T}(0) = \mathcal{T}(1) = \mathcal{T}(2) = \mathcal{T}(3) = \mathcal{T}(4) = \mathbf{SIM}$, and

Fig. 2.18 State-tree matching holons in Fig. 2.15



• expansion functions:

- $\mathcal{E}(T) = \{x, y\}$,
- $\mathcal{E}(x) = \{x_0, x_1, x_2\}$,
- $\mathcal{E}(y) = \{y_0, y_1\}$,
- $\mathcal{E}(x_1) = \{0, 1, 2, 3, 4\}$,
- $\mathcal{E}(x_0) = \emptyset$,
- $\mathcal{E}(x_2) = \emptyset$,
- $\mathcal{E}(y_0) = \emptyset$,
- $\mathcal{E}(y_1) = \emptyset$,
- $\mathcal{E}(0) = \emptyset$,
- $\mathcal{E}(1) = \emptyset$,
- $\mathcal{E}(2) = \emptyset$,
- $\mathcal{E}(3) = \emptyset$, and
- $\mathcal{E}(4) = \emptyset$.

For the state-tree ST^T depicted in Fig. 2.18, we have

- $\mathcal{E}^*(T) = \{T, x, y, x_0, x_1, x_2, y_0, y_1, 0, 1, 2, 3, 4\}$, and
- $\mathcal{E}^+(T) = \{x, y, x_0, x_1, x_2, y_0, y_1, 0, 1, 2, 3, 4\}$. □

Say that ST^y is a *child-state-tree* of x_0 in ST , rooted by y . For convenience, if $y \in \mathcal{E}^+(x)$, we call y a *descendant* of x and x an *ancestor* of y , which is denoted by $x < y$. States x and y are *incomparable* if x is neither the *ancestor* nor the *descendant* of y . An **OR** superstate y is **AND-adjacent** to an **AND** superstate x , denoted by $x <_{\times} y$, if

$$x < y \ \& \ \mathcal{T}(x) = \text{AND} \ \& \ [(\forall z)x < z < y \Rightarrow \mathcal{T}(z) = \text{AND}]. \tag{2.17}$$

State z is the *nearest common ancestor* (NCA) of x and y if

$$z < x \ \& \ z < y \ \& \ [\neg(\exists a \in \mathcal{E}^+(z))a < x \ \& \ a < y]. \tag{2.18}$$

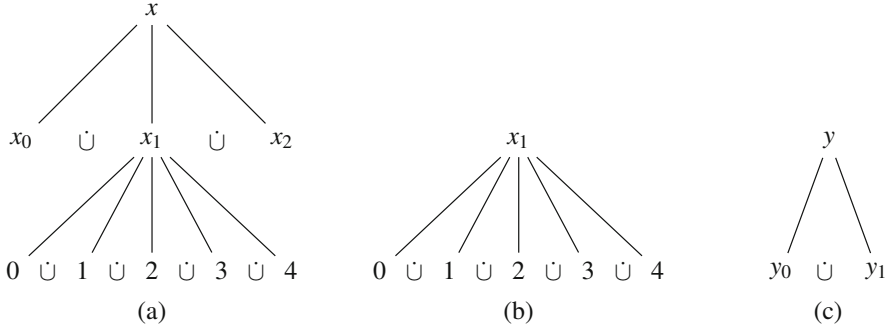


Fig. 2.19 Child-state-trees. (a) ST^x . (b) ST^{x_1} . (c) ST^{y_1}

Example For the state-tree ST^T depicted in Fig. 2.18, we have $T <_x x_0$, $T <_x y_1$, and the NCA of states 0 and y_1 is state T . Moreover, as depicted in Fig. 2.19, for the state-tree ST^T , we can obtain three child-state-trees ST^x , ST^{x_1} , and ST^y rooted by states x , x_1 , and y , respectively. \square

A *sub-state-tree* is denoted by

$$subST = (Y, x_0, \mathcal{F}', \mathcal{E}')$$

with $\mathcal{E}' : Y \rightarrow 2^Y$ defined for $y \in Y$ as

$$\begin{cases} \mathcal{E}'(y) = \mathcal{E}(y), & \text{if } \mathcal{F}'(y) \neq \text{OR} \\ \emptyset \subset \mathcal{E}'(y) \subseteq \mathcal{E}(y), & \text{if } \mathcal{F}'(y) = \text{OR} \end{cases} \quad (2.19)$$

A well-formed state-tree is a *basic-state-tree* if any of its OR superstates has exactly one expansion (or child).

Example The state-tree illustrated in Fig. 2.20 is a sub-state-tree of ST^T depicted in Fig. 2.18 and it is also a basic-state-tree. \square

A state-tree is *well-formed* if

- for any two states x and y , one of the following statements is satisfied:
 - $x \leq y$ or $y \leq x$, or
 - $x|y$, namely the NCA of incomparable states x and y is an AND superstate, or
 - $x \oplus y$, namely the NCA of incomparable states x and y is an OR superstate;
- $(\forall x, y \in X) \mathcal{F}(x) = \text{AND} \ \& \ y \in \mathcal{E}(x) \Rightarrow \mathcal{F}(x) \neq \text{SIM}$, i.e., AND components cannot be simple states; and
- all the *leaf states* are simple states.

Fig. 2.20 A basic-state-tree of the state-tree in Fig. 2.18

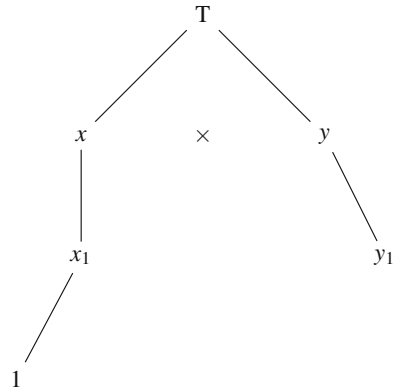


Fig. 2.21 Superstate expansions

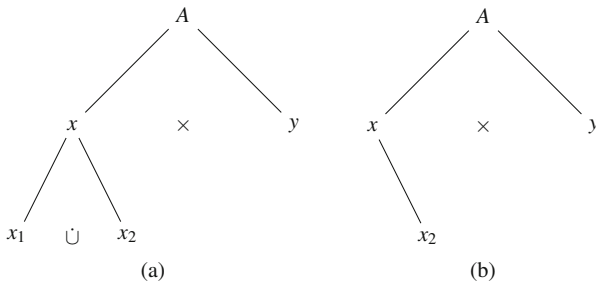
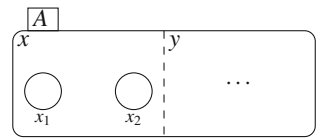


Fig. 2.22 Matching state-tree and a sub-state-tree. (a) The matching tree. (b) A sub-state-tree

Example The state-tree ST^T depicted in Fig. 2.18 is a well-formed state-tree. Moreover, suppose that an AND superstate A is expanded by two OR superstates x and y , i.e., $\mathcal{E}(A) = \{x, y\}$, and the superstate x is further expanded by two simple states x_1 and x_2 , i.e., $\mathcal{E}(x) = \{x_1, x_2\}$. The global expansion relation structured in Fig. 2.21 can be represented by the state-tree ST^A illustrated in Fig. 2.22a. Figure 2.22b depicts a sub-state-tree of ST^A . Neither of them is well-formed if the child-state-tree rooted by superstate y is not well-formed. \square

In accordance with [15], the *state aggregation* bonded with a superstate x is denoted by $X_{\mathcal{A}}(x)$. Formally,

$$X_{\mathcal{A}}(x) := \begin{cases} \mathcal{E}(x), & \text{if } \mathcal{T}(x) = \text{OR} \\ \bigcup_{x <_x y} \mathcal{E}(y), & \text{if } \mathcal{T}(x) = \text{AND} \end{cases} \quad (2.20)$$

Example For the state-tree ST^T depicted in Fig. 2.18, we have four state aggregations listed below:

- $X_{\mathcal{A}}(\mathbf{T}) = \{x_0, x_1, x_2, y_0, y_1\}$,
- $X_{\mathcal{A}}(x) = \{x_0, x_1, x_2\}$,
- $X_{\mathcal{A}}(y) = \{y_0, y_1\}$, and
- $X_{\mathcal{A}}(x_1) = \{0, 1, 2, 3, 4\}$. □

Given a proper sub-state-tree $T = (Y, x_0, \mathcal{F}', \mathcal{E}')$, with the full state-tree in mind, it can be equivalently represented by its *leaf state set*

$$V(T) = \{x \in Y \mid \mathcal{E}'(x) = \emptyset\}. \quad (2.21)$$

For simplification, the corresponding *key leaf state set* is defined below.

$$\mathcal{V}(T) := \begin{cases} V(T), & \text{if } (\nexists x) X_{\mathcal{A}}(x) \subseteq V(T) \\ V(T) - \bigcup_{(\forall x \in X) X_{\mathcal{A}}(x) \subseteq V(T)} X_{\mathcal{A}}(x), & \text{otherwise} \end{cases} \quad (2.22)$$

It shows that the key leaf states in $\mathcal{V}(T)$ only record the proper subsets of OR expansions. Given a state-tree and $\mathcal{V}(T)$, T can be restored.

Example The key leaf state set of the basic-state-tree shown in Fig. 2.20 is denoted by $\mathcal{V}(T) = \{y_1, 1\}$. No matter what is the expansion of the OR superstate y in the sub-state-tree depicted in Fig. 2.22b, $\mathcal{V}(ST^A) = \{x_2\}$ is always true. □

2.3.4 State-Tree Structures

With holons and state-trees defined, now we are ready to present the definition of STS formally. An STS is a six-tuple

$$\mathbf{G} = (ST, \mathcal{H}, \Sigma, \Delta, ST_0, ST_m),$$

where

- ST is a *state-tree*,
- \mathcal{H} is the set of *holons*,
- Σ is the union of events appearing in \mathcal{H} ,
- Δ is the *global forward transition function* $\mathcal{S}T(ST) \times \Sigma \rightarrow \mathcal{S}T(ST)$, where $\mathcal{S}T(ST)$ is the set of all *sub-state-trees*,
- ST_0 is the *initial state-tree*, and
- ST_m is the *marker state-tree set*.

Example The holons shown in Fig. 2.15 and the matching state-tree ST^T depicted in Fig. 2.18 together form an STS with $\mathcal{V}(ST_0^T) = \mathcal{V}(ST_m^T) = \{x_0, y_0\}$. \square

An STS \mathbf{G} is well-formed if it satisfies:

- ST is a well-formed state-tree;
- the states in any holon H^x are *boundary consistency*, i.e., state $y \in X_I^x$ satisfies $y \in \mathcal{E}(x)$ and $y \in X_E^x$ satisfies $(\exists z, \exists w \in X)z <_x w \ \& \ x, y \in \mathcal{E}(w)$; and
- the states in any holon are *local coupling*, i.e., for holons $H^x, H^y \in \mathcal{H}$, they satisfy

$$\Sigma_I^x \cap \Sigma_I^y \neq \emptyset \Rightarrow (\exists z)z <_x x \ \& \ z <_x y.$$

The boundary consistency requires that the boundary transitions in a holon should not skip holon levels. The local coupling requires that only the holons that have an AND superstate as the NCA of their matching superstates should share events. Hence, this NCA superstate is viewed as the synchronous product of these holons. Unless otherwise stated, in this monograph, the STS under analysis are well-formed.

The synchronous product principle (the shared event σ occurring in local coupling holons simultaneously) [17] is integrated in the *largest eligible state-tree* and *largest next state-tree*, denoted by

$$Elig_{\mathbf{G}} : \Sigma \rightarrow \mathcal{ST}(ST)$$

and

$$Next_{\mathbf{G}} : \Sigma \rightarrow \mathcal{ST}(ST),$$

respectively. The key leaf states of $Elig_{\mathbf{G}}(\sigma)$ and $Next_{\mathbf{G}}(\sigma)$ are the *exits* and *entrances* of event σ in all the holons where it appears, respectively. As stated before, the *forward transitions* are defined as

$$\Delta : \mathcal{ST}(ST) \times \Sigma \rightarrow \mathcal{ST}(ST).$$

Given any sub-state-tree $T \in \mathcal{ST}(ST)$, $T' = \Delta(T, \sigma)$ is obtained via replacing the source states of σ in $T \wedge Elig_{\mathbf{G}}$ by the corresponding target states simultaneously. The *backward transitions* are defined as

$$\Gamma : \mathcal{ST}(ST) \times \Sigma \rightarrow \mathcal{ST}(ST)$$

in a dual route.

Example For all the events σ appearing in the holons shown in Fig. 2.15, $Elig_{\mathbf{G}}(\sigma)$ and $Next_{\mathbf{G}}(\sigma)$ are depicted in Figs. 2.23 and 2.24, respectively. Moreover, all the corresponding key leaf state sets are listed in Table 2.1.

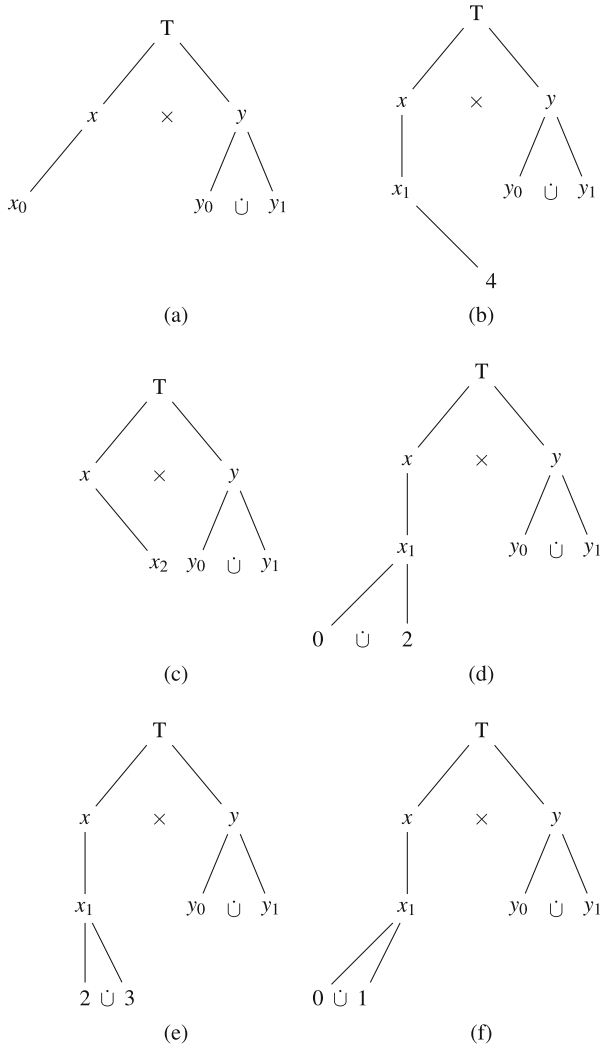


Fig. 2.23 $Elig_G(\sigma)$ for $\sigma \in \Sigma$. (a) $Elig_G(a)$. (b) $Elig_G(b)$. (c) $Elig_G(c)$. (d) $Elig_G(\alpha)$. (e) $Elig_G(\beta)$. (f) $Elig_G(\lambda)$

We have $ST_0 \in \mathcal{S}T(ST)$ and $a \in \Sigma$. Moreover, we obtain

$$ST_0 \wedge Elig_G(a) \neq \emptyset$$

and

$$\Delta(ST_0, a) = ST_1$$

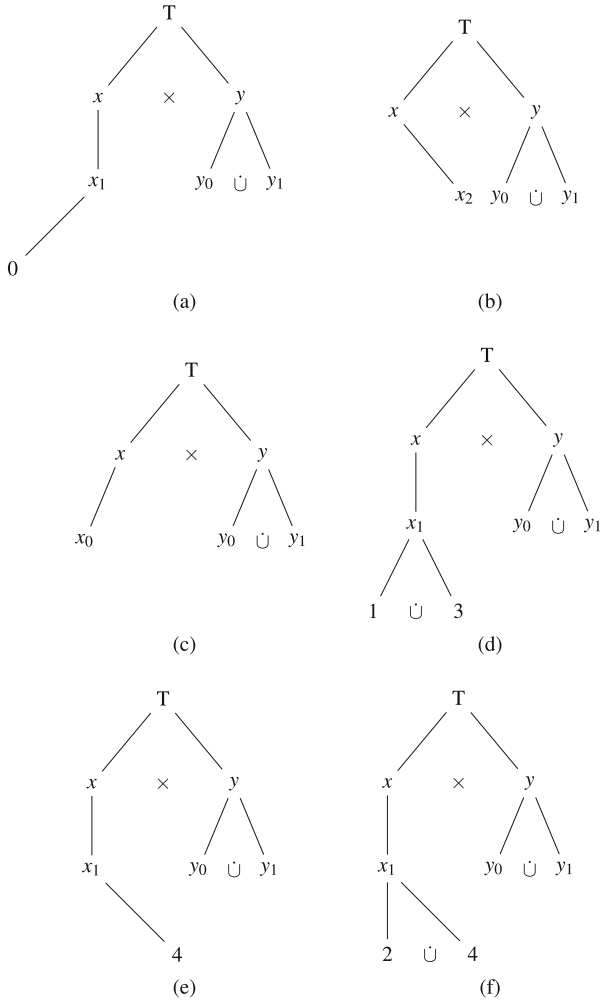


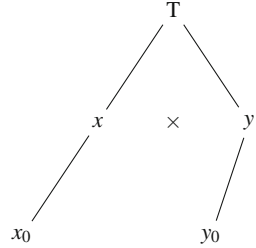
Fig. 2.24 $Next_G(\sigma)$ for $\sigma \in \Sigma$. (a) $Next_G(a)$. (b) $Next_G(b)$. (c) $Next_G(c)$. (d) $Next_G(\alpha)$. (e) $Next_G(\beta)$. (f) $Next_G(\lambda)$

Table 2.1 $Elig_G(\sigma)$ and $Next_G(\sigma)$

Event σ	$Elig_G(\sigma)$	$Next_G(\sigma)$
a	$\{x_0\}$	$\{0\}$
b	$\{4\}$	$\{x_2\}$
c	$\{x_2\}$	$\{x_0\}$
α	$\{0, 2\}$	$\{1, 3\}$
β	$\{2, 3\}$	$\{4\}$
λ	$\{0, 1\}$	$\{2, 4\}$

Table 2.2 Enabled events at each sub-state-tree in \mathbf{ST}

Sub-state-tree	Key leaf states	Enabled event set
\mathbf{ST}_0	$\{x_0\}$	$\{a\}$
\mathbf{ST}_1	$\{0\}$	$\{\alpha, \lambda\}$
\mathbf{ST}_2	$\{1\}$	$\{\lambda\}$
\mathbf{ST}_3	$\{2\}$	$\{\alpha, \beta\}$
\mathbf{ST}_4	$\{3\}$	$\{\beta\}$
\mathbf{ST}_5	$\{4\}$	$\{b\}$
\mathbf{ST}_6	$\{x_2\}$	$\{c\}$

Fig. 2.25 Initial state-tree

that is $Next_{\mathbf{G}}(a)$ shown in Fig. 2.24a. For all the other events $\sigma \in \Sigma - \{a\}$, we have

$$T \wedge Elig_{\mathbf{G}}(\sigma) = \emptyset$$

and

$$\Delta(\mathbf{ST}_0, \sigma) = \emptyset.$$

We say that at state-tree \mathbf{ST}_0 , event a is enabled. By repeating this process iteratively, we obtain all the individual sub-state-trees in \mathbf{ST} and the corresponding enabled event sets, which are listed in Table 2.2. The computation of the total function Γ is started from \mathbf{ST}_m in an opposite way. The details are omitted. \square

Given an HFSM, there always exists an equivalent single level DES representing its global behavior [1, 10, 11]. Similarly, given an STS, the set of its *basic-state-trees* is denoted by $\mathcal{B}(\mathbf{ST})$, in which an element T corresponds to a state in a single level DES representing its global behavior. The presented transition relations Δ or Γ maps an element $T \in \mathcal{B}(\mathbf{ST})$ to another. In this monograph, these basic-state-trees are symbolically encoded into predicates that are represented by binary decision diagrams (BDD).

Example For the STS shown in Figs. 2.15 and 2.18, its initial state-tree being a basic-state-tree is depicted in Fig. 2.25. Suppose that there exists an equivalent single level DES with its initial state representing this initial state-tree. Clearly, such a DES can be built by tracking the transition relations in the STS. \square

2.3.5 Predicates

Given an STS \mathbf{G} , the components of $\mathcal{B}(\mathbf{ST})$ are symbolically encoded into *predicates* that are represented by BDD. Intuitively, a predicate P (or a *characteristic function*) is defined over $\mathcal{B}(\mathbf{ST})$, i.e.,

$$P : \mathcal{B}(\mathbf{ST}) \rightarrow \{0, 1\}.$$

The *truth-value* 1 (resp., 0) represents logical *true* (resp., *false*). The predicate containing all the basic-state-trees is denoted by a predicate

$$P_{\mathbf{ST}} := \{b \in \mathcal{B}(\mathbf{ST}) \mid P(b) = 1\}.$$

Formally,

$$P(b) = 1$$

is represented by

$$b \models P.$$

Propositional logic operators are defined by:

- $(\neg P)(b) = 1$ iff $P(b) = 0$,
- $(P_1 \wedge P_2)(b) = 1$ iff $P_1(b) = 1$ and $P_2(b) = 1$, and
- $(P_1 \vee P_2)(b) = 1$ iff $P_1(b) = 1$ or $P_2(b) = 1$.

Example The initial state-tree \mathbf{ST}_0 and the marker state-tree set \mathbf{ST}_m are represented by two predicates

$$P_0 := \{b \in \mathcal{B}(\mathbf{ST}_0) \mid P(b) = 1\}$$

and

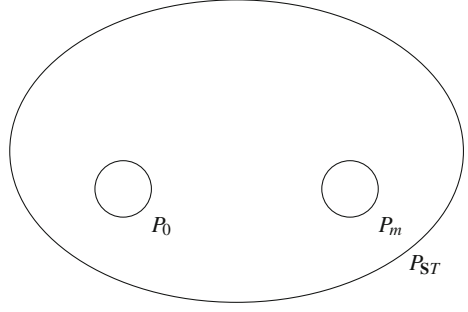
$$P_m := \{b \in \mathcal{B}(\mathbf{ST}_m) \mid P(b) = 1\},$$

respectively. The predicate containing all the basic-state-trees is denoted by a predicate

$$P_{\mathbf{ST}} := \{b \mid b \in \mathcal{B}(\mathbf{ST}) \mid P(b) = 1\}. \quad \square$$

The set of all predicates on $\mathcal{B}(\mathbf{ST})$ is defined by $\text{Pred}(\mathbf{ST})$. The partial order for subset containment is defined by $P_1 \preceq P_2$ iff $P_1 \wedge P_2 = P_1$. It is clear that P_1 is *stronger* than P_2 and $(\text{Pred}(\mathbf{ST}), \preceq)$ is a *complete lattice*. The top and bottom elements of a predicate are denoted as *true* (\top) and *false* (\perp), respectively.

Fig. 2.26 Predicate containment



Example By definition, we have $P_0 \leq P_{ST}$ and $P_m \leq P_{ST}$. As shown in Fig. 2.26, P_{ST} is the *weakest predicate* which is identified by all the basic-state-trees in $\mathcal{B}(ST)$. \square

Let $P \in \text{Pred}(ST)$. According to [10] and [11], the *reachability predicate* $R(\mathbf{G}, P)$ is true for a basic-state-tree if it can be reached in \mathbf{G} , from some $b_0 \models P \wedge P_0$, via a sequence of basic-state-trees satisfying P . Formally,

- $P \wedge P_0 = \perp \Rightarrow R(\mathbf{G}, P) = \perp$,
- $(b_0 \models P \wedge P_0) \Rightarrow (b_0 \models R(\mathbf{G}, P))$,
- $b \models R(\mathbf{G}, P) \ \& \ \sigma \in \Sigma \ \& \ \Delta(b, \sigma) \neq \emptyset \ \& \ \Delta(b, \sigma) \models P \Rightarrow \Delta(b, \sigma) \models R(\mathbf{G}, P)$, and
- no other basic-state-trees satisfy $R(\mathbf{G}, P)$.

Dually, the *coreachability predicate* $CR(\mathbf{G}, P)$ is true for a basic-state-tree if it can reach some $b_m \models P \wedge P_m$ in \mathbf{G} by a sequence of basic-state-trees satisfying P . Formally,

- $P \wedge P_m = \perp \Rightarrow CR(\mathbf{G}, P) = \perp$,
- $(b_m \models P \wedge P_m) \Rightarrow (b_m \models CR(\mathbf{G}, P))$,
- $b \models CR(\mathbf{G}, P) \ \& \ \sigma \in \Sigma \ \& \ \Gamma(b, \sigma) \neq \emptyset \ \& \ \Gamma(b, \sigma) \models P \Rightarrow \Gamma(b, \sigma) \models CR(\mathbf{G}, P)$, and
- no other basic-state-trees satisfy $CR(\mathbf{G}, P)$.

Given a predicate P , a predicate transformer $[P]$ in \mathbf{G} is defined by

1. $b \models P \Rightarrow b \models [P]$,
2. $b \models P \ \& \ \sigma \in \Sigma_u \Rightarrow \Gamma(b, \sigma) \models [P]$, and
3. no other basic-state-trees satisfy $[P]$.

2.3.6 State Feedback Control

Nonblocking supervisory control of STS utilizes predicates to record the system's behavior. The *weakest liberal precondition* $M_\sigma(P)$ is defined in [10] and [11] as

$$b \models M_\sigma(P)$$

iff

$$\Delta(b, \sigma) \models P.$$

Let \mathbf{G} be an STS, $T \in \mathcal{B}(\mathbf{ST})$, and $\sigma \in \Sigma$. In STS [10, 11], according to *state feedback control* (SFBC), the act of preventing the occurrence of an uncontrollable event σ at T is denoted by (T, σ) , where sub-state-tree T is considered as an illegal sub-state-tree. By integrating all such sub-state-trees with user predefined other illegal sub-state-trees, an illegal predicate is obtained.

Given an illegal predicate P , a *predicate transformer* $[\cdot]$ is utilized to find all the basic-state-trees that can reach P through *uncontrollable paths*. As a consequence, the family of *weakly controllable subpredicates* of P is denoted by

$$\sup^{\mathcal{C}} \mathcal{P}(\neg P) = \neg[P]. \quad (2.23)$$

Given an illegal predicate P , by SFBC, the supremal element of *weakly controllable and coreachable behavior*, i.e., *optimal behavior* of \mathbf{G} , is denoted by a nonblocking subpredicate $\sup^{\mathcal{C}^2} \mathcal{P}(\neg P)$. It is synthesized iteratively by the following steps:

1. Let $K_0 := \neg P$,
2. compute $K_{i+1} := \neg P \wedge CR(\mathbf{G}, \neg[K_i])$, and
3. If $K_{i+1} = K_i$, then $\sup^{\mathcal{C}^2} \mathcal{P}(\neg P) = K_i$; otherwise, go back to step 2.

The corresponding calculation is detailed in [10] and [11], based on which the *control function* f_σ for each controllable event $\sigma \in \Sigma_c$ is obtained. Function f_σ is represented by a predicate, which contains all the basic-state-trees where event σ is allowed to occur. Let

$$f : \mathcal{B}(\mathbf{ST}) \rightarrow \Pi$$

denote the SFBC for \mathbf{G} , where

$$\Pi := \{\Sigma' \subseteq \Sigma \mid \Sigma_u \subseteq \Sigma'\}. \quad (2.24)$$

Hence, the closed-loop transition function is represented by

$$\Delta^f(b, \sigma) = \Delta(b, \sigma)$$

iff

$$f_\sigma(b) = 1.$$

Let

$$P \in \text{Pred}(\mathbf{ST})$$

and $P \wedge P_0 \neq \perp$. The STS under control is

$$\mathbf{G}^f = (ST, \mathcal{H}, \Sigma, \Delta^f, P_0^f, P_m^f)$$

with

$$P_0^f = P \wedge P_0$$

and

$$P_m^f = P \wedge P_m.$$

As shown in Fig. 2.27, given a specification predicate P , the optimal behavior of STS \mathbf{G} is represented by $\sup^{\mathcal{L}^2} \mathcal{P}(P)$ that is viewed as an *agent* $\mathbf{G}_{tracker}$. For the *current status* (a basic-state-tree b) of \mathbf{G} , a set of *decision makers* f_{σ_i} is provided by $\mathbf{G}_{tracker}$ with $\sigma_i \in \Sigma_{con}$ and $i = 1, 2, \dots, n$, which makes the decisions by applying b as the argument. If

$$f_{\sigma_i}(b) = 1,$$

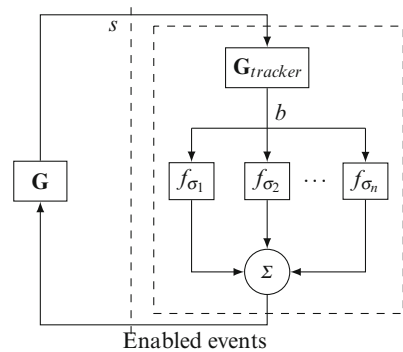
then σ_i is allowed to occur; otherwise, it is disabled.

Example As stated in [10] and [11], let us take the transfer line shown in Fig. 2.28 as an example. Suppose that the capacities of the buffers B1 and B2 are both one. A test unit is represented by TU. As depicted in Fig. 2.29, the system behavior of machines M1 and M2 are described by two holons. The corresponding state-tree is shown in Fig. 2.30. The events denoted by odd and even numbers are controllable and uncontrollable events, respectively. For nonblocking supervisory control, the controllers with positive BDD node sizes are shown in Fig. 2.31 in which only the BDD true parts are depicted to clearly show the control logic.

The control patterns for the controllable events are:

- event 1 is enabled at: $\{B1_0, B2_0, M1_0, TU_0\}$,

Fig. 2.27 STS control diagram



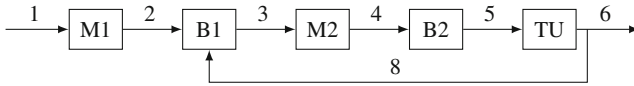


Fig. 2.28 Transfer line

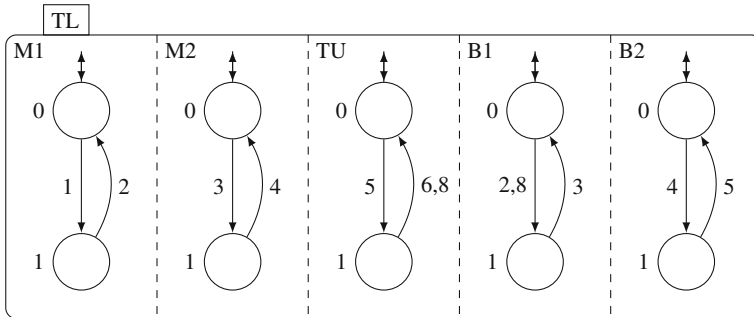


Fig. 2.29 Holons of transfer line

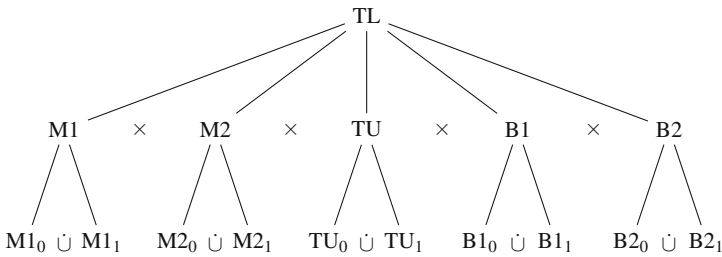


Fig. 2.30 State-tree of transfer line

- event 3 is enabled at: $\{B1_1\}$, and
- event 5 is enabled at: $\{B2_1\}$.

The control patterns show that:

- event 1 is allowed to occur only when machine M2 is idle,
- event 3 is allowed to occur only when buffer B1 is occupied, and
- event 5 is allowed to occur only when buffer B2 is occupied. □

2.3.7 Compact Representation of Predicates

In the STS framework, the predicates of an STS are encoded into BDD. Given an STS, its *BDD variables* are ordered in a top-down approach according to the subordination relation among STS nests. According to STS [10], we require that:

- the encoding for each transition labelled event σ should be linear in the number of transitions, and

Fig. 2.31 Control functions for transfer line. (a) f_1 . (b) f_3 . (c) f_5

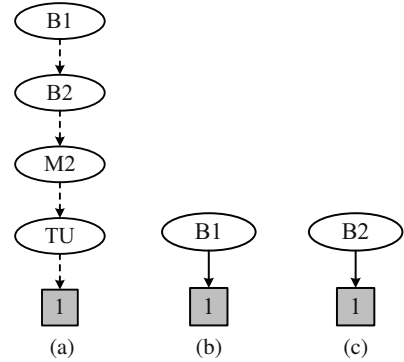


Table 2.3 BDD vectors encoding states

State	BDD vector
$M1_0$	$\langle M1 : 0 \rangle$
$M1_1$	$\langle M1 : 1 \rangle$
$M2_0$	$\langle M2 : 0 \rangle$
$M2_1$	$\langle M2 : 1 \rangle$
TU_0	$\langle TU : 0 \rangle$
TU_1	$\langle TU : 1 \rangle$
$B1_0$	$\langle B1 : 0 \rangle$
$B1_1$	$\langle B1 : 1 \rangle$
$B2_0$	$\langle B2 : 0 \rangle$
$B2_1$	$\langle B2 : 1 \rangle$

- in the case that holon H^y is subordinate to holon H^x , the BDD variables of H^x should precede those of holon H^y .

The *computational complexity* of supervisor synthesis is polynomial in the number of BDD nodes in use. Usually, it is much smaller than the states of an STS.

As proposed in [3] and [4], the states in the state set X^x of a holon H^x are encoded by BDD nodes (variables). Consider a state set X^x with a state space $|X^x| = N$. Each element y in X^x is encoded as a *vector* of n binary values, where $n = \lceil \log_2 N \rceil$. The encoding process is denoted by a function $f : X^x \rightarrow \{0, 1\}^n$ that maps each element y in X^x to a distinct n -bit binary vector. According to [10], the n variables are denoted by x_i with $0 \leq i < n$.

Example As shown in Fig. 2.29, there are two states in holon H^{M1} , i.e., $X^{M1} = \{M1_0, M1_1\}$. As a consequence, one BDD node $M1$ is required. For example, let $M1 : 0$ and $M1 : 1$ denote that $M1$ is encoded as 0 and 1, respectively. The encoding pairs for the states in the transfer line example are shown in Table 2.3.

The control functions of events 1, 3, and 5 are denoted by f_1 , f_3 , and f_5 , respectively. The *truth table* for these control functions is obtained, as shown in Table 2.4, where “*” denotes a variable that can be assigned 0 or 1. \square

Table 2.4 Truth table of control functions

Control functions	$M1$	$M2$	TU	$B1$	$B2$
f_1	*	1	1	1	1
f_3	*	*	*	1	*
f_5	*	*	*	*	1

2.4 Real-Time Scheduling/Reconfiguration Based on Supervisory Control

In Chap. 4, supervisory control of TDES is implemented in the non-preemptive scheduling of RTS processing two types of tasks: *resource-sharing tasks* or *independent tasks*. The TDES framework is utilized to model each RTS task as a monolithic TTG. For the purpose of reconfiguring independent tasks, a multi-period periodic task model is proposed in Chap. 4, which provides a set of possible periods varying between a lower bound and an upper bound. The default period for an independent task is the shortest one. In the case that an RTS is non-schedulable, based on NSC, the multi-period is used to reconfigure the RTS automatically. A uni-processor RTS' execution model is the synchronous product of all the tasks running in it. For both scheduling and reconfiguration, supervisory control of TDES is utilized to find all the safe execution sequences.

Thereafter, RTS processing independent tasks are modelled by DES frameworks monolithically (Chap. 5) and modularly (Chap. 6). The latter also provides the DES version multi-periods. Priorities in real-time scheduling are generalized as *priority-free conditional-preemption* (PFCP, presented in Chap. 5) relations. Based on SCT, for real-time scheduling and reconfiguration, all the possible safe execution sequences are found.

Finally, in Chap. 7, an RTS processing independent tasks is modelled in an STS hierarchically. By assigning specifications for the STS model, the PFCP and a classical dynamic scheduling earliest-deadline first (EDF) scheme, proposed in [9], are addressed. Finally, the RTS can be scheduled or reconfigured according to the computed controllers for the controllable events. In particular, with the dynamic specifications assigned, a few sequences are selected, which rank at the top according to some specified (dynamic) optimality criteria.

References

1. Alur, R., Kannan, S., Yannakakis, M.: Communicating hierarchical state machines. In: International Colloquium on Automata, Languages, and Programming, pp. 169–178. Springer, Berlin (1999)
2. Brandin, B.A., Wonham, W.M.: Supervisory control of timed discrete-event systems. *IEEE Trans. Autom. Control* **39**(2), 329–342 (1994)
3. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.* **100**(8), 677–691 (1986)

4. Chao, W., Gan, Y., Wang, Z., Wonham, W.M.: Modular supervisory control and coordination of state tree structures. *Int. J. Control* **86**(1), 9–21 (2013)
5. Francis, B.A., Wonham, W.M.: The internal model principle of control theory. *Automatica* **12**(5), 457–465 (1976)
6. Gaudin, B., Marchand, H.: Supervisory control of product and hierarchical discrete event systems. *Eur. J. Control* **10**(2), 131–145 (2004)
7. Gaudin, B., Marchand, H.: Safety control of hierarchical synchronous discrete event systems: a state-based approach. In: *IEEE International Symposium on, Mediterrean Conference on Control and Automation Intelligent Control*, pp. 889–895. IEEE (2005)
8. Harel, D.: Statecharts: a visual formalism for complex systems. *Sci. Comput. Program.* **8**(3), 231–274 (1987)
9. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM* **20**(1), 46–61 (1973)
10. Ma, C., Wonham, W.M.: *Nonblocking Supervisory Control of State Tree Structures*, vol. 317. Springer, Berlin (2005)
11. Ma, C., Wonham, W.M.: Nonblocking supervisory control of state tree structures. *IEEE Trans. Autom. Control* **51**(5), 782–793 (2006)
12. Marchand, H., Gaudin, B.: Supervisory control problems of hierarchical finite state machines. In: *IEEE Conference on Decision and Control*, vol. 2, pp. 1199–1204. IEEE (2002)
13. Ramadge, P.J., Wonham, W.M.: Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.* **25**(1), 206–230 (1987)
14. Wang, B.: Top-down design for RW supervisory control theory. Master’s Thesis, Department of Electrical and Computer Engineering, University of Toronto (1996)
15. Wang, X., Moor, T., Li, Z.: Top-down nested supervisory control of state-tree structures based on state aggregations. In: *IFAC-PapersOnLine*, vol. 53, pp. 11175–11180. Elsevier, Amsterdam (2020)
16. Wonham, W.M.: Towards an abstract internal model principle. *IEEE Trans. Syst. Man Cybern.* **SMC-6**(11), 735–740 (1976)
17. Wonham, W.M., Cai, K.: *Supervisory Control of Discrete-Event Systems*. Monograph Series Communications and Control Engineering. Springer, Berlin (2018)

Chapter 3

Real-Time Scheduling and Reconfiguration



3.1 Real-Time Systems

In this monograph, a *real-time system* (RTS) is denoted by \mathbb{S} . We assume that a set of n RTS tasks processed by a *uni-processor* RTS is represented by a task set $\mathbb{S} = \{\tau_1, \tau_2, \dots, \tau_i, \dots, \tau_n\}$. In accordance with [3], we refer to task instantiations as *task release* and to the respective instances as a *job*. Let $i \in \mathbf{n} := \{1, 2, \dots, n\}$. Each task τ_i is of a specific type that determines the mechanism of its release. On the basis of [23–26], this monograph addresses the task types *non-repetitive*, *sporadic*, and *periodic*. Depending on its type, a task refers to a parameter tuple that specifies the quantitative timing of its instantiation, the *processing time* required for each job, and a deadline that the *job completion* must satisfy. We first go through the parameters relevant to all types under consideration and then turn our attention to the individual types.

First-release time R_i . By the optional parameter $R_i \in \mathbb{N}$, we specify the *absolute clock time* at which the task τ_i is first released. When not explicitly given, the first release can occur at any time.

Execution time C_i . Once the task τ_i is released, the processing unit needs to be allocated to the respective job for a particular number time units in order for the job to be completed. As stated in Chap. 6, generally, the *exact execution time* of an RTS task is considered unknown but is guaranteed to be within the non-empty integer interval

$$C_i = [C_i^l, C_i^u] \in \mathbb{N} \times \mathbb{N},$$

where C_i^l and C_i^u are referred to as the *best-case execution time* (BCET) and the *worst-case execution time* (WCET), respectively. Clearly, this includes the special case of $C_i^l = C_i^u$ where the exact amount of time units is a known constant. In

this case, in Chaps. 4, 5, and 7, the execution time of task τ_i is denoted by C_i , i.e., $C_i = C_i^l = C_i^u$.

Relative deadline D_i . Referring to the absolute clock time at which a task τ_i is instantiated as the *release time*, the *relative deadline* $D_i \in \mathbb{N}$ is given as the maximum number of time units that may elapse from the release time until the respective job is completed. Throughout this monograph, we only consider hard deadlines, i.e., deadlines which must be kept unconditionally.

Period \mathbf{T}_i . The basic case for a periodic task τ_i is parameterized by a strict period (the time interval between two successive arrivals) of $T_i \in \mathbb{N}$ time units between any two successive releases. This is addressed by the classical real-time scheduling theory with deadline $D_i = T_i$ (see [15]) and deadlines $D_i \leq T_i$ (see [18]). In Chaps. 4 and 7, considering the dynamic reconfiguration in the modelling phase, we use the generalization of the so called *multi-period periodic tasks*, i.e., the period is specified by a non-empty interval

$$\mathbf{T}_i = [T_i^l, T_i^u] \in \mathbb{N} \times \mathbb{N},$$

where the number of time units that elapses between any two successive releases lies within \mathbf{T}_i .

Periodic tasks. Given any periodic task τ_i , its deadline $D_i \leq T_i^l$ guarantees job completion before the next release. We consider multi-period periodic tasks as a general case. For deadline $D_i \in (T_i^l, T_i^u]$, we impose the additional assumption that the task can arrive only when the previously instantiated job is completed. Deadlines $D_i > T_i^u$ are of no practical value and, hence, are not considered. Obviously, as studied in Chap. 4, our discussion includes strict periods by $T_i = T_i^l = T_i^u$ as a special case and, hence, we from now on use the terminology *periodic task* as a concise synonym for multi-period periodic task and we refer to T_i as its *period*.

Sporadic tasks. A sporadic task τ_i can be released at any time, provided that the job from the previous instantiation is completed, i.e., we do not consider queueing. Typical use cases are low-priority background operations like garbage collection. Parameters for this type are given as either

$$\tau_i = (\mathbf{C}_i, D_i)$$

or

$$\tau_i = (\mathbf{C}_i),$$

i.e., the deadline is optional. In contrast to non-repetitive tasks, we here implicitly refer to the first-release time $R_i = 0$. Again, the absence of a deadline may lead to an unboundedly postponed execution. Based on supervisory control theory, the presented scheduling policies ensure eventual job completion.

Non-repetitive tasks. Similar to sporadic tasks, a non-repetitive task can be released at any time. A non-repetitive task arrives only once, and hence it is viewed as a special case of a sporadic task.

We conclude by a summary of the task configurations that are addressed throughout this monograph:

- sporadic or non-repetitive task without a deadline: $\tau_i = (\mathbf{C}_i)$,
- sporadic or non-repetitive task with a deadline: $\tau_i = (\mathbf{C}_i, D_i)$, and
- periodic tasks with a deadline: $\tau_i = (R_i, \mathbf{C}_i, D_i, \mathbf{T}_i)$.

Notice that in the case of $C_i = C_i^l = C_i^u$ or $T_i = T_i^l = T_i^u$, \mathbf{C}_i and \mathbf{T}_i are replaced by C_i and T_i , respectively. Formally, a periodic task τ_i consists of an infinite sequence of jobs

$$J_{i,j} = (r_{i,j}, C_i, d_{i,j}, p_{i,j})$$

repeated periodically. The subscript “ i, j ” of $J_{i,j}$ with $i, j \in \mathbb{N}$ represents the j -th execution of task τ_i . For each j , $J_{i,j}$ requests the processor at absolute clock time $r_{i,j}$. The *absolute deadline* $d_{i,j}$ denotes the global clock time at which the execution of $J_{i,j}$ must be completed. Similarly, we define the *absolute release time* (resp., *period*) $r_{i,j}$ (resp., $p_{i,j}$) to mean the global clock time at which τ_i must be released (resp., start the next period). The execution of $J_{i,j}$ takes C_i time units, which must be completed no later than $d_{i,j}$. The absolute deadline $d_{i,j}$ occurs no later than the absolute period $p_{i,j}$.

In order to introduce the classical real-time scheduling and reconfiguration policies, the examples in this chapter are in the case of $C_i = C_i^l = C_i^u$ and $T_i = T_i^l = T_i^u$. Hence, such a periodic task with deadline is denoted by

$$\tau_i = (R_i, C_i, D_i, T_i).$$

Suppose that an RTS \mathbb{S} is a uni-processor system. The *processor utilization* of a periodic task τ_i is the fraction of processor time spent on its execution [15], i.e.,

$$U_i = \frac{C_i}{T_i}. \quad (3.1)$$

The total processor utilization of \mathbb{S} is

$$U_{\mathbb{S}} = \sum_{i=1}^n U_i. \quad (3.2)$$

A processor is non-schedulable in the case that there is overload [20], i.e., $U_{\mathbb{S}} > 1$.

Example Suppose that a uni-processor RTS \mathbb{S} executes four synchronous periodic tasks τ_1, τ_2, τ_3 , and τ_4 . Their parameters are shown in Table 3.1. \square

In past decades, most of the existing real-time scheduling algorithms are based on dynamic or fixed priorities [2, 7–9, 11–15, 17, 20, 27]. Leaving out the assigned scheduling policies, when real-time tasks are under execution, the processing of each individual job (belonging to a real-time task) falls into the following two categories:

- *preemptive*: a running job can be interrupted by the execution of other jobs, and
- *non-preemptive*: the execution of a running job cannot be interrupted.

3.2 Fixed Priority Scheduling

The studies in both [9] and [15] provide a sufficient utilization-based condition for feasibility scheduling of RTS when a set of tasks is assigned priorities according to a rate-monotonic (RM) policy. The work in [12] considers deadline monotonic (DM) scheduling, a *fixed-priority scheduling* of sets of tasks, which may have deadlines less than their periods. A DM policy assigns higher priorities to the tasks with shorter deadlines.

We choose a classical *fixed priority* scheduling algorithm, RM scheduling, to show the main idea of the scheduling policy. RM scheduling requires that the deadline of any task should be equal to its period. By RM, the task with a short period is assigned with the *highest priority*. Hence, the priorities of the tasks shown in Table 3.1 are provided in Table 3.2, in which the highest priority is labelled by 1 and the lowest priority is labelled by 4.

Example For the example shown in Table 3.1, by revising the deadlines to be equal to the periods, a preemptive uni-processor real-time RM scheduling is shown in the *Gantt chart* depicted in Fig. 3.1 (simulated by scheduling simulator Cheddar¹ [22]). According to the assigned priority, we have:

- in *time interval* $[0, 15)$, the tasks are processed in the order of τ_4 , τ_1 , τ_2 , and τ_3 ;

Table 3.1 Parameters of four synchronous tasks

Task	R_i	C_i	D_i	T_i
τ_1	0	4	12	20
τ_2	0	5	16	25
τ_3	0	5	18	30
τ_4	0	4	9	15

Table 3.2 Priorities of tasks

Task	Priority
τ_1	2
τ_2	3
τ_3	4
τ_4	1

¹ <http://beru.univ-brest.fr/cheddar/>.

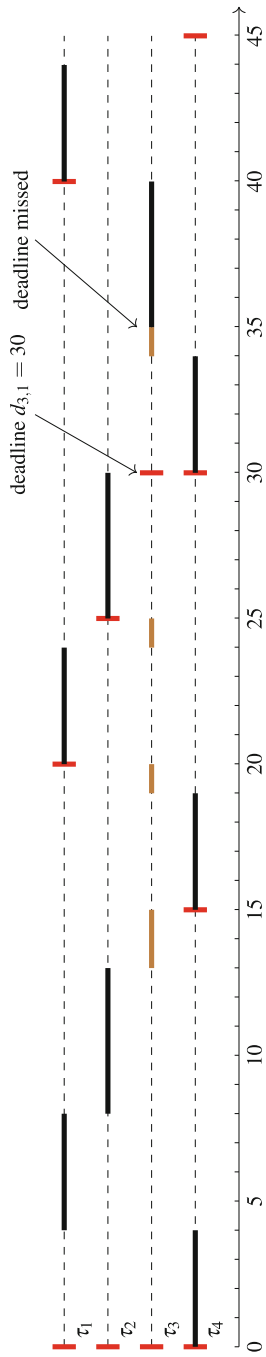


Fig. 3.1 RM scheduling of a synchronous task set with preemptive scheme

- at time $t = 15$, the second job of task τ_4 , denoted by $J_{4,2}$, arrives, which is assigned with the highest priority;
- in time interval $[15, 19)$, the execution of the first job of task τ_3 , denoted by $J_{3,1}$, is preempted by job $J_{4,2}$;
- at time $t = 19$, the execution of $J_{4,2}$ completes and its next job $J_{4,3}$ has not arrived. Now task τ_3 has the highest priority among all the available tasks. It enters the processor for execution again;
- in time interval $[20, 24)$, job $J_{3,1}$ is preempted by $J_{1,2}$;
- in time interval $[25, 30)$, job $J_{3,1}$ is preempted by $J_{2,2}$;
- in time interval $[30, 34)$, job $J_{3,1}$ is preempted by $J_{4,3}$;
- in time intervals $[19, 20)$, $[24, 25)$, and $[34, 35)$, job $J_{3,1}$ is under execution;
- at time $t = 35$, the execution of job $J_{3,1}$ completes, which misses its deadline $d_{3,1} = 30$.

We say that the task set shown in Table 3.1 is non-schedulable, in which the execution of the job missing its deadline is coloured brown. The Gantt charts in this monograph describing RTS scheduling follow the display interface of Cheddar.

In comparison, as depicted in Fig. 3.2, under the RM scheduling with non-preemptive scheme, the tasks shown in Table 3.1 are schedulable. For example, at time $t = 15$, even though task τ_4 has a higher priority than task τ_3 , the execution of the latter is not preempted. Then the execution of task τ_3 completes at time $t = 18$, which does not miss its deadline.

According to the work in [2], if a *synchronous RTS* is feasible, then, any derived *asynchronous RTS* is feasible too. In this case, for any asynchronous system, we can study its schedulability by analyzing the schedulability of its synchronous counterpart. However, it is not true on the other way round. As shown in Figs. 3.3 and 3.4, if we reset $R_4 = 6$, by RM scheduling, the real-time tasks are schedulable with both preemptive and non-preemptive schemes. \square

3.3 Dynamic Priority Scheduling

In [8], the author shows that, among all preemptive scheduling algorithms, the *earliest deadline first* (EDF) scheduling policy is optimal. If there exists a feasible scheduling for a task set, then the scheduling produced by EDF is also feasible. Under EDF scheduling, the analysis of periodic tasks with deadlines less than periods is proposed in [2]. *Least laxity first* (LLF), as another optimal algorithm, is proposed by Mok in [17], which assigns the processor to the *active tasks* with the smallest laxity. However, LLF has a larger overhead than EDF due to a larger number of context switches caused by laxity changes at run time.

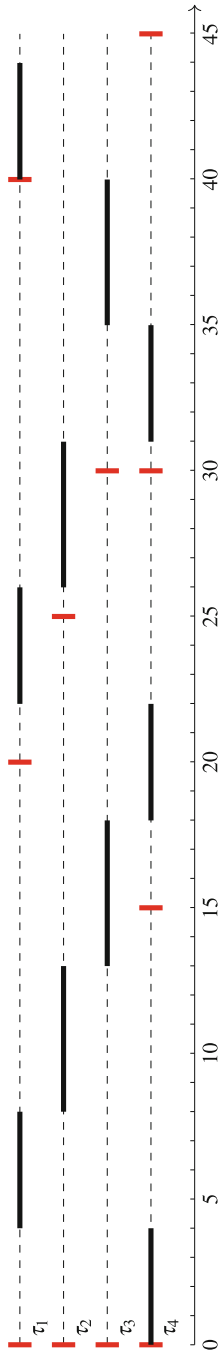


Fig. 3.2 RM scheduling of a synchronous task set with a non-preemptive scheme

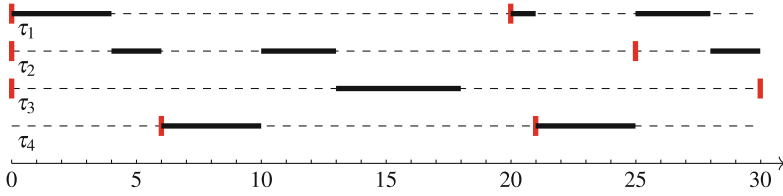


Fig. 3.3 RM scheduling of an asynchronous task set with a preemptive scheme

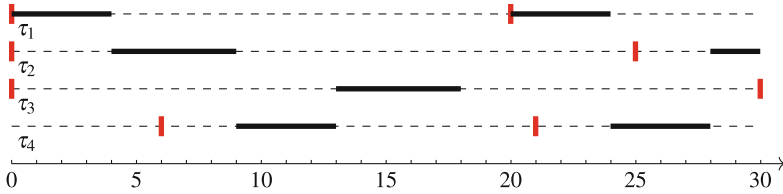


Fig. 3.4 RM scheduling of an asynchronous task set with a non-preemptive scheme

3.3.1 Earliest Deadline First Scheduling

EDF is a dynamic scheduling policy, where in any time unit, the tasks with the earliest deadlines have the highest dynamic priorities [15]. The EDF scheduling algorithm assigns the priority of a job based on the absolute deadlines: at each time unit, the job with the highest priority enters the processor. If the execution of a job is allowed to be preempted by other jobs before its execution finishes, the scheduling is preemptive; otherwise, it is non-preemptive.

Example Consider the four synchronous tasks shown in Table 3.1. The preemptive EDF scheduling result is depicted in Fig. 3.5. According to EDF scheduling scheme, we have:

- in time interval $[0, 15)$, the tasks are processed in the order of τ_4 , τ_1 , τ_2 , and τ_3 ;
- at time $t = 15$, the second job of task τ_4 , denoted by $J_{4,2}$, arrives, and its deadline is equal to $d_{4,2} = 15 + 9 = 24$ which is later than $J_{3,1}$'s deadline $d_{3,1} = 18$. Hence, τ_3 has the highest priority;
- in time interval $[15, 18)$, job $J_{3,1}$ continues its execution;
- at time $t = 18$, after the execution of job $J_{3,1}$ completes, the second job of task τ_4 , denoted by job $J_{4,2}$, enters the processor for execution;
- at time $t = 20$, when job $J_{1,2}$ is released, its deadline $d_{1,2} = 20 + 12 = 32$ is later than $d_{4,2} = 24$. Hence, the execution of job $J_{1,2}$ cannot preempt job $J_{4,2}$ until its execution completes;
- in time interval $[18, 22)$, job $J_{4,2}$ is under execution;
- at time $t = 22$, after the execution of job $J_{4,2}$ completes, job $J_{1,2}$ enters the processor for execution;

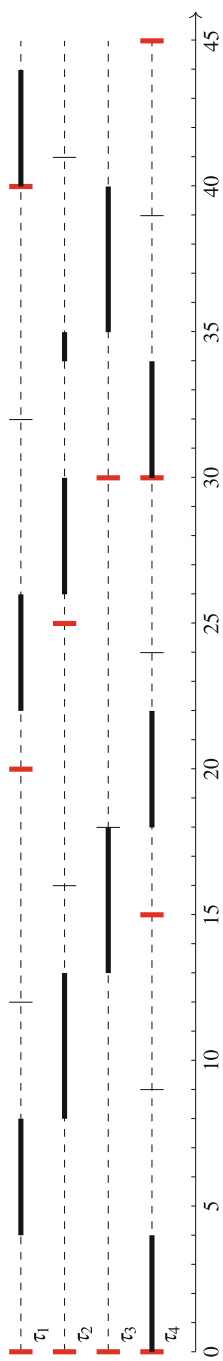


Fig. 3.5 EDF scheduling of a synchronous task set with preemptive scheme

- at time $t = 25$, when job $J_{2,2}$ is released, its deadline $d_{2,2} = 25 + 16 = 41$ is later than $d_{1,2} = 32$. Hence, the execution of job $J_{2,2}$ cannot preempt job $J_{1,2}$ until its execution completes;
- in time interval $[22, 26)$, job $J_{1,2}$ is under execution;
- in time interval $[26, 30)$, job $J_{2,2}$ is under execution;
- in particular, at time $t = 30$, job $J_{2,2}$ is still under execution at the arrival of jobs $J_{3,2}$ and $J_{4,3}$. We have $d_{2,2} = 41$, $d_{3,2} = 30 + 18 = 48$, and $d_{4,3} = 15 \times 2 + 9 = 39$. Clearly, job $J_{4,3}$ has the highest priority. Hence, the execution of job $J_{2,2}$ is preempted by job $J_{4,3}$;
- in time interval $[30, 34)$, job $J_{4,3}$ is under execution;
- at time $t = 34$, we have $d_{2,2} = 41$ and $d_{3,2} = 48$. Clearly, job $J_{2,2}$ has the highest priority. Hence, the execution of job $J_{2,2}$ continues in time interval $[34, 35)$;
- at time $t = 35$, after the execution of job $J_{2,2}$ completes, job $J_{3,2}$ enters the processor for execution;
- in time interval $[35, 40)$, job $J_{3,2}$ is under execution;
- at time $t = 40$, job $J_{1,3}$ has the highest priority since no other jobs are under execution or preempted;
- in time interval $[40, 44)$, job $J_{1,3}$ is under execution.

In comparison, the non-preemptive EDF scheduling result is depicted in Fig. 3.6. At time $t = 30$, the execution of job $J_{2,2}$ cannot be preempted by job $J_{4,3}$. Job $J_{4,3}$ can only enter the processor by time $t = 31$ at which the execution of job $J_{2,2}$ completes. By EDF scheduling, the real-time tasks are schedulable with both preemptive and non-preemptive schemes.

Suppose $R_4 = 6$. The EDF scheduling of the four tasks under preemptive and non-preemptive schemes is depicted in Figs. 3.7 and 3.8, respectively. By EDF scheduling, the real-time tasks are schedulable with both preemptive and non-preemptive schemes. \square

3.3.2 Least Laxity First Scheduling

Mok presented another optimal algorithm least laxity first (LLF) in [17], which assigns the processor to the active task with the smallest laxity. The laxity of a job is the difference between its absolute deadline and the remaining time units needed for finishing its execution.

Example For the same example shown in Table 3.1, between $t = 0$ and $t = 18$, we provide the laxity of each task in Table 3.3, in which the laxity is denoted by a number, and “exe” represents that, in the following time unit, the corresponding task is under execution. For example, at time $t = 0$, tasks τ_1 , τ_2 , τ_3 , and τ_4 are released synchronously with laxity of $12 - 4 = 8$, $16 - 5 = 11$, $18 - 5 = 13$, and $9 - 4 = 5$, respectively. Hence, task τ_4 has the least laxity and enters the processor for execution. At $t = 5$, the execution of τ_4 completes. It is not an active task until

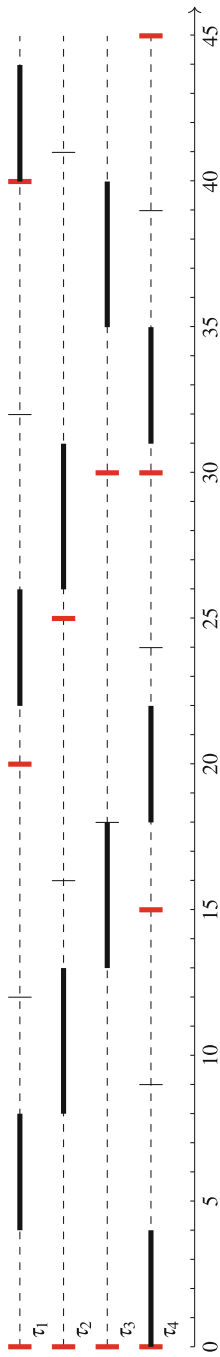


Fig. 3.6 EDF scheduling of a synchronous task set with a non-preemptive scheme

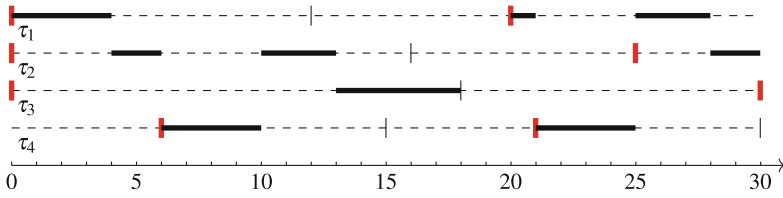


Fig. 3.7 EDF scheduling of an asynchronous task set with preemptive scheme

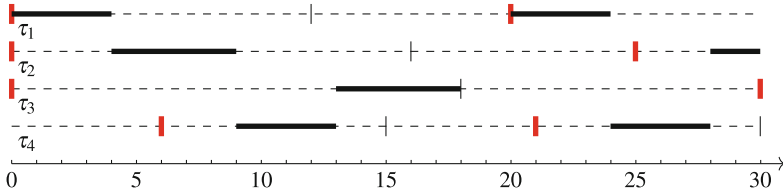


Fig. 3.8 EDF scheduling of an asynchronous task set with a non-preemptive scheme

Table 3.3 Laxity of four synchronous tasks

Time units	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
τ_1	8	7	6	5	5	4	4	4	-	-	-	-	-	-	-	-	-	-	-
τ_2	11	10	9	8	7	6	5	4	3	3	3	3	2	2	1	-	-	-	-
τ_3	13	12	11	10	9	8	7	6	5	4	3	2	2	1	1	0	0	0	-
τ_4	5	5	5	5	4	-	-	-	-	-	-	-	-	-	-	5	4	3	2
exe	τ_4	τ_4	τ_4	τ_1	τ_4	τ_1	τ_1	τ_1	τ_2	τ_2	τ_2	τ_3	τ_2	τ_3	τ_2	τ_3	τ_3	τ_3	τ_4

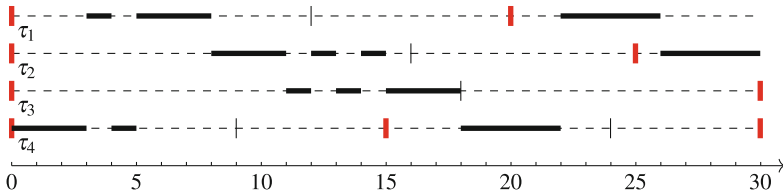


Fig. 3.9 LLF scheduling of a synchronous task set with a preemptive scheme

the next job releases. As a result, it is not under consideration, denoted by “-” in Table 3.3. Finally, the real-time LLF scheduling under preemptive scheme is shown in Fig. 3.9. Note that at time $t = 3$, $t = 7$, $t = 10$, $t = 12$, and $t = 14$, there is more than one task that has the least laxity. It is clear that we can assign the highest priority to any of them. Obviously, Fig. 3.9 only provides one of many schedulable sequences. If we keep tracking the scheduling sequence, we find that some task deadlines will be missed. Hence, the task set is not schedulable.

Suppose $R_4 = 6$. We provide the laxity of each task in Table 3.4. Due to the same approach, their laxities between $t = 0$ and $t = 18$ are given in Table 3.3, and the corresponding LLF scheduling is depicted in Fig. 3.10. Note that at time $t = 3$,

Table 3.4 Laxity of four asynchronous tasks

Time units	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
τ_1	8	8	8	8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
τ_2	11	10	9	8	7	7	7	6	5	5	4	3	3	2	1	-	-	-	-
τ_3	13	12	11	10	9	8	7	6	5	4	4	3	2	2	1	0	0	0	-
τ_4	-	-	-	-	-	-	5	5	5	4	3	3	2	1	-	-	-	-	-
exe	τ_1	τ_1	τ_1	τ_1	τ_2	τ_2	τ_4	τ_4	τ_2	τ_3	τ_4	τ_2	τ_3	τ_4	τ_2	τ_3	τ_3	τ_3	-

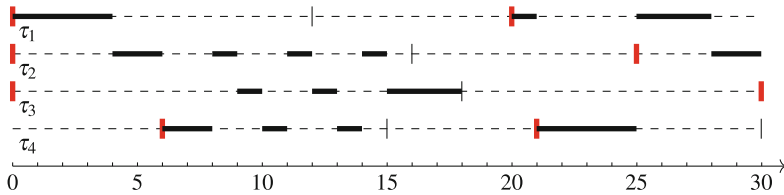


Fig. 3.10 LLF scheduling of an asynchronous task set with a preemptive scheme

$t = 6, t = 7, t = 8, t = 10, t = 11, t = 12,$ and $t = 13,$ there is more than one task that has the least laxity. It is clear that we can assign the highest priority to any of them. Obviously, Fig. 3.10 only provides one of many schedulable sequences. The real-time tasks are schedulable. \square

The non-preemptive LLF scheduling and the EDF scheduling of either the synchronous task set or the asynchronous task set are identical, as depicted in Figs. 3.6 and 3.8, respectively. Although both LLF and EDF are optimal algorithms, LLF has a larger overhead due to a larger number of context switches caused by laxity changes at run time.

3.4 Elastic Period Model for Reconfiguration

Real-time reconfigurations are of critical importance to RTS. A *reconfiguration scenario* can be the addition/removal/update of the tasks at run-time in order to save the whole system when random disturbances occur. There has been a fair amount of significant research from academia and industry [10, 19, 21, 28, 29] for real-time reconfiguration, which are based on fixed or dynamic priority-based scheduling with preemptive/non-preemptive schemes. In principle, two sets of reconfiguration scenarios can be identified:

- static reconfiguration scenario applied *offline* before any system’s *cold start* [1], and
- dynamic reconfiguration scenario applied at run-time [30].

As a dynamic reconfiguration scenario, the elastic period task model is proposed in [4–6, 16] to handle the *overload* of an RTS by decreasing its task processor

Table 3.5 Parameters of four synchronous tasks

Task	R_i	C_i	D_i	\mathbf{T}_i
τ_1	0	4	12	[20, 25]
τ_2	0	5	16	[25, 30]
τ_3	0	5	18	[30, 35]
τ_4	0	4	9	[15, 20]

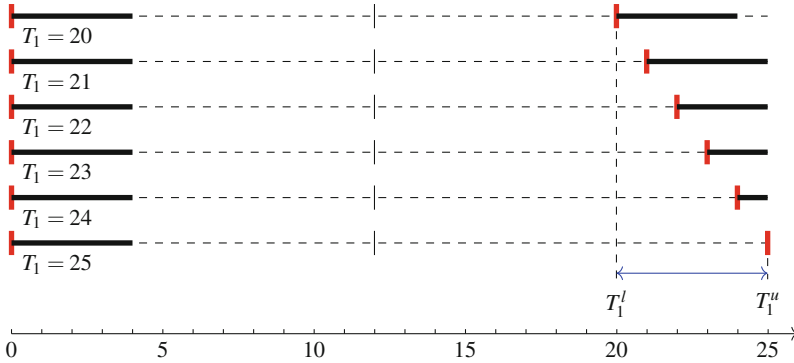


Fig. 3.11 Task τ_1 with an elastic period

utilization. The main idea is to consider the period of each task running in the RTS as a spring with a given rigidity coefficient and length constraints. To reconfigure each task is to modify its period, so that its processor utilization can be changed within a specified range. Hence, the proposed model can be used to handle overload situations in a flexible way.

Example While new tasks arrive, instead of rejecting them, we can remodel the tasks listed in Table 3.1 by assigning periods with upper bounds. Now the tasks in Table 3.1 are reconfigured to be elastic period tasks. Their parameters are listed in Table 3.5, in which $\mathbf{T}_i = [T_i^l, T_i^u]$ indicates a lower bound T_i^l and an upper bound T_i^u . In particular, T_i^l is identical with T_i in Table 3.1. Without considering the exact execution time of task τ_1 , we depict its six possible periods in Fig. 3.11. Clearly, for task τ_1 , its processor utilization $\frac{4}{25}$ (with T_i^u assigned) is smaller than $\frac{4}{20}$ (with T_i^l assigned). \square

References

1. Angelov, C., Sierszecki, K., Marian, N.: Design models for reusable and reconfigurable state machines. In: International Conference on Embedded and Ubiquitous Computing, pp. 152–163. Springer, Berlin (2005)
2. Baruah, S.K., Rosier, L.E., Howell, R.R.: Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Syst.* **2**(4), 301–324 (1990)

3. Buttazzo, G.C.: *Hard real-time computing systems: predictable scheduling algorithms and applications*, vol. 24. Springer Science & Business Media (2011)
4. Buttazzo, G., Abeni, L.: Adaptive workload management through elastic scheduling. *Real-Time Syst.* **23**, 7–24 (2002)
5. Buttazzo, G.C., Lipari, G., Abeni, L.: Elastic task model for adaptive rate control. In: *IEEE Real-Time Systems Symposium*, pp. 286–295 (1998)
6. Buttazzo, G.C., Lipari, G., Caccamo, M., Abeni, L.: Elastic scheduling for flexible workload management. *IEEE Trans. Comput.* **51**(3), 289–302 (2002)
7. Davis, R.I.: A review of fixed priority and EDF scheduling for hard real-time uniprocessor systems. *ACM SIGBED Rev.* **11**(1), 8–19 (2014)
8. Dertouzos, M.L.: Control robotics: the procedural control of physical processes. In: *IF IP Congress*, pp. 807–813 (1974)
9. Fineberg, M.S., Serlin, O.: Multiprogramming for hybrid computation. In: *Fall Joint Computing Conference*, pp. 1–13 (1967)
10. Gaujal, B., Navet, N.: Dynamic voltage scaling under EDF revisited. *Real-Time Syst.* **37**(1), 77–97 (2007)
11. Howell, R.R., Venkatrao, M.K.: On non-preemptive scheduling of recurring tasks using inserted idle times. *Inf. Comput.* **117**(1), 50–62 (1995)
12. Leung, J.Y.T., Whitehead, J.: On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Perform. Eval.* **2**(4), 237–250 (1982)
13. Li, J., Shu, L., Chen, J., Li, G.: Energy-efficient scheduling in nonpreemptive systems with real-time constraints. *IEEE Trans. Syst. Man Cybern. Syst.* **43**(2), 332–344 (2013)
14. Li, D., Li, M., Meng, X., Tian, Y.: A hyperheuristic approach for intercell scheduling with single processing machines and batch processing machines. *IEEE Trans. Syst. Man Cybern. Syst.* **45**(2), 315–325 (2015)
15. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM* **20**(1), 46–61 (1973)
16. Marinoni, M., Buttazzo, G.: Elastic DVS management in processors with discrete voltage/frequency modes. *IEEE Trans. Ind. Inf.* **3**(1), 51–62 (2007)
17. Mok, A.K.: *Fundamental design problems of distributed systems for the hard-real-time environment*. Ph.D. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge (1983)
18. Nassor, E., Bres, G.: Hard real-time sporadic task scheduling for fixed priority schedulers. In: *International Workshop on Responsive Systems*, pp. 44–47 (1991)
19. Quan, G., Hu, X.S.: Minimal energy fixed-priority scheduling for variable voltage processors. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **22**(8), 1062–1071 (2003)
20. Sha, L., Abdelzaher, T., Cervin, A., Baker, T., Burns, A., Buttazzo, G., Caccamo, M., Lehoczky, J., Mok, A.K.: Real time scheduling theory: a historical perspective. *Real-Time Syst.* **28**(2), 101–155 (2004)
21. Shin, Y., Choi, K.: Power conscious fixed priority scheduling for hard real-time systems. In: *Design Automation Conference*, pp. 134–139. IEEE, New Orleans (1999)
22. Singhoff, F., Legrand, J., Nana, L., Marcé, L.: Cheddar: a flexible real time scheduling framework. *ACM SIGAda Ada Lett.* **4**, 1–8 (2004)
23. Wang, X., Khemaissia, I., Khalgui, M., Li, Z., Mosbahi, O., Zhou, M.: Dynamic low-power reconfiguration of real-time systems with periodic and probabilistic tasks. *IEEE Trans. Autom. Sci. Eng.* **12**(1), 258–271 (2015)
24. Wang, X., Li, Z., Wonham, W.M.: Dynamic multiple-period reconfiguration of real-time scheduling based on timed DES supervisory control. *IEEE Trans. Ind. Inf.* **12**(1), 101–111 (2016)
25. Wang, X., Li, Z., Wonham, W.M.: Optimal priority-free conditionally-preemptive real-time scheduling of periodic tasks based on DES supervisory control. *IEEE Trans. Syst. Man Cybern. Syst.* **47**(7), 1082–1098 (2017)
26. Wang, X., Li, Z., Wonham, W.M.: Priority-free conditionally-preemptive scheduling of modular sporadic real-time systems. *Automatica* **89**, 392–397 (2018)

27. Xia, Y., Zhou, M., Luo, X., Pang, S., Zhu, Q.: A stochastic approach to analysis of energy-aware DVS-enabled cloud datacenters. *IEEE Trans. Syst. Man Cyber. Syst.* **45**(1), 73–83 (2015)
28. Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. In: *Annul Symposium on Founation Computer Science*, pp. 374–382 (1995)
29. Yun, H.S., Kim, J.: On energy-optimal voltage scheduling for fixed-priority hard real-time systems. *ACM Trans. Embed. Comput. Syst.* **2**(3), 393–430 (2003)
30. Zhang, J., Khalgui, M., Li, Z., Mosbahi, O., Alahmari, A.M.: R-TNCES: a novel formalism for reconfigurable discrete event control systems. *IEEE Trans. Syst. Man Cybern. Syst.* **43**(4), 757–772 (2013)

Chapter 4

Non-Preemptive Scheduling/Reconfiguration Based on Supervisory Control of TDES



4.1 Introduction

Historically, *real-time systems* (RTS) were constructed in an *ad hoc* manner, which is scheduled by *cyclic executives* [8]. An RTS consists of a number of tasks with explicit *timing requirements*, which could be a water vessel system, computer numerical control machine, a robot, or an assembly-line worker. In the literature, RTS are usually scheduled by two approaches. One is to divide the scheduling problem in two steps:

- determine whether the RTS is schedulable, and
- find an algorithm to generate a *safe execution sequence*.

The other approach works in reverse: a candidate (existing) schedule scheme is first generated, and then it is checked whether all the tasks in the RTS meet their deadlines.

The study in [5] is the first work to combine RTS scheduling and *supervisory control theory* (SCT). Based on the nonblocking supervisory control of *timed discrete-event systems* (TDES), Chen and Wonham treat the overall scheduling problem as an integral problem. In the proposed method, solving the schedulability problem of an RTS implies solving its scheduling-algorithm problem and vice versa.

In [5], Chen and Wonham propose a TDES-based task model and a real-time scheduling technique. The behavior of real-time tasks, such as *task release/arrival* and their *execution starting/completion*, is represented by *active events*. The *temporal characteristics* are considered to build a *timed transition graph* (TTG) that describes the possible behavior of real-time tasks. Any RTS tasks with deadlines less than or equal to their periods can be modelled by TTG in this approach. Given an RTS processing such tasks, with user-defined *non-preemptive scheduling specifications*, the safe execution sequences are found by the supervisory control of TDES. An RTS is claimed to be *non-schedulable* if the supervisor is empty. The

proposed scheduling approach is different from the traditional real-time scheduling approaches:

- The work in [5] does not check the schedulability and find safe scheduling sequences separately. Since the found safe scheduling sequences are included in a proper supervisor for the TDES, their timing characteristics do not need to be verified.
- The traditional scheduling algorithms only provide one safe execution for the real-time scheduling. However, the work in [5] provides all the possible safe execution sequences.

The research in [5] is a significant improvement over real-time scheduling. However, the authors do not reconfigure the system in the case of nonschedulability. Later, *dynamic reconfigurations* of RTS are addressed in [10] and we summarize it in this chapter.

In [2–4, 7], an *elastic period task model* is proposed to handle the overload of an RTS by decreasing the task processor utilization. Building on the two latter studies, we present a new modelling technique to endow the real-time tasks with multi-periods. To handle the overload of an RTS, SCT is utilized to find all the possible solutions based on different periods of each task. For any solution, all the safe execution sequences are provided. The presented dynamic reconfiguration approach consists of two steps:

- The initial model of any task is assigned with the *shortest period* (i.e., assigned with the *highest processor utilization*), and by utilizing SCT, all the RTS' safe execution sequences (if any) are found.
- For the purpose of reconfiguring the RTS in the case of nonschedulability, this monograph reconfigures the RTS' composite task model by assigning multi-periods to the tasks.

The multi-period provides multiple processor utilization for a task. Consequently, a *processor utilization interval* for the RTS is obtained. SCT is utilized again to find all the safe execution sequences (possible reconfiguration scenarios) in the predefined processor utilization interval. If the supervisor is still empty, we claim that the RTS is non-schedulable. A real-world example is implemented in this chapter. The results illustrate that, in the dynamic reconfiguration approach, the presented method finds a set of safe execution sequences.

Clearly, the presented multi-period model is a uniformed model that can schedule and reconfigure RTS (if necessary). Building on this idea, in the following chapters, we present similar multi-period models to schedule and/or reconfigure RTS based on supervisory control of discrete-event systems (Chaps. 5 and 6) and supervisory control of state-tree structures (Chap. 7).

The rest of this chapter is structured as follows. The TDES model for RTS scheduling and reconfiguration is defined in Sect. 4.2. Section 4.3 reports methodologies of dynamic scheduling and reconfiguration of RTS. A real-world example is implemented in Sect. 4.4 to verify the dynamic scheduling and reconfiguration. Conclusions are provided in Sect. 4.5.

4.2 RTS Modelled by Timed Discrete-Event Systems

In [5], Chen and Wonham propose a TDES modelling mechanism to model real-time tasks running in uni-processor RTS and schedule them non-preemptively. Later the TDES model is generalized in [10]. A generalized TDES modelling method is provided to assign a *multi-period* that is a set of possible periods for a real-time task. The default period for a task is the shortest one. In the case that the RTS is non-schedulable, based on supervisory control, the multi-period is used to reconfigure the RTS automatically. Generally, the TDES task model proposed in [5] can be viewed as a special case of the one presented in [10].

4.2.1 Multi-Period RTS Task Model

In this section, we provide a general TDES modelling mechanism to represent the execution of periodic real-time tasks. Such a TDES model can be utilized to schedule or reconfigure RTS. Similar to the *elastic task model* [2–4, 7], we assume that a task is associated with a multi-period, i.e., having a lower bound and an upper bound. By assigning multi-periods to real-time tasks, the presented TDES model generalizes the TDES model proposed in [5].

Suppose that a periodic RTS \mathbb{S} processes n tasks, i.e., $\mathbb{S} = \{\tau_1, \tau_2, \dots, \tau_i, \dots, \tau_n\}$, $i \in \mathbf{n} = \{1, 2, \dots, n\}$. The execution model of an RTS is a set of tasks processed in a uni-processor, in which a task τ_i is described by

$$\tau_i = (R_i, C_i, D_i, \mathbf{T}_i)$$

with

- a *release time* R_i ,
- a *worst-case execution time (WCET)* C_i ,
- a *deadline* D_i , and
- a *multi-period* \mathbf{T}_i .

A multi-period is specified by a non-empty interval

$$\mathbf{T}_i = [T_i^l, T_i^u] \in \mathbb{N} \times \mathbb{N},$$

where the number of time units that elapse between any two successive releases lies within \mathbf{T}_i . Hence a multi-period has a lower bound (i.e., shortest one) represented by T_i^l and an upper bound (i.e., longest one) represented by T_i^u . As stated in Sect. 3.1, a periodic task τ_i consists of an infinite sequence of *jobs* repeated periodically that are represented by a corresponding four-tuple

$$J_{i,j} = (r_{i,j}, C_i, d_{i,j}, p_{i,j}).$$

The subscript “ i, j ” of $J_{i,j}$ represents the j -th job execution of task τ_i .

During the real-time scheduling process, for task τ_i , only one period T satisfying

$$T_i^l \leq T \leq T_i^u$$

is selected in a scheduling period. The processor utilization U_i of task τ_i is calculated by

$$U_i = \frac{C_i}{T}. \quad (4.1)$$

The total processor utilization of \mathbb{S} is

$$U_{\mathbb{S}} = \sum_{i=1}^n U_i. \quad (4.2)$$

According to [8], an RTS \mathbb{S} is non-schedulable if $U_{\mathbb{S}} > 1$.

A task is *non-reconfigurable* if its multi-period satisfies $T_i^l = T_i^u$, which is considered as a traditional RTS task. In particular, such a task is identical with the task model proposed by Chen and Wonham in [5]. The dynamic reconfiguration of a real-time task consists of two steps given below:

- Initially, a task with $T = T_i^l$ plays the role of the task proposed in [5]. In this case, task τ_i always stays at the highest processor utilization.
- If the RTS is non-schedulable, the multi-period model with $\mathbf{T}_i = [T_i^l, T_i^u]$ is utilized to provide all the possibilities to compress the processor utilization. By integrating dynamic reconfigurations, our study in Chaps.6 and 7 is built on multi-periods.

4.2.2 Real-Time Tasks Modelled by Timed Discrete-Event Systems

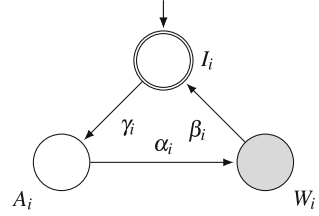
Given a real-time periodic task $\tau_i = (R_i, C_i, D_i, \mathbf{T}_i)$ with $i \in \mathbf{n}$ and $D_i \leq T_i$, represented by a TDES, as depicted in Fig. 4.1 (from [5]), its corresponding *activity transition graph* (ATG) is

$$\mathbf{G}_{act,i} = (A_i, \Sigma_{act,i}, \delta_{act,i}, a_{0,i}, A_{m,i}).$$

States I_i , A_i , and W_i represent *idle*, *arrival*, and *work*, respectively. The events in the alphabet $\Sigma_{act,i}$ are

- γ_i : the event that τ_i is released,
- α_i : the event that the execution of τ_i is started, and
- β_i : the event that the execution of τ_i is completed.

Fig. 4.1 ATG of a real-time task from [5]



Event α_i is *controllable* and events γ_i and β_i are *uncontrollable*. Moreover, all the events in $\Sigma_{act,i}$ are *forcible* [1]. In this monograph, a state filled with gray, say W_i , represents that the corresponding task is under execution.

An ATG describes only the logical behavior of a TDES. In the transition graph of the ATG, the *initial state* is labeled with an *entering arrow*, and a *marker state* is represented by a *double circle*.

Suppose that, after enabling, events γ_i , α_i , and β_i should wait for t_{γ_i} , t_{α_i} , and t_{β_i} ticks, respectively, until they are eligible to occur. Thus, t_{α_i} is the time at which τ_i starts its execution. Furthermore, in the TDES model, we have

$$t_{\beta_i} = C_i.$$

A TDES model describing an RTS task's behavior has the following two features:

- γ_i signals that after $r_{i,1}$, τ_i will release periodically, and
- β_i must occur before τ_i is released again.

The time interval between the occurrences of events β_i and γ_i is the remaining time of the current period, which decreases along with the increase of t_{α_i} . Hence, in two adjacent periods, the values of t_{γ_i} could be different. Formally, let $T_i^l \leq T \leq T_i^u$; we have:

- γ_i has time bounds $\begin{cases} [0, 0], & \text{if } \tau_i \text{ releases at } r_{i,1} \\ [T - t_{\alpha_i} - t_{\beta_i}, T - t_{\alpha_i} - t_{\beta_i}], & \text{if } (\forall j > 1) \tau_i \text{ releases at } r_{i,j} \end{cases}$,
- α_i has time bounds $[0, D_i - t_{\beta_i}]$, and
- β_i has time bounds $[t_{\beta_i}, t_{\beta_i}]$.

Generally, the presented task model assigns a lower and an upper period bound for each task to dynamically reconfigure an RTS \mathbb{S} . At each time, a task's period is assigned with a value between the two bounds T_i^l and T_i^u . Consequently, the processor utilization of an elastic periodic task τ_i has a lower bound U_i^l and an upper bound U_i^u . Formally, we conclude that the processor utilization of τ_i is in an interval

$$\mathbf{U}_i = [U_i^l, U_i^u]$$

with

$$U_i^l = \frac{C_i}{T_i^u}$$

and

$$U_i^u = \frac{C_i}{T_i^l}.$$

The system processor utilization of \mathbb{S} is in an interval

$$\mathbf{U}_{\mathbb{S}} = [U^l, U^u]$$

with

$$U^l = \sum_{i=1}^n U_i^l$$

and

$$U^u = \sum_{i=1}^n U_i^u.$$

In the possible processor utilization interval $[U^l, U^u]$, there may exist multiple safe execution sequences (reconfiguration scenarios) that correspond to different processor utilizations.

From the perspective of TDES, on the occurrence of event α_i , the processor starts the processing of the current job of τ_i . After the global *tick* event t occurs C_i times, the execution of τ_i is completed. The next occurrence of event γ_i drives τ_i into the next execution period.

In this section, a multi-period RTS task represented by the TTG of a TDES

$$\mathbf{G}_i = (Q_i, \Sigma_i, \delta_i, q_{0,i}, Q_{m,i})$$

is depicted in Fig. 4.2. Let $w = \min\{D_i, T_i^l\}$. We have

- y_l is the initial state, representing that the processor is in an idle operation,
- state y_k^α with $0 \leq k \leq w - C_i$: before starting the execution of task τ_i , k time units have passed since the recent release of task τ_i ,
- state $y_{k,p}$ with $0 \leq k \leq w - C_i$ and $0 \leq p \leq C_i$: starting the p -th time unit execution of task τ_i at the k -th time unit,
- state y_k^β with $C_i \leq k \leq w$: the targeting state of event β_i , representing that the execution of τ_i is completed at the k -time unit,

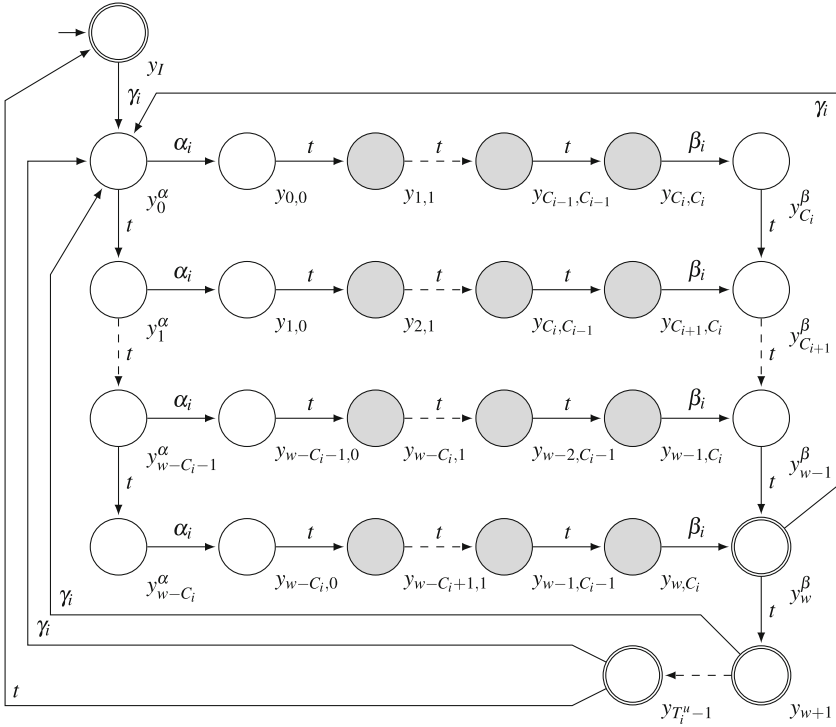


Fig. 4.2 General TTDG model for multi-period tasks

- states y_k with $w < k < T_i^u$: the execution of task τ_i is completed and k time units have elapsed since the recent release of task τ_i , and
- $\{y_w^\beta, y_{w+1}, \dots, y_{T_i^u-1}, y_I\}$: the marker state set.

A marker state represents that τ_i has finished the current execution of job $J_{i,j}$ and is ready for the release of the next job $J_{i,j+1}$. Marker state y_I represents that job $J_{i,j}$ finishes its operation at time $t = T_i^u$ or has never been invoked. Marker states y_w^β , y_{w+1} , and $y_{T_i^u-1}$ represent that job $J_{i,j}$ finishes its operation at time units $t = T_w$, $t = T_{w+1}$, and $t = T_i^u - 1$, respectively, with $C_i \leq w \leq T_i$. Transition $\delta(y_{0,0}, t) = y_{1,1}$ leads the system from a normal state to a state filled with gray, which represents that the task τ_i is under execution for one tick (or equally one time unit). Throughout this monograph, any state filled with gray represents that the corresponding task is under execution.

The remaining time between the occurrences of events β_i and γ_i equals 0 if

- $D_i = T_i$, and
- α_i occurs at time $D_i - t_{\beta_i}$.

As a result, the occurrence of β_i may lead the TDES model to state y_I directly.

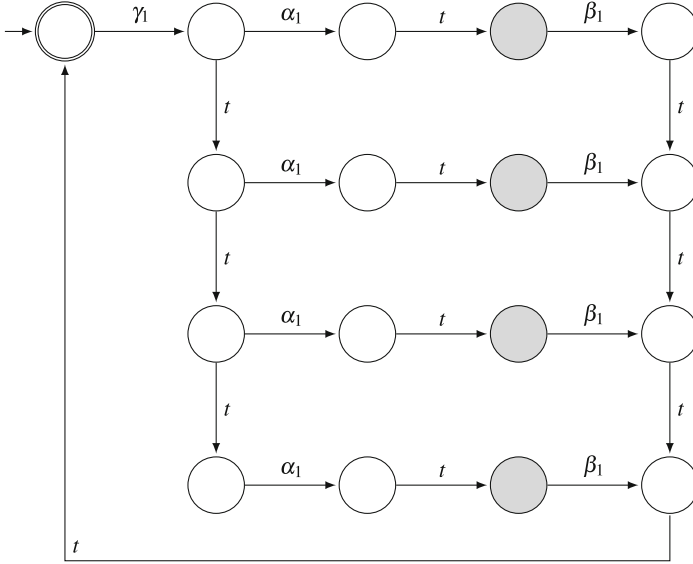


Fig. 4.3 TTG G_1 for task τ_1

Example A non-reconfigurable periodic task τ_1 is defined as $\tau_1 = (0, 1, 4, [5, 5])$. The processor utilization of task τ_1 is fixed to be $U_1 = \frac{1}{5}$. The TTG model G_1 for task τ_1 is depicted in Fig. 4.3. For the events in $\Sigma_{act,1}$,

- γ_1 has time bounds $\begin{cases} [0, 0], & \text{if } \tau_1 \text{ releases at } r_{1,1} \\ [4 - t_{\alpha_1}, 4 - t_{\alpha_1}], & \text{if } (\forall j > 1) \tau_1 \text{ releases at } r_{1,j} \end{cases}$,
- α_1 has time bounds $[0, 4 - t_{\beta_1}]$, and
- β_1 has time bounds $[1, 1]$.

Suppose that we have two other tasks $\tau_2 = (0, 2, 6, [4, 6])$ and $\tau_3 = (0, 2, 5, [3, 5])$. The corresponding TTG models G_2 and G_3 are illustrated in Figs. 4.4 and 4.5, respectively, in which events β_2 and β_3 lead the TDES model to the initial states directly. All the possible processor utilizations of τ_2 are $\frac{2}{4}$, $\frac{2}{5}$, and $\frac{2}{6}$; and all the possible processor utilizations of τ_3 are $\frac{2}{3}$, $\frac{2}{4}$, and $\frac{2}{5}$. \square

Remarks

1. In this monograph, in the TDES or DES representing the behavior of RTS tasks, all the states are named as y_{\cdot} .
2. In the presented TDES model, the time bounds $[T_i^l - t_{\alpha_i} - t_{\beta_i}, T_i^u - t_{\alpha_i} - t_{\beta_i}]$ for event γ_i with $j \geq 1$ are dynamic, which decrease while t_{α_i} increases. However, the TDES synthesis tool TTCT¹ can convert an ATG into a TTG only if the

¹ <http://www.control.utoronto.ca/DES>.

events have fixed time bounds. Hence, according to the presented TDES model, users need to create the TTG for every real-time task directly. \square

4.2.3 Global RTS Execution Model

Suppose that there exists a set of tasks to be executed in an RTS. A TDES representing its monolithic behavior is required. One standard way to construct such a “composite” TDES for a set of tasks represented by TTGs is to apply the *synchronous product* [11] over the individual TTG.

Suppose that two tasks τ_1 and τ_2 are executed in an RTS \mathbf{G} . Their system behaviors are represented by $L(\mathbf{G}_1)$ and $L(\mathbf{G}_2)$, respectively; their marked behaviors are represented by $L_m(\mathbf{G}_1)$ and $L_m(\mathbf{G}_2)$, respectively. Through the synchronous product, the TTG of τ_1 and τ_2 are combined into a TTG representing the system behavior of \mathbf{G} . Formally, we have

$$L(\mathbf{G}) = L(\mathbf{G}_1) || L(\mathbf{G}_2)$$

and

$$L_m(\mathbf{G}) = L_m(\mathbf{G}_1) || L_m(\mathbf{G}_2).$$

Note that the only shared event between $L(\mathbf{G}_1)$ and $L(\mathbf{G}_2)$ is event t . The obtained composite task contains all execution sequences that meet the deadlines of the two tasks, but with the following two implicit assumptions:

- Resource-sharing tasks τ_i satisfying $D_i = T_i^l = T_i^u$: resources are available to execute the tasks concurrently. This result is illustrated in the example stated below.
- Independent tasks τ_i satisfying $T_i^l \leq T_i^u$ and $D_i < T_i^u$: As illustrated in Fig. 4.2, string t^* (a string consists of $n \geq 0$ tick events) may occur between events β_i and γ_i . Given two tasks τ_i and τ_j , there may exist a string $\beta_j t^* (\alpha_i t^{C_i} \beta_i) t^* \gamma_j$ representing that they are executed in sequence.

Example Figure 4.6 depicts the TTG of two tasks $\tau_1 = (0, 2, 2, 2)$ and $\tau_2 = (0, 1, 1, 1)$. The composite task execution model, obtained by $L(\mathbf{G}) = L(\mathbf{G}_1) || L(\mathbf{G}_2)$ and $L_m(\mathbf{G}) = L_m(\mathbf{G}_1) || L_m(\mathbf{G}_2)$, has 20 states and 29 transitions, which is illustrated in Fig. 4.7. In the TDES synthesis procedures (introduced in [11]), the synchronous product is computed using the procedure **sync**. \square

Clearly, the execution sequences shown in the TTG depicted in Fig. 4.7 allow the resources to be available to execute tasks τ_1 and τ_2 concurrently. It shows that for each duration of two ticks, a total of one τ_1 and two τ_2 are executed.

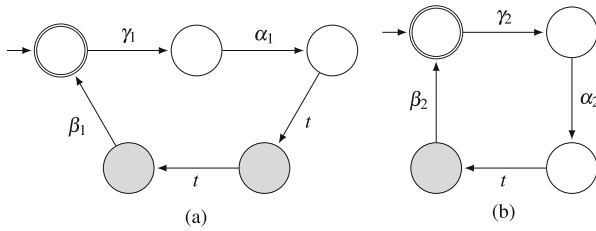


Fig. 4.6 Tasks τ_1 and τ_2 . (a) Task τ_1 . (b) Task τ_2

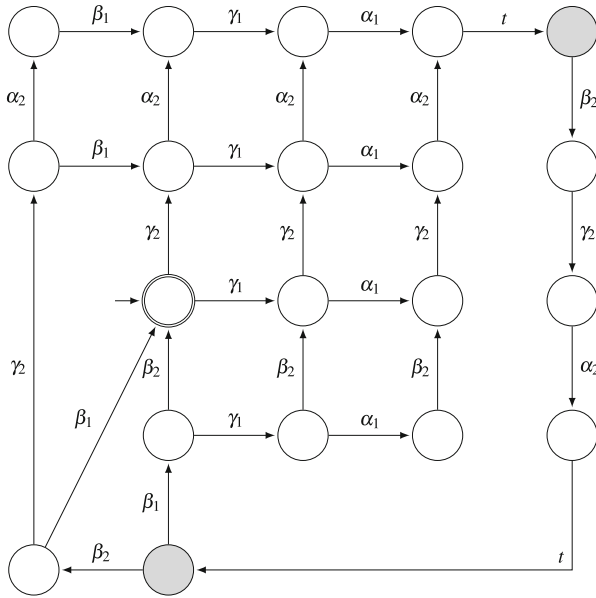


Fig. 4.7 Synchronous product of tasks τ_1 and τ_2

4.2.4 Timed Discrete-Event System Generators

The modelling, scheduling, and reconfiguration of RTS based on supervisory control of TDES, using TTCT, are presented below. The possible behaviors of tasks τ_1 , τ_2 , and τ_3 are represented by \mathbf{G}_1 , \mathbf{G}_2 , and \mathbf{G}_3 , respectively. The parameters of these created tasks are presented in Table 4.1. In the TDES models, events γ_i , α_i , β_i , and tick are represented by $i0$, $i1$, $i2$, and 0, respectively. A task with a superscript “ l ” (resp., “ u ”) represents that it possesses the lower (resp., upper) period bound; the corresponding task name in TTCT is prefixed by an L (resp., U). Taking \mathbf{G}_2^l as an example, the lower and upper bounds of its multi-period are equal to the lower bound of \mathbf{T}_2 . Evidently, we have

Table 4.1 Parameters of RTS tasks

Task	TDES	TTCT	R_i	C_i	D_i	T_i
τ_1	\mathbf{G}_1	TASK1	0	1	4	[5, 5]
τ_2	\mathbf{G}_2	TASK2	0	2	6	[4, 6]
τ_3	\mathbf{G}_3	TASK3	0	2	5	[3, 5]
τ_2^l	\mathbf{G}_2^l	LTASK2	0	2	4	[4, 4]
τ_2^u	\mathbf{G}_2^u	UTASK2	0	2	6	[6, 6]
τ_3^l	\mathbf{G}_3^l	LTASK3	0	2	3	[3, 3]
τ_3^u	\mathbf{G}_3^u	UTASK3	0	2	5	[5, 5]

$$L(\mathbf{G}_2^l) \subseteq L(\mathbf{G}_2), L(\mathbf{G}_2^u) \subseteq L(\mathbf{G}_2), L(\mathbf{G}_3^l) \subseteq L(\mathbf{G}_3), L(\mathbf{G}_3^u) \subseteq L(\mathbf{G}_3);$$

and

$$L_m(\mathbf{G}_2^l) \subseteq L_m(\mathbf{G}_2), L_m(\mathbf{G}_2^u) \subseteq L_m(\mathbf{G}_2), L_m(\mathbf{G}_3^l) \subseteq L_m(\mathbf{G}_3), L_m(\mathbf{G}_3^u) \subseteq L_m(\mathbf{G}_3).$$

The corresponding TTCT MAKEIT.TXT file for all the created tasks are listed below. In TTCT, the **edit** procedure can be utilized to convert a multi-period periodic task to a task with a fixed-period or vice-versa.

TASK1 = **create** (TASK1, [mark 0], [tran [0, 10, 1], [1, 0, 5], [1, 11, 2], [2, 0, 3], [3, 12, 4], [4, 0, 8], [5, 0, 9], [5, 11, 6], [6, 0, 7], [7, 12, 8], [8, 0, 12], [9, 0, 13], [9, 11, 10], [10, 0, 11], [11, 12, 12], [12, 0, 16], [13, 11, 14], [14, 0, 15], [15, 12, 16], [16, 0, 0]], [forcible 10, 11, 12]) (17, 20)

TASK2 = **create** (TASK2, [mark 0, 15, 20], [tran [0, 20, 1], [1, 0, 6], [1, 21, 2], [2, 0, 3], [3, 0, 4], [4, 22, 5], [5, 0, 10], [6, 0, 11], [6, 21, 7], [7, 0, 8], [8, 0, 9], [9, 22, 10], [10, 0, 15], [11, 0, 16], [11, 21, 12], [12, 0, 13], [13, 0, 14], [14, 22, 15], [15, 0, 20], [15, 20, 1], [16, 0, 21], [16, 21, 17], [17, 0, 18], [18, 0, 19], [19, 22, 20], [20, 0, 0], [20, 20, 1], [21, 21, 22], [22, 0, 23], [23, 0, 24], [24, 22, 0]], [forcible 20, 21, 22]) (25, 32)

LTASK2 = **edit** (TASK2, [trans -[11, 0, 16], -[15, 0, 20]]) (25, 29)

LTASK2 = **trim** (LTASK2) (16, 18)

LTASK2 = **minstate** (LTASK2) (15, 17)

UTASK2 = **edit** (TASK2, [mark -[15], -[20]], [trans -[15, 20, 1], -[20, 20, 1]]) (25, 29)

LTASK3 = **create** (LTASK3, [mark 0], [tran [0, 30, 1], [1, 0, 6], [1, 31, 2], [2, 0, 3], [3, 0, 4], [4, 32, 5], [5, 0, 0], [6, 31, 7], [7, 0, 8], [8, 0, 9], [9, 32, 0]], [forcible 30, 31, 32]) (10, 11)

$\text{TASK3} = \mathbf{edit}$ (LTASK3, [mark +[10], +[15]], [trans +[5, 0, 10], +[6, 0, 11], +[9, 32, 10], +[10, 0, 15], +[10, 30, 1], +[11, 0, 16], +[11, 31, 12], +[12, 0, 13], +[13, 0, 14], +[14, 32, 15], +[15, 0, 0], +[15, 30, 1], +[16, 31, 17], +[17, 0, 18], +[18, 0, 19], +[19, 32, 0], -[5, 0, 0], -[9, 32, 0]]) (20, 25)

$\text{UTASK3} = \mathbf{edit}$ (TASK3, [mark -[10], -[15]], [trans -[10, 30, 1], -[15, 30, 1]]) (20, 23)

The composite model of an RTS is generated by the synchronous product of all the processed tasks [5, 11]. Suppose that tasks τ_1 and τ_2 are running in RTS \mathbb{S}_0 . We generate \mathbb{S}_0 by the following TTCT procedures (all the **sync** operations in the original MAKEIT.TXT file are reported with the message “Blocked_events = None”, eliminated in this monograph for readability):

$$\text{SYS0} = \mathbf{sync} (\text{TASK1}, \text{TASK2}) (425, 644)$$

where “(425, 644)” denotes that \mathbb{S}_0 , represented by SYS0, has 425 states and 644 transitions. Suppose that another RTS \mathbb{S}_1 , represented by SYS1, contains τ_1 , τ_2 , and τ_3 . It is generated based on \mathbb{S}_0 as follows.

$$\text{SYS1} = \mathbf{sync} (\text{SYS0}, \text{TASK3}) (8500, 16367)$$

The composite task model of traditional periodic RTS is generated by the technique proposed by Chen and Wonham in [5]. By choosing the periodic tasks with the lower (resp., upper) bound of periods, we generate \mathbb{S}_0^l (LSYS0), \mathbb{S}_1^l (LSYS1), and \mathbb{S}_1^u (USYS1) as follows. They are the counterparts of \mathbb{S}_0 and \mathbb{S}_1 with fixed-periods.

$$\text{LSYS0} = \mathbf{sync} (\text{TASK1}, \text{LTASK2}) (255, 364)$$

$$\text{LSYS1} = \mathbf{sync} (\text{LSYS0}, \text{LTASK3}) (2550, 4475)$$

$$\text{USYS1} = \mathbf{sync} (\text{TASK1}, \text{UTASK2}) (425, 610)$$

$$\text{USYS1} = \mathbf{sync} (\text{USYS1}, \text{UTASK3}) (1750, 3064)$$

Finally, the five generated RTS are listed in Table 4.2. They will be utilized in the supervisory control and evaluation of the closed behavior of the controlled RTS.

Table 4.2 RTS with multi-periods

RTS	TTCT	Tasks
\mathbb{S}_0	SYS0	τ_1, τ_2
\mathbb{S}_1	SYS1	τ_1, τ_2, τ_3
\mathbb{S}_l^0	LSYS0	τ_1, τ_2^l
\mathbb{S}_l^1	LSYS1	$\tau_1, \tau_2^l, \tau_3^l$
\mathbb{S}_u^1	USYS1	$\tau_1, \tau_2^u, \tau_3^u$

4.3 Dynamic Scheduling and Reconfiguration of Multi-Period RTS

The event controllability and the supervisory control of TDES in this chapter follow the principles proposed in [5] and [11]. In this present chapter, instead of utilizing the method proposed in [5] to dynamically modify the specification for the tasks running in the uni-processor, a general specification \mathbf{S} with

$$L(\mathbf{S}) = L(\mathbf{S}_1) || L(\mathbf{S}_2) || \cdots || L(\mathbf{S}_n)$$

is defined with event set $\Sigma = \bigcup_{1 \leq i \leq n} \Sigma_i$, the union of the event sets of all the potential tasks which may be called by the processor. Let $L(\mathbf{S}) = E \subseteq \Sigma^*$. In addition, $L_m(\mathbf{G}) \subseteq \Sigma^*$ is always satisfied. Hence, let $E \subseteq \Sigma^*$ and

$$K = \text{sup}\mathcal{C}(E \cap L_m(\mathbf{G})).$$

If $K \neq \emptyset$, there exists a marking nonblocking supervisory control (MNSC) for \mathbf{G} such that

$$L_m(V/\mathbf{G}) = K.$$

Clearly, K can be found by the procedure **supcon** (introduced in [11]).

In order to utilize SCT to schedule the RTS non-preemptively, the specifications are defined to ensure that after the occurrence of an event α_i , no other event α_j with $j \neq i$ can occur to preempt it. Hence, the TDES model of specification \mathbf{S}_i for task τ_i is illustrated in Fig. 4.8, in which α_j and β_j with $j \neq i$ represent events α and β for any other task, respectively. The symbol $*$ represents the other events in Σ . The specifications for \mathbf{G}_1 , \mathbf{G}_2 , and \mathbf{G}_3 are created by TTCT. Thereafter, by utilizing **sync**, the general specification \mathbf{S} presented in Fig. 4.10 is generated.

According to [11], given any RTS represented by \mathbf{G} , for any specification, selfloops of events appeared in \mathbf{G} but not in the specification must be adjoined to account for all the events that are irrelevant to the specification but probably executed in the plant. For simplification, we only focus on the specification created by procedure **create**; thereafter the selfloops are adjoined by the procedures **allevents** and **sync**. As listed below, **allevents** is utilized to generate a TTG representing Σ^* . As shown in Fig. 4.9, the nonblocking specification for \mathbf{G} is a generator with only one state at which all the events appeared in \mathbf{G} are enabled. The

Fig. 4.8 Task non-preemptive specification

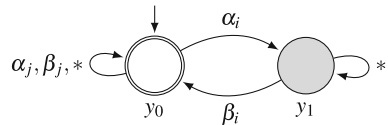


Fig. 4.9 Nonblocking specification for RTS

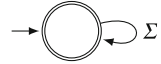
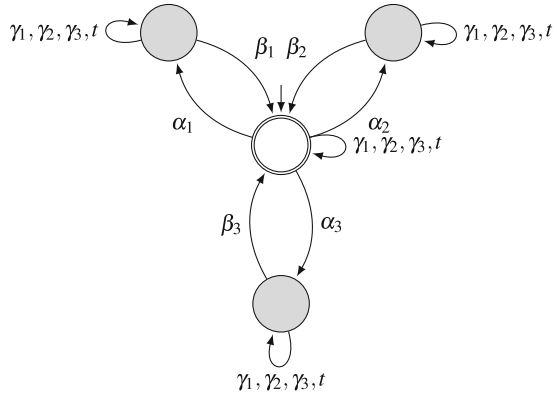


Fig. 4.10 General non-preemptive specification



generated files for the specifications are recorded below, where the SPEC shown in Fig. 4.10 with 4 states and 22 transitions is the general one.

SPEC1 = **create** (SPEC1, [mark 0], [tran [0, 11, 1], [0, 21, 0], [0, 22, 0], [0, 31, 0], [0, 32, 0], [1, 12, 0]], [forcible 11, 12, 21, 22, 31, 32]) (2, 6)

SPEC2 = **create** (SPEC2, [mark 0], [tran [0, 11, 0], [0, 12, 0], [0, 21, 1], [0, 31, 0], [0, 32, 0], [1, 22, 0]], [forcible 11, 12, 21, 22, 31, 32]) (2, 6)

SPEC3 = **create** (SPEC3, [mark 0], [tran [0, 11, 0], [0, 12, 0], [0, 21, 0], [0, 22, 0], [0, 31, 1], [1, 32, 0]], [forcible 11, 12, 21, 22, 31, 32]) (2, 6)

ALLSYS1 = **allevents** (SYS1) (1, 10)

SPEC1 = **sync** (SPEC1, ALLSYS1) (2, 14)

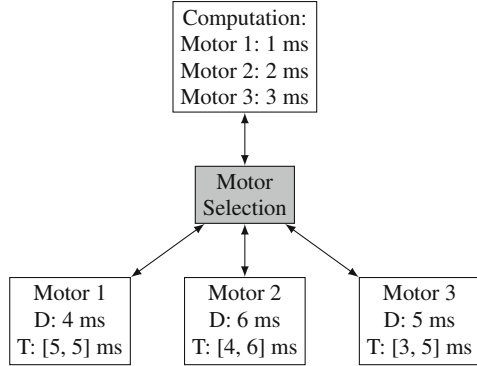
SPEC = **sync** (SPEC1, SPEC2) (3, 18)

SPEC = **sync** (SPEC, SPEC3) (4, 22)

4.4 Case Study: Supervisor Synthesis of Motor Network

As illustrated in Fig. 4.11, the example of a motor network studied in [5] is revised and considered as a reconfigurable RTS. Suppose that three electric motors are controlled by a uni-processor. As depicted in Fig. 4.11, their deadlines and periods are represented by D_i and T_i , respectively. At any time instant, only a subset of these motors is called by the processor. Their parameters coincide with those of the tasks shown in Table 4.1 as

Fig. 4.11 A motor network example



- Motor 1: τ_1 ,
- Motor 2: τ_2 , and
- Motor 3: τ_3 .

Suppose that the motor network has two work plans, coinciding with the defined RTS \mathbb{S}_0 and \mathbb{S}_1 :

Plan 1: uses only Motors 1 and 2, and

Plan 2: uses all three motors.

4.4.1 Real-Time Scheduling

Take \mathbb{S}_0^l (LSYS0) as an example. All the safe execution sequences are calculated by the procedure **supcon**, i.e.,

$$\text{LSUPER0} = \mathbf{supcon}(\text{LSYS0}, \text{SPEC})(153, 190).$$

Since LSUPER0 is not empty, \mathbb{S}_0^l is schedulable at processor utilization

$$U^u = \frac{1}{5} + \frac{2}{4} = 0.7.$$

The safe execution sequence set in LSUPER0 is represented by a TDES with 153 states and 190 transitions. By projecting out all events but α_i , as depicted in Fig. 4.12, which contains 12 states and 15 transitions. We have

$$\text{PJLSUPER0} = \mathbf{project}(\text{LSUPER0}, \text{Image}[11, 21])(12, 15).$$

We obtain the scheduling map illustrated in the *Gantt chart* depicted in Fig. 4.13. PJLSUPER0 provides eight safe execution sequences to schedule the RTS with processor utilization being 0.7:

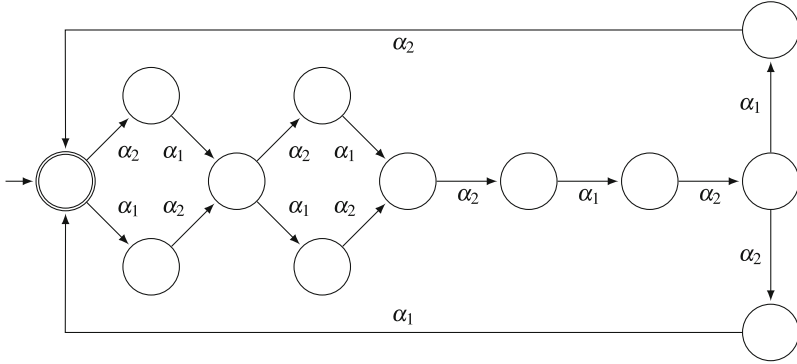


Fig. 4.12 Scheduling map of \mathbb{S}_0^l

1. $\alpha_1\alpha_2\alpha_2\alpha_1\alpha_2\alpha_1\alpha_2\alpha_1\alpha_2$
2. $\alpha_1\alpha_2\alpha_2\alpha_1\alpha_2\alpha_1\alpha_2\alpha_2\alpha_1$
3. $\alpha_1\alpha_2\alpha_1\alpha_2\alpha_2\alpha_1\alpha_2\alpha_1\alpha_2$
4. $\alpha_1\alpha_2\alpha_1\alpha_2\alpha_2\alpha_1\alpha_2\alpha_2\alpha_1$
5. $\alpha_2\alpha_1\alpha_2\alpha_1\alpha_2\alpha_1\alpha_2\alpha_1\alpha_2$
6. $\alpha_2\alpha_1\alpha_2\alpha_1\alpha_2\alpha_1\alpha_2\alpha_2\alpha_1$
7. $\alpha_2\alpha_1\alpha_1\alpha_2\alpha_2\alpha_1\alpha_2\alpha_1\alpha_2$
8. $\alpha_2\alpha_1\alpha_1\alpha_2\alpha_2\alpha_1\alpha_2\alpha_2\alpha_1$

For comparison, the earliest deadline first (EDF) scheduling [6] result of \mathbb{S}_0^l by Cheddar ² [9] is displayed in Fig. 4.13, which coincides with Sequence (1.) above within PJLSUPER0. Sequence (8.), depicted in Fig. 4.14, can never be generated by EDF. By comparing the two sequences in Figs. 4.13 and 4.14, if τ_2^l (with the earliest deadline) cannot arrive on time at $t = 4$, then according to the multiple sequences users can choose another available sequence shown in Fig. 4.13 to schedule task τ_1 first. Thus, recalculating the scheduling sequences is unnecessary. However, there is no EDF sequence to schedule task τ_1 first. If τ_2 cannot arrive on time, the EDF scheduling cannot schedule \mathbb{S}_0^l successfully. The supervisory control technique provides a greater number of safe execution sequences as compared with EDF scheduling. Intuitively, thanks to $L(\mathbf{G}_2^l) \subset L(\mathbf{G}_2)$ and $L_m(\mathbf{G}_2^l) \subset L_m(\mathbf{G}_2)$, the safe execution sequences in \mathbb{S}_0^l should be a proper subset of the safe execution sequences of \mathbb{S}_0 . This is proved as follows.

By calling procedure **supcon**, all the safe execution sequences of the multi-period version RTS \mathbb{S}_0 are obtained. By using the procedure **complement**, we obtain the set of the behaviors prohibited by SUPER0, which is contained in CSUPER0. By computing the **meet** of CSUPER0 and LSUPER0, if the **trim** version

² <http://beru.univ-brest.fr/cheddar/>.

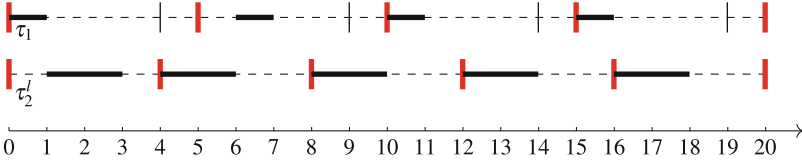


Fig. 4.13 EDF scheduling map of \mathbb{S}_0^l

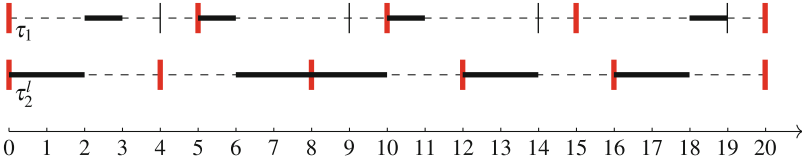


Fig. 4.14 Scheduling of Sequence (8.)

of **meet** is empty, this represents that the reachable and coreachable sequences within LSUPER0 are not in CSUPER0. Hence, LSUPER0 is a proper subset of SUPER0. The corresponding TTCT operations are listed below.

$$\text{SUPER0} = \text{supcon} (\text{SYS0}, \text{SPEC}) (263, 362)$$

$$\text{CSUPER0} = \text{complement} (\text{SUPER0}, []) (264, 1848)$$

$$\text{TEST} = \text{meet} (\text{CSUPER0}, \text{LSUPER0}) (156, 195)$$

$$\text{TEST} = \text{trim} (\text{TEST}) (0, 0)$$

The scheduling map for \mathbb{S}_0 is more complex than that for \mathbb{S}_0^l , which has 50 states and 86 transitions, i.e.,

$$\text{PJSUPER0} = \text{project} (\text{SUPER0}, \text{Image} [11, 21]) (50, 86).$$

Evidently, even though the supervisor for \mathbb{S}_0^l excludes some safe execution sequences of \mathbb{S}_0 , the scheduling map still provides more choices than the EDF scheduling algorithm.

4.4.2 Dynamic Reconfiguration

The set of safe execution sequences of \mathbb{S}_1^l (LSYS1) found by the procedure **supcon** is empty, i.e.,

$$\text{LSUPER1} = \text{supcon} (\text{LSYS1}, \text{SPEC}) (0, 0).$$

Clearly, \mathbb{S}_1^l is non-schedulable at processor utilization

$$U^u = \frac{1}{5} + \frac{2}{4} + \frac{2}{3} > 1.$$

Thus, we need to reconfigure the system to be the multi-period model \mathbb{S}_1 (SYS1) and utilize SCT again to find the safe execution sequences by

$$\text{SUPER1} = \mathbf{supcon} (\text{SYS1}, \text{SPEC}) (2180, 3681).$$

This represents that **supcon** finds all the possible safe execution sequences between the processor utilization

$$U^l = \frac{1}{5} + \frac{2}{6} + \frac{2}{5} < 1$$

and the full processor utilization 1.

The system is finally schedulable since SUPER1 is nonempty. In order to find the *scheduling map* after the reconfiguration, we need to call the procedure **project**. However, since the reconfigured periods provide more than one reconfiguration scenarios, TTCT fails to output the result of projecting SUPER1 onto events α_i , which shows that the dynamic reconfiguration of the periods (event γ_i) violates the observer property discussed in [11]. However, we choose the following method to view a part of the scheduling map of the reconfigured RTS \mathbb{S}_1 :

Step 1

We choose \mathbb{S}_1^u as a subset of the composite task model of \mathbb{S}_1 , based on which we find the safe execution sequence set, which contains 417 states and 574 transitions. The scheduling map is calculated by projecting the safe execution sequences onto events α_1 , α_2 , and α_3 ; it contains 37 states and 54 transitions, as seen in Fig. 4.15. The corresponding TTCT operations are given as follows.

$$\text{USUPER1} = \mathbf{supcon} (\text{USYS1}, \text{SPEC}) (417, 574)$$

$$\text{PJUSUPER1} = \mathbf{project} (\text{USUPER1}, \text{Image} [11, 21, 31]) (37, 54)$$

Step 2

We can verify that \mathbb{S}_1^u is a proper subset of \mathbb{S}_1 via the following TTCT procedures:

$$\text{CSUPER1} = \mathbf{complement} (\text{SUPER1}, []) (2181, 21,810)$$

$$\text{TEST} = \mathbf{meet} (\text{CSUPER1}, \text{USUPER1}) (417, 574)$$

$$\text{TEST} = \mathbf{trim} (\text{TEST}) (0, 0)$$

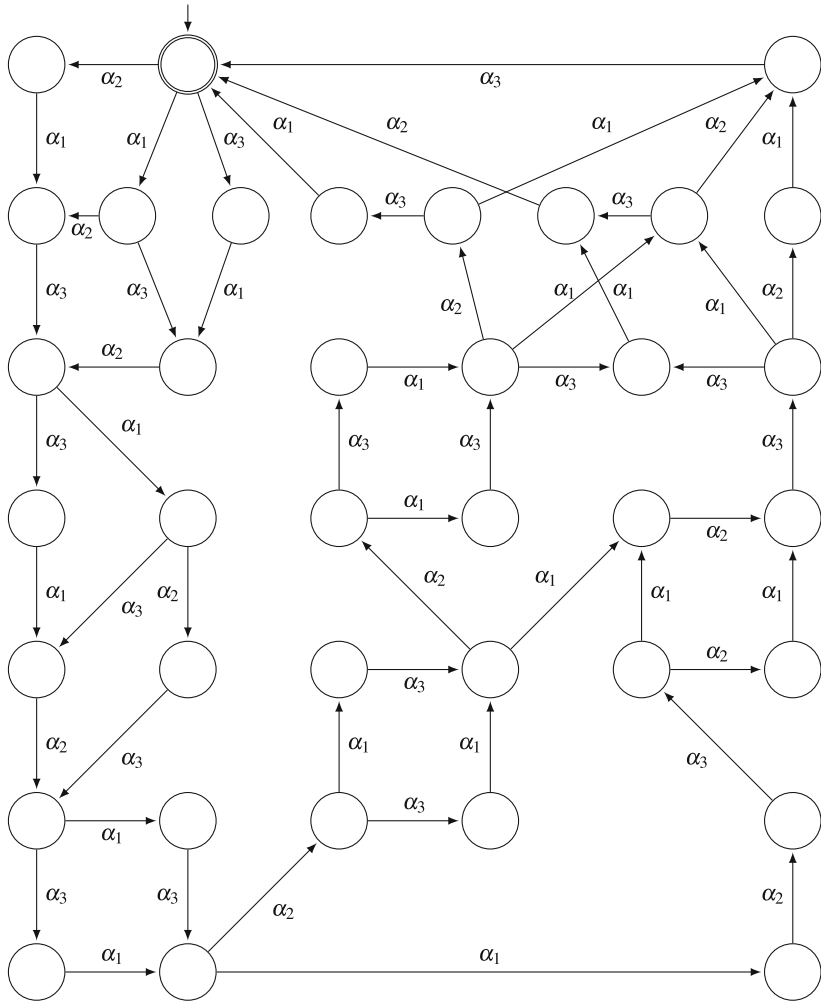


Fig. 4.15 Scheduling map of \mathbb{S}_1^u

Finally, we claim that, after the reconfiguration, the scheduling map of \mathbb{S}_1 is at least as complex as that presented in Fig. 4.15. More precisely, SUPER1 (resp., USUPER1) contains 2180 (resp., 417) states and 3681 (resp., 574) transitions. Intuitively, the scheduling map of SUPER1 should be more complex than that depicted in Fig. 4.15, in which the periods are dynamically reconfigured. The EDF scheduling of \mathbb{S}_1^u by Cheddar is illustrated in the Gantt chart depicted in Fig. 4.16. It can find only one schedulable sequence. Moreover, no sequence for the multi-period RTS can be found by EDF scheduling in Cheddar.

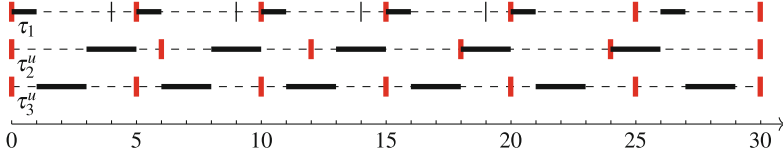


Fig. 4.16 Scheduling map of \mathbb{S}_1^u in Cheddar

4.4.3 Multi-Periods in the Safe Execution Sequences

In Sect. 4.4.1, before the reconfiguration, every scheduling sequence is based on the *fixed periods* of the real-time tasks, with the scheduling stated in LSUPER0. The processor utilization of a task is fixed permanently. For example, we randomly choose a sequence

$$\gamma_1 \alpha_1 \gamma_2 t \beta_2 t \alpha_2 t t \beta_2 \gamma_2 \dots$$

By projecting out events γ_1 , α_1 , and β_1 , we obtain

$$\gamma_2 t \beta_2 t \alpha_2 t t \beta_2 \gamma_2 \dots$$

We note that a period of task τ_2 equals four time units and its processor utilization is equal to $U_2 = \frac{1}{2}$.

In SUPER0 (the reconfigured RTS with multi-periods), we randomly choose two sequences as follows:

1. $\gamma_1 \alpha_1 \gamma_2 t \beta_1 \alpha_2 t t \beta_2 t t \gamma_2 \dots$
2. $\gamma_1 \alpha_1 \gamma_2 t \beta_1 t t \alpha_2 t \gamma_1 t \beta_2 \gamma_2 \alpha_2 t t \beta_2 \alpha_1 t \beta_1 t \gamma_2 \dots$

By projecting out γ_1 , α_1 , and β_1 in Sequence (1.), we obtain

$$\gamma_2 t \alpha_2 t t \beta_2 t t \gamma_2 \dots$$

Obviously, a period of task τ_2 equals five time units. Then the processor utilization of τ_2 is $\frac{2}{5}$.

By projecting out γ_1 , α_1 , and β_1 in Sequence (2.), we obtain

$$\gamma_2 t t t \alpha_2 t t \beta_2 \gamma_2 \alpha_2 t t \beta_2 t t \gamma_2 \dots$$

Evidently, in two adjacent periods of task τ_2 , its periods are five and four time units, respectively. Hence, in the second period of the execution of τ_2 , its processor utilization is changed from $\frac{2}{5}$ to $\frac{1}{2}$ to speed up the scheduling process. This means that, according to the processor utilization interval predefined by the users, the processor utilization of the RTS is dynamically changed at run-time.

By comparing Sequences (1.) and (2.), we see that after the occurrence of substring $\gamma_1\alpha_1\gamma_2t\beta_1$, the controller provides at least two subsequences in Sequences (1.) and (2.) to schedule τ_2 . However, neither the non-preemptive scheduling proposed in [5] nor the EDF scheduling can provide such scheduling plans.

4.5 Conclusion

As summarized in this chapter, based on supervisory control of TDES, Chen and Wonham propose a formal constructive method in [5], for the purpose of scheduling the non-preemptive execution of a set of periodic tasks on a processor, which could be either resource-sharing or independent. Thereafter, in order to address the reconfiguration problem, the proposed model is generalized in [10]. This model can be used to schedule or reconfigure RTS. As a consequence, the formal SCT of TDES can be considered as a rigorous analysis and synthesis tool to dynamically schedule and reconfigure the non-preemptive scheduling of RTS. Suppose that in every scheduling plan only a subset of tasks of an RTS is called by the processor. Instead of dynamically updating the specification for the tasks running in the uni-processor, a general specification is presented, which guarantees that all the potential tasks called by the processor can be scheduled non-preemptively. In the case of the RTS claimed by [5] to be non-schedulable, the presented two-step dynamic reconfiguration approach can be utilized to find all the safe execution sequences (possible reconfiguration scenarios) of each task in the RTS. These sequences provide more choices than the EDF scheduling algorithm. The processor and the real-time tasks are general models for real-world RTS. The multi-period model can be utilized to describe the behavior of a manual assembly process or a robotic pick-and-place operation executed by a processor that could be a water vessel system, computer numerical control machine, a robot, or an assembly-line worker. This leads to the possibility that the offline reconfiguration method can be implemented in practical contexts based on reconfigurable real-time scheduling. Building on the multi-period framework, and for the purpose of solving the problem of the TDES model faced, the presented RTS modelling tool is generalized in Chaps. 5 and 6 as modular discrete-event system (DES) models (based on the nonblocking supervisory control of DES) and Chap. 7 as a hierarchical DES model (based on the nonblocking supervisory control of state-tree structures).

References

1. Brandin, B.A., Wonham, W.M.: Supervisory control of timed discrete-event systems. *IEEE Trans. Autom. Control* **39**(2), 329–342 (1994)
2. Buttazzo, G., Abeni, L.: Adaptive workload management through elastic scheduling. *Real-Time Syst.* **23**, 7–24 (2002)

3. Buttazzo, G.C., Lipari, G., Abeni, L.: Elastic task model for adaptive rate control. In: IEEE Real-Time Systems Symposium, pp. 286–295 (1998)
4. Buttazzo, G.C., Lipari, G., Caccamo, M., Abeni, L.: Elastic scheduling for flexible workload management. *IEEE Trans. Comput.* **51**(3), 289–302 (2002)
5. Chen, P.C.Y., Wonham, W.M.: Real-time supervisory control of a processor for non-preemptive execution of periodic tasks. *Real-Time Syst.* **23**, 183–208 (2002)
6. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM* **20**(1), 46–61 (1973)
7. Marinoni, M., Buttazzo, G.: Elastic DVS management in processors with discrete voltage/frequency modes. *IEEE Trans. Ind. Inf.* **3**(1), 51–62 (2007)
8. Sha, L., Abdelzaher, T., Cervin, A., Baker, T., Burns, A., Buttazzo, G., Caccamo, M., Lehoczky, J., Mok, A.K.: Real time scheduling theory: a historical perspective. *Real-Time Syst.* **28**(2), 101–155 (2004)
9. Singhoff, F., Legrand, J., Nana, L., Marcé, L.: Cheddar: a flexible real time scheduling framework. *ACM SIGAda Ada Lett.* **4**, 1–8 (2004)
10. Wang, X., Li, Z., Wonham, W.M.: Dynamic multiple-period reconfiguration of real-time scheduling based on timed DES supervisory control. *IEEE Trans. Indust. Inf.* **12**(1), 101–111 (2016)
11. Wonham, W.M., Cai, K.: *Supervisory Control of Discrete-Event Systems*. Monograph Series Communications and Control Engineering. Springer, Berlin (2018)

Chapter 5

Priority-Free Conditionally-Preemptive Real-Time Scheduling Based on R-W Method



5.1 Introduction

Real-time systems (RTS) can be loosely defined as systems whose *response time* is an important determinant of *correct functioning* [8]. Historically, RTS were constructed in an *ad hoc* manner, and scheduled by *cyclic executives* [16]. An RTS consists of a number of tasks with explicit timing requirements, which could be a water vessel system, a computer numerical control machine, a robot, or an assembly-line worker. In the literature, most of the existing classical real-time scheduling algorithms are based on dynamic or fixed priorities [1, 4–7, 9–12, 15, 16, 21]. The study in [5] shows that, when the tasks' periods are equal to their deadlines, the *preemptive earliest deadline first (EDF) scheduling* algorithm is optimal.

As stated in Chap. 4, in the seminal work [3], Chen and Wonham propose a timed discrete-event system (TDES) modelling mechanism to model *real-time tasks* running in uni-processor RTS and schedule them non-preemptively. Later the TDES model is generalized in [18] for the purpose of dynamic reconfigurations of RTS when they are non-schedulable.

In this chapter, a discrete-event system (DES)-based real-time task model is presented to schedule the real-time tasks running in uni-processor RTS. The timing constraints of RTS tasks are represented by different events in the DES model. Hence, in the DES modelling mechanism, the execution of different tasks will always happen sequentially, which is more realistic. Clearly, a DES is more general for modelling RTS than TDES, which provides the possibility of preemptive SCT-based scheduling of RTS.

Compared with non-preemptive scheduling, preemptability can provide more flexibility to real-time scheduling. In fully preemptive systems, at any time, the execution of a running task can be interrupted by tasks with higher priorities, and it continues when all tasks with higher priorities have been completed [2]. However, in some special cases, both preemptive and non-preemptive scheduling policies are conservative. Users may customize specific preemption plans that are neither

preemptive nor non-preemptive. In fact, for such real-time scheduling requirements, priorities cannot be assigned to real-time tasks.

The behavior of periodic RTS tasks is described by *formal language* [20]. Each language can be represented by a DES *generator*. The *synchronous product* [20] of these DES generators can integrate the models of the tasks running in a processor into a complex generator to represent the global processor behavior.

This chapter presents *priority-free conditionally-preemptive (PFCP) scheduling*, which generalizes priority-based preemption. By defining the preemption relation among any two tasks running in a processor, a preemption matrix can be utilized to describe all the possible *fixed-priority* (FP) preemption relations and other user-specified preemption relations. Based on this matrix, the corresponding DES specifications are designed accordingly. From the perspective of a task's execution, between any two adjacent *processor time units*, the task-centered conditional-preemption relations are also depicted by DES specifications. Clearly, the two presented general conditional-preemption specifications are utilized to customize scheduling and preemption requirements conditionally. The *worst-case response time* (WCRT, no later than the corresponding deadline) of a real-time task is used to evaluate the schedulability of the tasks processed in an RTS [17]. In this chapter, WCRT can also be restricted by a specification. Furthermore, the nonblocking preemptive scheduling of real-time tasks is also addressed, which provides all the dynamic priority scheduling sequences.

Similar to Chap. 4, all the safe execution sequences generated by the synchronized specifications with respect to the tasks running in an RTS can be synthesized offline by SCT. Users can choose any sequence to schedule the processor. The real-time scheduling with conditional-preemption is applied to the real-world uni-processor systems.

In comparison with the TDES-based RTS scheduling frameworks proposed in [3, 18], the DES modelling framework and the PFCP scheduling specifications presented in this chapter are more realistic:

- The task behavior is modelled by DES, and thus the execution of each real-time task's processing is represented by an individual event instead of the global *tick* event t in TDES.
- The priorities of tasks are not treated and the preemption relations are described by a matrix.
- The specifications are imposed from the perspective of both the processor and individual task. Based on this scheduling principle, some classic real-time scheduling policies such as FP or *partially FP scheduling* [1, 4, 7, 9, 12, 15, 16] are treated as special cases of the PFCP specifications.

The rest of this chapter is organized as follows. The system model and priority-free real-time scheduling with conditional-preemption principles are described in Sect. 5.2. The DES model for the periodic tasks and the RTS are proposed in Sect. 5.3. The specifications are formalized and established in Sect. 5.4. Sections 5.5 and 5.6 report methodologies for the real-time scheduling with conditional-preemption applied to real-world systems. Finally, Sect. 5.7 reaches conclusions.

5.2 Task Models and Preemption Policies

In this section, the behavior of a *periodic real-time task* is represented by a DES diagram. Three types of specifications are provided. The presented PFCP real-time scheduling algorithm generalizes priority-based preemption scheduling. In addition, from the perspective of each individual task, *task-centered conditional-preemption specification* is presented, which allows the task execution to be preempted by a subset of other tasks between any two adjacent processor time units. Furthermore, the WCRT of a task no later than its deadline can also be restricted by a specification.

5.2.1 Task Model

Consider the regular periodic tasks with regular periods, i.e., the time intervals between any two adjacent arrivals are constant. Suppose that a periodic RTS \mathbb{S} processes n tasks, i.e., $\mathbb{S} = \{\tau_1, \tau_2, \dots, \tau_i, \dots, \tau_n\}$, $i \in \mathbf{n} = \{1, 2, \dots, n\}$. A periodic task is specified as a four-tuple

$$\tau_i = (R_i, C_i, D_i, T_i)$$

with

- a *release time* R_i ,
- a *worst-case execution time (WCET)* C_i ,
- a *deadline* D_i , and
- a *regular period* T_i .

As stated in Sect. 3.1, a periodic task τ_i consists of an infinite sequence of *jobs* repeated periodically that are represented by a corresponding four-tuple

$$J_{i,j} = (r_{i,j}, C_i, d_{i,j}, p_{i,j}).$$

The subscript “ i, j ” of $J_{i,j}$ represents the j -th execution of task τ_i .

Motivating Example

Suppose that a uni-processor RTS \mathbb{S} executes four periodic tasks τ_1 , τ_2 , τ_3 , and τ_4 . Their parameters are shown in Table 5.1. We assume that the execution of τ_1 can be preempted only by τ_2 , τ_2 only by τ_4 , and τ_4 only by τ_1 ; moreover, τ_3 cannot be preempted. Clearly, no priorities can be assigned to these tasks. In this case, the EDF and FP algorithms cannot be utilized to schedule this RTS. In order to solve such problems, we discard the priorities and consider the real-time scheduling as priority-free. \square

Table 5.1 Parameters of four tasks

Task	R_i	C_i	D_i	T_i
τ_1	0	3	9	9
τ_2	3	3	6	6
τ_3	0	1	4	5
τ_4	0	2	14	18

5.2.2 Priority-Free Real-Time Scheduling

For some real-world preemption policies, the tasks running in a processor cannot be assigned with priorities [19]. Hence, for both preemptive and non-preemptive scheduling, from the perspective of the processor, conditional-preemption among real-time tasks is presented. By a preemption matrix that will be defined later, users can define any preemption relation among all the tasks running in the same processor.

Definition 5.1 [priority-free] A scheduling policy is said to be *priority-free* if all the released tasks in a processor can be processed in any order. \diamond

As discussed in Sect. 1.4.1, in each hyper-period, all the processor time units are partitioned into:

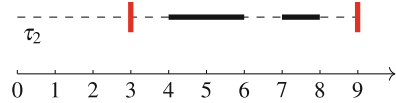
- *busy time*: the processor is occupied by other tasks, and thus τ_i cannot be executed,
- *running time*: τ_i is in process,
- *preemption time* (if any): after the execution of τ_i has started, its execution is interrupted by (a subset of) other tasks, and
- *free time*: the execution of τ_i is completed or τ_i has not arrived yet. These processor time units can be idle or utilized to execute other tasks.

A priority-free scheduling policy can be utilized to schedule all the periodic tasks randomly, i.e., for a real-time task, its busy time and preemption time can be occupied by any other tasks. Moreover, in accordance with traditional real-time scheduling, a task is not allowed to be interrupted if the system would thereby come to an idle operation. In this case, the free time is also allowed to be in an idle operation only when no task is in process.

Example For task τ_2 shown in Table 5.1, a possible conditionally-preemptive real-time scheduling is illustrated in the *Gantt chart* depicted in Fig. 5.1. By allowing preemption of other tasks, the first nine processor time units are partitioned into:

- busy time: time interval [3, 4),
- running time: time intervals [4, 6) and [7, 8),
- preemption time: time interval [6, 7), and
- free time: time intervals [0, 3) and [8, 9).

Fig. 5.1 Real-time scheduling of task τ_2



The time intervals $[0, 4)$, $[6, 7)$, and $[8, 9)$ can be occupied by other tasks running in the same processor. In addition, only time intervals $[0, 3]$ and $[8, 9]$ could alternatively be idle. \square

From the perspective of processor and individual tasks, two sets of general scheduling policies are presented, respectively, which can be applied to any specific conditional-preemption plan.

5.2.3 Preemption Matrices

From the perspective of a processor, its preemption matrix is defined to describe the preemption relations among the tasks running in it.

Definition 5.2 [preemption matrix] An $n \times n$ matrix \mathbf{A} is said to be a *preemption matrix* if $\mathbf{A}_{i,j} = 1$ (resp., $\mathbf{A}_{i,j} = 0$) represents that a task τ_i is allowed (resp., not allowed) to be preempted by task τ_j . \diamond

The *preemption matrix* \mathbf{A} of an RTS \mathbb{S} is in the form

$$\mathbf{A} = \begin{pmatrix} 0 & * & \cdots & * \\ * & 0 & \cdots & * \\ \vdots & \vdots & \ddots & \vdots \\ * & * & \cdots & 0 \end{pmatrix} \tag{5.1}$$

where $*$, either 0 or 1, can be predefined by users.

Definition 5.3 [matrix-based PFCP] A *preemption policy* is said to be *matrix-based PFCP* if it can be represented by a preemption matrix. \diamond

For a uni-processor, according to the preemption matrix, a running task τ_i can be interrupted by the execution of a specified task set \mathbf{A}_{τ_i} that is a subset of the other tasks processed in \mathbb{S} , i.e.,

$$\mathbf{A}_{\tau_i} = \{\tau_j \in \mathbb{S} \mid \text{the execution of } \tau_i \text{ is allowed to be preempted by } \tau_j\}.$$

Example Consider the real-time tasks shown in Table 5.1. Suppose that every task can be preempted by other tasks, i.e., the tasks are preemptive. All the variable entries in its corresponding preemption matrix \mathbf{A} are replaced by 1, as shown in \mathbf{A}_1 .

In contrast, suppose that the tasks are non-preemptive, i.e., no task can be preempted by other tasks. Thus, all the *'s in its corresponding preemption matrix \mathbf{A} are replaced by 0, as shown in \mathbf{A}_2 .

Moreover, in accordance with the FP real-time scheduling by assigning priorities to tasks τ_1 , τ_2 , τ_3 , and τ_4 in an *ascending order*, preemption matrix \mathbf{A}_3 is customized, which shows that

- τ_1 can be preempted by τ_2 , τ_3 , and τ_4 ,
- τ_2 can be preempted by τ_3 and τ_4 , and
- τ_3 can be preempted by τ_4 .

Similarly, we can assign priorities to tasks τ_1 , τ_2 , τ_3 , and τ_4 in a *descending order*, as shown in \mathbf{A}_4 . By assigning partial preemption relation to real-time tasks conditionally, \mathbf{A}_5 is customized. Moreover, the preemption matrix for the motivating example is stated in \mathbf{A}_6 . \square

$$\mathbf{A}_1 = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}, \mathbf{A}_2 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

$$\mathbf{A}_3 = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \mathbf{A}_4 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix},$$

$$\mathbf{A}_5 = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \mathbf{A}_6 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

5.2.4 Task-Centered Conditional-Preemption Constraints

Focusing on any task's execution, a general preemption policy, namely task-centered conditional-preemption specification, is presented. It can be utilized to designate how long after its execution is started, a task can be preempted by other tasks. In each processing period, from the perspective of an individual task, between any two adjacent processor time units, the execution of τ_i can be preempted by a user-defined set of other tasks running in the same processor.

Example According to \mathbf{A}_1 in the previous example, τ_2 can be preempted by τ_1 , τ_3 , and τ_4 , which represents that the execution of τ_2 can be interrupted by τ_1 , τ_3 , or τ_4 immediately upon their arrival. As shown in Table 5.1, we have $C_2 = 3$, i.e., the

execution of τ_2 takes three time units. Thus we can define two different task-centered conditional-preemption plans for the execution of τ_2 , e.g., between the first two time units, only τ_1 and τ_3 can interrupt the execution of τ_2 . \square

5.2.5 Response Time Constraints

The WCRT of a real-time task is a procedure provided by RTS scheduler Cheddar¹ [17] to evaluate the schedulability of the tasks processed in an RTS. From the perspective of supervisory control, WCRT is restricted by a specification.

Example For task τ_2 , we have $D_2 = 9$. A WCRT specification can be customized to restrict the execution of τ_2 to be completed no later than 7 time units. Formally, we have $\mathbb{W}_2 = 7$. \square

Then, all the safe execution sequences generated by the synchronized specifications with respect to the tasks running in a uni-processor RTS can be calculated offline by the supervisory control of DES. Users can choose any sequence to schedule the processor. In the case that task-centered conditional-preemption relations do not exist, the preemption relations defined in the matrix are applied to the real-time scheduling throughout the execution. Otherwise, the real-time scheduling should take both the task-centered conditional-preemption and PFCP specifications into account simultaneously.

5.3 Tasks Modelled by Discrete-Event Systems

In this section, DES generators are utilized to describe the *processor behavior* to execute periodic real-time tasks. As stated in Sect. 4.2.2, the states filled with gray represent that the corresponding task is under execution. Eventually, the synchronized language, represented by a more complex DES generator, is utilized to describe the global processor behavior for real-time scheduling. In a synchronized DES *generator*, all the enabled events can occur without considering their priorities. Thus, if two or more events are eligible simultaneously, their synchronous product allows them to occur in any order. Since the synchronous product can provide all the possible sequences that are not related to priorities, conditional-preemption is possible. Each DES generator can be represented by a regular language, which is stated in the Appendix of this chapter.

The DES generator for an RTS task τ_i is represented by

$$\mathbf{G}_i = (Q_i, \Sigma_i, \delta_i, q_{0,i}, Q_{m,i}),$$

¹ <http://beru.univ-brest.fr/cheddar/>.

where

- Q_i is the finite *state set*,
- Σ_i is the *alphabet* with $\Sigma_i = \Sigma_{con,i} \dot{\cup} \Sigma_{unc,i}$:
 - $\Sigma_{con,i} := \{\alpha_i, c_1, c_2, \dots, c_i, \dots, c_n\}$: *controllable event subset*, and
 - $\Sigma_{unc,i} := \{\beta_i, \gamma_i, l\}$: *uncontrollable event subset*,
- $\delta_i : Q_i \times \Sigma_i \rightarrow Q_i$ is the (*partial*) *transition function*,
- $q_{0,i}$ is the *initial state*, and
- $Q_{m,i}$ is the subset of *marker states*.

Furthermore, for \mathbf{G}_i , its *alphabet (event labels)*, written as Σ_i to describe the behavior of \mathbf{G}_i , is the disjoint union of $\Sigma_{o,i}$ and Σ_e , i.e., $\Sigma_i = \Sigma_{o,i} \dot{\cup} \Sigma_e$ with $\Sigma_{o,i} \cap \Sigma_e = \emptyset$, $\Sigma_{o,i} = \{\gamma_i, \alpha_i, \beta_i\}$, and $\Sigma_e = \{c_1, c_2, \dots, c_i, \dots, c_n, l\}$, where

- $\Sigma_{o,i}$ is the operation event set of task τ_i , with
 - γ_i : task τ_i is released,
 - α_i : the execution of τ_i is started, and
 - β_i : the execution of τ_i is completed,
- Σ_e is the execution event set, with
 - ($i \in \mathbf{n}$) c_i : task τ_i is under execution in the processor, and
 - l : no task is under execution in the processor, i.e., the corresponding processor time unit is in an idle operation.

The global event set of an RTS \mathbb{S} is denoted by

$$\Sigma = \Sigma_1 \cup \Sigma_2 \cup \dots \cup \Sigma_n.$$

Clearly, Σ is partitioned into

- $\Sigma_{con} = \{\alpha_i, c_i | i \in \mathbf{n}\}$: the *controllable event set*, and
- $\Sigma_{unc} = \{\beta_i, \gamma_i, l | i \in \mathbf{n}\}$: the *uncontrollable event set*;

and also partitioned into

- $\Sigma_o = \{\gamma_i, \alpha_i, \beta_i | i \in \mathbf{n}\}$: the *operation event set*, and
- $\Sigma_e = \{c_i, l | i \in \mathbf{n}\}$: the *execution event set*.

The controllability of event α_i (resp., c_i), $i \in \mathbf{n}$, endows the uni-processor RTS with the authority to choose and execute (resp., interrupt) any task among all the released ones. The controllable event α_i is disabled in order to delay the execution of task τ_i for the purpose of avoiding blocking. The general DES model for real-time periodic tasks is presented in Fig. 5.2, in which * represents the events in $\Sigma_e - \{c_i\}$. All the states and transitions are defined by:

- states $y_{-R_i}, y_{-R_i+1}, \dots, y_{-1}$, and y^y form the state set before task τ_i is released for the first time,
- state y^y : τ_i is ready for execution,

- state y_k^α with $0 \leq k \leq D_i - C_i$: k time units have passed since the recent release of task τ_i ,
- state $y_{k,p}$ with $0 \leq k \leq D_i - C_i$ and $0 \leq p \leq C_i$: starting the p -th time unit execution of task τ_i at the k -th time unit in a period,
- state y_k^β with $C_i \leq k \leq D_i$: the targeting state of event β_i ,
- states y_k with $D_i < k < T_i$: the execution of task τ_i is completed and k time units have elapsed since the recent release of task τ_i ,
- $D_i = T_i \Rightarrow y_{D_i}^\beta = y^\gamma$,
- the state with an *entering arrow* is the initial state y_{-R_i} ,
- the states represented by *double circles* are the marker states $Q_{m,i}$,
- c_i (resp., c_j) represents the execution of τ_i (resp., $\tau_j \in \mathbb{S}, j \neq i$), and
- the function δ_i satisfies
 - $(-R_i \leq k \leq -1) \delta_i(y_k, l) = y_{k+1}$: the processor is in an idle operation,
 - $(-R_i \leq k \leq -1) \delta_i(y_k, c_j) = y_{k+1}$: task τ_j is in process,
 - $\delta(y^\gamma, \gamma_i) = y_0^\alpha$: τ_i is released,
 - $(0 \leq k \leq D_i - C_i) \delta_i(y_k^\alpha, \alpha_i) = y_{k,0}$: at state y_k^α , the execution of τ_i is started,
 - $(C_i \leq k \leq D_i) \delta_i(y_{k,C_i}, \beta_i) = y_k^\beta$: at state y_{k,C_i} , the *processing* of τ_i is completed,
 - $(0 \leq k < D_i, 0 \leq p < C_i) \delta_i(y_{k,p}, c_i) = y_{k+1,p+1}$: task τ_i is in process,
 - $(0 \leq k < D_i, 0 \leq p < C_i) \delta_i(y_{k,q}, c_j) = y_{k+1,q}$: the execution of task τ_i is preempted by τ_j ,
 - $(C_i \leq k < D_i), \delta_i(y_k^\beta, c_j) = y_k^\beta$: after the occurrence of β_i , τ_j is in process,
 - $(C_i \leq k < D_i), \delta_i(y_k^\beta, l) = y_k^\beta$: after the occurrence of β_i , the processor is in an idle operation,
 - $\delta_i(y_{D_i}^\beta, c_j) = y_{D_i+1}$: after the occurrence of β_i , τ_j is in process,
 - $\delta_i(y_{D_i}^\beta, l) = y_{D_i+1}$: after the occurrence of β_i , the processor is in an idle operation,
 - $(D_i < k \leq T_i - 1), \delta_i(y_k, c_j) = y_{k+1}$: task τ_j is in process, and
 - $(D_i < k \leq T_i - 1), \delta_i(y_k, l) = y_{k+1}$: the processor is in an idle operation.

Example The DES model \mathbf{G}_2 corresponding to tasks τ_2 is depicted in Fig. 5.3. Since $R_2 = 3$, event γ_2 occurs at the end of the third processor time unit. After a period of τ_2 is finished, γ_2 occurs immediately to repeat the process. \square

All the operations in TCT² and the names of the generated files are recorded in an annotated file MAKEIT.TXT. In TCT, $\mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_3$, and \mathbf{G}_4 are named as TASK1, TASK2, TASK3, and TASK4, respectively. The creation of TASK2 is reported below, in which for $i \in \mathbf{n}$, events $\gamma_i, \alpha_i, \beta_i$, and c_i are renamed $i0, i1, i2$, and $i9$, respectively. Moreover, event l is represented by 0. The corresponding system

² <http://www.control.utoronto.ca/DES>.

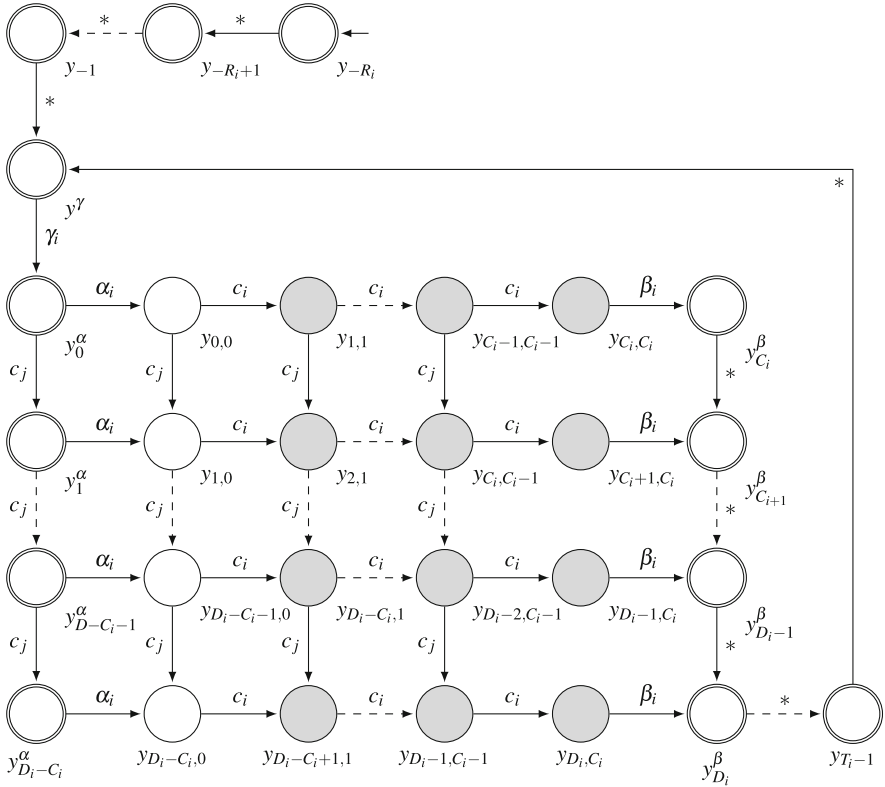


Fig. 5.2 General DES model for task τ_i

behavior is calculated by the synchronous product procedure. In accordance with Sect. 4.3, the utilized TCT procedures are introduced in [20].

TASK2 = create (TASK2, [mark 0, 1, 2, 3, 4, 9, 10, 15, 16, 21, 22], [tran [0, 0, 1], [0, 19, 1], [0, 39, 1], [0, 49, 1], [1, 0, 2], [1, 19, 2], [1, 39, 2], [1, 49, 2], [2, 0, 3], [2, 19, 3], [2, 39, 3], [2, 49, 3], [3, 20, 4], [4, 19, 10], [4, 21, 5], [4, 39, 10], [4, 49, 10], [5, 19, 11], [5, 29, 6], [5, 39, 11], [5, 49, 11], [6, 19, 12], [6, 29, 7], [6, 39, 12], [6, 49, 12], [7, 19, 13], [7, 29, 8], [7, 39, 13], [7, 49, 13], [8, 22, 9], [9, 0, 15], [9, 19, 15], [9, 39, 15], [9, 49, 15], [10, 19, 16], [10, 21, 11], [10, 39, 16], [10, 49, 16], [11, 19, 17], [11, 29, 12], [11, 39, 17], [11, 49, 17], [12, 19, 18], [12, 29, 13], [12, 39, 18], [12, 49, 18], [13, 19, 19], [13, 29, 14], [13, 39, 19], [13, 49, 19], [14, 22, 15], [15, 0, 21], [15, 19, 21], [15, 39, 21], [15, 49, 21], [16, 19, 22], [16, 21, 17], [16, 39, 22], [16, 49, 22], [17, 19, 23], [17, 29, 18], [17, 39, 23], [17, 49, 23], [18, 19, 24], [18, 29, 19], [18, 39, 24], [18, 49, 24], [19, 19, 25], [19, 29, 20], [19, 39, 25], [19, 49, 25], [20, 22, 21], [21, 0, 3], [21, 19, 3], [21, 39, 3], [21, 49, 3], [22, 21, 23], [23, 29, 24], [24, 29, 25], [25, 29, 26], [26, 22, 3]]) (27, 81)

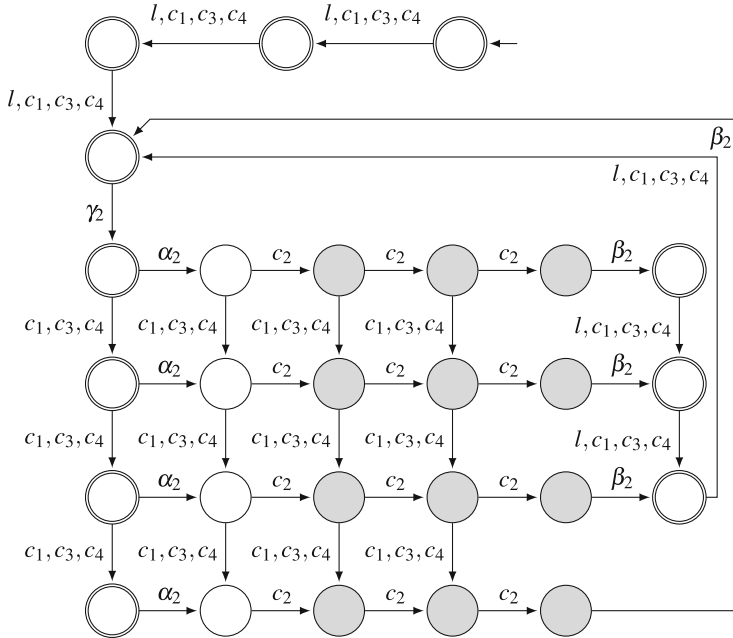


Fig. 5.3 DES model G_2

Suppose that we have an RTS \mathbb{S} that possesses only one processor; three scheduling plans are discussed below.

Plan 1: τ_1 and τ_2 in Process

Suppose that tasks τ_1 and τ_2 are processed in an RTS \mathbb{S} , denoted by \mathbb{S}_1 . It is generated by the procedure **sync** provided by TCT. Since tasks τ_3 and τ_4 are not in \mathbb{S}_1 , within τ_1 and τ_2 , we can eliminate c_3 and c_4 (events 39 and 49) by relabelling them to be c_2 (event 29) and c_1 (event 19), respectively. Thereafter, the DES model representing \mathbb{S}_1 can be generated by a synchronous product. In TCT, \mathbb{S}_1 is represented by SYS1 that contains 71 states and 98 transitions. The corresponding TCT procedures are (all the **sync** operations in the original MAKEIT.TXT file are reported with the message “Blocked_events = None”, eliminated in this monograph for readability):

$$\text{TEST1} = \mathbf{relabel} (\text{TASK1}, [[39, 29], [49, 29]]) (42, 72)$$

$$\text{TEST2} = \mathbf{relabel} (\text{TASK2}, [[39, 19], [49, 19]]) (27, 45)$$

$$\text{SYS1} = \mathbf{sync} (\text{TEST1}, \text{TEST2}) (71, 98)$$

Plan 2: τ_1 , τ_2 , and τ_4 in Process

Suppose that tasks τ_1 , τ_2 , and τ_4 are processed in RTS \mathbb{S} , denoted by \mathbb{S}_2 . It is represented by SYS2 in TCT, and can be generated in a similar way, i.e.,

$$\text{TEST1} = \text{relabel}(\text{TASK1}, [[39, 29]]) (42, 102)$$

$$\text{TEST2} = \text{relabel}(\text{TASK2}, [[39, 19]]) (27, 63)$$

$$\text{TEST4} = \text{relabel}(\text{TASK4}, [[39, 19]]) (69, 173)$$

$$\text{SYS2} = \text{sync}(\text{TEST1}, \text{TEST2}, \text{TEST4}) (170, 279)$$

Plan 3: τ_1 , τ_2 , τ_3 , and τ_4 in Process

Suppose that all four tasks τ_1 , τ_2 , τ_3 , and τ_4 are processed in RTS \mathbb{S} , denoted by \mathbb{S}_3 . Represented by SYS3 in TCT, \mathbb{S}_3 can be generated in a similar way, i.e.,

$$\text{SYS3} = \text{sync}(\text{TASK1}, \text{TASK2}, \text{TASK3}, \text{TASK4}) (952, 2056) \text{ Blocked_events} = [0]$$

This shows that there is a blocked event 0 (l) in SYS3 (\mathbb{S}_3), which represents that in the real-time scheduling, there is no idle time unit. This means that the processor utilization of \mathbb{S}_3 is $U_3 \geq 1$ [12]. By calculating the processor utilization of \mathbb{S}_3 , we obtain

$$U_3 = \frac{3}{9} + \frac{3}{6} + \frac{1}{5} + \frac{2}{14} > 1.$$

Thus, \mathbb{S}_3 is non-schedulable. Hence, it can be reconfigured by following the approach proposed in Chap. 4. Furthermore, based on a modular modelling approach, the scheduling/reconfiguration approach is generalized in Chap. 6.

In the modelling phase, “bad decisions” made by the synchronous product procedure may block the PFCP real-time scheduling. As a solution, in the rest of this chapter, SCT is utilized to supervise the RTS to be nonblocking.

5.4 Specifications Modelled by Discrete-Event Systems

In accordance with [20], all possible behaviors in a processor of an RTS are generated by a DES, called the *plant*. Hence, the behavior of an RTS under control is a subset of the generated languages with respect to certain constraints that are provided by some specification languages. In order to schedule the processor to be nonblocking and conditionally-preemptive, we shall impose the synchronous product of proper specifications on the behavior of the processor. For each task running in a processor, four types of specifications are defined:

- *nonblocking specifications*: nonblocking preemptive scheduling of real-time tasks,
- *PFCP specifications*: the preemption relation among all the tasks,

- *task-centered conditional-preemption specifications* : during the execution of each task, the exact preemption plan between two adjacent time units, and
- *WCRT-based specifications*: the WCRT in all the periods.

According to the study in [20] and Chap. 4, given any RTS represented by \mathbf{G} , for any specification, selfloops of events appeared in \mathbf{G} but not in the specification must be adjoined to account for all the events that are irrelevant to the specification but possibly executed in the plant. For simplification, in this section, we build the specification by the **create** procedure in TCT; thereafter the selfloops are adjoined by the procedures **allevents** and **sync**. The selfloop of events for τ_i is viewed as its nonblocking specification.

5.4.1 Nonblocking Specifications

In order to control the RTS to be nonblocking, the specification \mathbf{S}_i^N for task τ_i should allow the occurrence of any strings over Σ_i . Formally, we have:

$$L(\mathbf{S}_i^N) = \Sigma_i^*.$$

The TCT procedure **allevents** can be utilized to generate a DES representing Σ_i^* . As shown in Fig. 5.4, the nonblocking specification for task τ_i is a generator with only one state at which all the events appeared in Σ_i are enabled.

The corresponding TCT operations to create such specifications are provided. SN1, SN2, SN3, and SN4 are the nonblocking specifications \mathbf{S}_1^N , \mathbf{S}_2^N , \mathbf{S}_3^N , and \mathbf{S}_4^N , respectively. In TCT, the monolithic nonblocking specification for an RTS \mathbb{S} is denoted by SN. The corresponding TCT operations are given below.

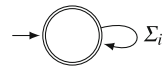
$$\text{SN1} = \text{allevents}(\text{TASK1})(1, 8)$$

$$\text{SN2} = \text{allevents}(\text{TASK2})(1, 8)$$

$$\text{SN3} = \text{allevents}(\text{TASK3})(1, 8)$$

$$\text{SN4} = \text{allevents}(\text{TASK4})(1, 8)$$

Fig. 5.4 Nonblocking specification for task τ_i



5.4.2 Matrix-Based Priority-Free Conditional-Preemption Specifications

In a processor, all the possible preemptions that may occur during the execution of task τ_i are defined in the i -th row of the preemption matrix \mathbf{A} . More precisely, task τ_i can be preempted by τ_j if $\mathbf{A}_{i,j} = 1$. The preemption occurs between the occurrences of α_i and β_i . Thus, a matrix-based PFCP specification $\mathbf{S}_i^{\mathbf{A}}$ is defined for a task τ_i with a generator

$$\mathbf{S}_i^{\mathbf{A}} = (Q_i^{\mathbf{A}}, \Sigma_i^{\mathbf{A}}, \delta_i^{\mathbf{A}}, q_{0i}^{\mathbf{A}}, Q_{mi}^{\mathbf{A}}).$$

Here

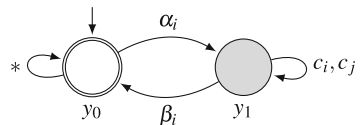
- $Q_i^{\mathbf{A}}$: the state set that contains two states:
 - y_0 : task τ_i is not in process and
 - y_1 : task τ_i is in process,
- $\Sigma_i^{\mathbf{A}} = \bigcup_{i \in \mathbf{n}} \Sigma_i$: the set of all the events appearing in the processor,
- $\delta_i^{\mathbf{A}}$: the (partial) transition function:
 - $\delta_i^{\mathbf{A}}(y_0, \sigma) = y_0, \sigma \in \Sigma_e - \{c_i\}$: τ_i is not in process, and the time unit can be taken by other tasks or idle,
 - $\delta_i^{\mathbf{A}}(y_1, c_i) = y_1$: task τ_i is in process,
 - $\delta_i^{\mathbf{A}}(y_1, c_j) = y_1$: task τ_i can be preempted by τ_j , i.e., $\mathbf{A}_{i,j} = 1$,
 - $\delta_i^{\mathbf{A}}(y_0, \alpha_i) = y_1$: the execution of τ_i is started, and
 - $\delta_i^{\mathbf{A}}(y_1, \beta_i) = y_0$: the execution of τ_i is completed,
- $q_{0,i}^{\mathbf{A}} = y_0$ is the initial state, and
- $Q_{m,i}^{\mathbf{A}} = \{q_{0,i}^{\mathbf{A}}\}$ is the subset of marker states.

The DES model of specification $\mathbf{S}_i^{\mathbf{A}}$ for τ_i is illustrated in Fig. 5.5, where

- * represents the events in $\Sigma_e - \{c_i\}$, and
- c_j represents the execution of task τ_j that is allowed to preempt the execution of τ_i .

As a result, the TCT operations to create the specifications corresponding to matrix \mathbf{A}_2 are listed below, in which 1NP, 2NP, 3NP, and 4NP are the non-preemptive specifications $\mathbf{S}_1^{\mathbf{A}}$, $\mathbf{S}_2^{\mathbf{A}}$, $\mathbf{S}_3^{\mathbf{A}}$, and $\mathbf{S}_4^{\mathbf{A}}$ for task τ_1 , τ_2 , τ_3 , and τ_4 , respectively. Each specification represents that the execution of the corresponding task cannot be preempted.

Fig. 5.5 Matrix-based conditional-preemption specification



1NP = **create** (1NP, [mark 0], [tran [0, 0, 0], [0, 11, 1], [0, 29, 0], [0, 39, 0], [0, 49, 0], [1, 12, 0], [1, 19, 1]]) (2, 7)

2NP = **create** (2NP, [mark 0], [tran [0, 0, 0], [0, 19, 0], [0, 39, 0], [0, 49, 0], [0, 21, 1], [1, 22, 0], [1, 29, 1]]) (2, 7)

3NP = **create** (3NP, [mark 0], [tran [0, 0, 0], [0, 19, 0], [0, 29, 0], [0, 31, 1], [0, 49, 0], [1, 32, 0], [1, 39, 1]]) (2, 7)

4NP = **create** (4NP, [mark 0], [tran [0, 0, 0], [0, 19, 0], [0, 29, 0], [0, 39, 0], [0, 41, 1], [1, 42, 0], [1, 49, 1]]) (2, 7)

Moreover, the matrix-based PFCP specifications are created for matrix \mathbf{A}_5 :

- 1B34: the execution of task τ_1 can be preempted by τ_3 and τ_4 , and
- 2B4: the execution of task τ_2 can be preempted by τ_4 .

The corresponding TCT operations are listed below.

$$1B34 = \mathbf{edit} (1NP, [\text{trans } +[1, 39, 1], +[1, 49, 1]]) (2, 9)$$

$$2B4 = \mathbf{edit} (2NP, [\text{trans } +[1, 49, 1]]) (2, 8)$$

Similarly, the PFCP specifications are created for matrix \mathbf{A}_6 :

- 1B2: the execution of task τ_1 can be preempted by τ_2 ,
- 2B4: the execution of task τ_2 can be preempted by τ_4 (DES created already), and
- 4B1: the execution of task τ_4 can be preempted by τ_1 .

The TCT operations are given below.

$$1B2 = \mathbf{edit} (1NP, [\text{trans } +[1, 29, 1]]) (2, 8)$$

$$4B1 = \mathbf{edit} (4NP, [\text{trans } +[1, 19, 1]]) (2, 8)$$

5.4.3 Task-Centered Specifications

From the perspective of each individual task, task-centered specifications define the exact preemption plans between the occurrences of two adjacent time units (c_i 's) representing the execution of task τ_i . A task-centered conditional-preemption specification \mathbf{S}_i^C is defined for each task τ_i with a generator

$$\mathbf{S}_i^C = (Q_i^C, \Sigma_i^C, \delta_i^C, q_{0i}^C, Q_{mi}^C).$$

Here $*$ represents the events in $\Sigma_e - \{c_i\}$, and

- Q_i^C : the state set containing $C_i + 2$ states,
- $\Sigma_i^C = \bigcup_{i \in \mathbf{en}} \Sigma_i$: the set of all the events appearing in the processor,
- $(0 \leq k < C_i) \Sigma_e^k$: user defined unempty subset of Σ_e that contains several events c_j representing the preemption of task τ_j ,
- δ_i^C : the (partial) transition function:
 - $(\sigma \in \Sigma_e - \{c_i\}) \delta_i^C(y_0^\alpha, \sigma) = y_0^\alpha$ and $\delta_i^C(y_{C_i}, \sigma) = y_{C_i}$: τ_i is not in process; the time unit can be occupied by other tasks or idle,
 - $\delta_i^C(y_0^\alpha, \alpha_i) = y_0$: task τ_i has arrived,
 - $(0 \leq k < C_i) \delta_i^C(y_k, c_i) = y_{k+1}$: task τ_i is in process,
 - $(0 \leq k < C_i, c_j \in \Sigma_e^k) \delta_i^C(y_k, c_j) = y_k$: τ_i is preempted by the execution of τ_j , and
 - $\delta_i^C(y_{C_i}, \beta_i) = y_0^\alpha$: the execution of τ_i is completed,
- $q_{0,i}^C = y_0^\alpha$: the initial state, and
- $Q_{m,i}^C = \{y_0^\alpha\}$: the subset of marker states.

The DES model of specification S_i^C for τ_i is shown in Fig. 5.6, in which the selfloops are labelled by event c_j in Σ_e corresponding to task τ_j , $j \neq i$, that can preempt the execution of task τ_i . Let $0 \leq k < C_i$. The preemption of the k -th execution time unit is defined at state y_k . More precisely, if event c_j is in event set Σ_e^k that is selflooped at state y_k , then the k -th execution time unit is allowed to be preempted by τ_j .

The corresponding task-centered conditional-preemption specifications are:

- ST1: the last time unit of τ_1 cannot be preempted by τ_2 ,
- ST2: the last time unit of τ_2 cannot be preempted by τ_1 ,
- ST3: the last time unit of τ_1 cannot be preempted by τ_4 , and
- ST4: the last time unit of τ_2 cannot be preempted by τ_4 .

The corresponding TCT operations to create specifications are listed below.

ST1 = **create** (ST1, [mark 0], [tran [0, 0, 0], [0, 11, 1], [0, 29, 0], [0, 39, 0], [0, 49, 0], [1, 19, 2], [1, 29, 1], [1, 39, 1], [1, 49, 1], [2, 19, 3], [2, 39, 2], [2, 49, 2], [3, 12, 0], [3, 19, 3], [3, 29, 3], [3, 39, 3], [3, 49, 3]]) (4, 17)

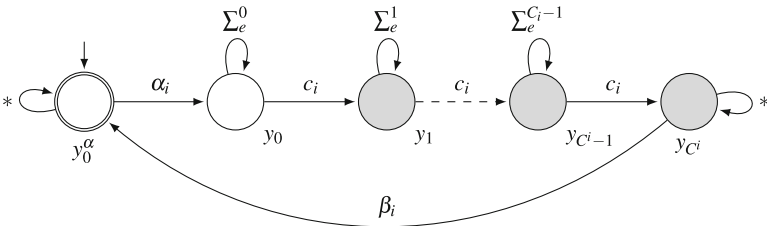


Fig. 5.6 Task-centered conditional-preemption specification

ST2 = **create** (ST2, [mark 0], [tran [0, 0, 0], [0, 19, 0], [0, 21, 1], [0, 39, 0], [0, 49, 0], [1, 19, 1], [1, 29, 2], [1, 39, 1], [1, 49, 1], [2, 19, 2], [2, 29, 3], [2, 39, 2], [2, 49, 2], [3, 29, 4], [3, 39, 3], [3, 49, 3], [4, 19, 4], [4, 22, 0], [4, 39, 4], [4, 49, 4]]) (5, 20)

ST3 = **edit** (ST1, [trans +[2, 29, 2], -[2, 49, 2]]) (4, 17)

ST4 = **edit** (ST2, [trans +[3, 19, 3], -[3, 49, 3]]) (5, 20)

5.4.4 Response Time Constraint Specifications

The preemption of real-time execution increases the response time of task τ_i . In order to constrain that the execution time of task τ_i is not longer than a value of WCRT \mathbb{W}_i , i.e., in a period T_i , the execution time between the occurrences of events γ_i and β_i is limited to be no greater than \mathbb{W}_i time units, a WCRT-based specification $\mathbf{S}_i^{\mathbb{W}}$ is defined for task τ_i with a generator

$$\mathbf{S}_i^{\mathbb{W}} = (Q_i^{\mathbb{W}}, \Sigma_i^{\mathbb{W}}, \delta_i^{\mathbb{W}}, q_{0,i}^{\mathbb{W}}, Q_{m,i}^{\mathbb{W}}),$$

where

- $Q_i^{\mathbb{W}}$: the state set containing $\mathbb{W}_i + 2$ states,
- $\Sigma_i^{\mathbb{W}} = \Sigma_i - \{l\}$: the set of all the events except l appearing in \mathbf{G}_i ,
- $\delta_i^{\mathbb{W}}$: the (partial) transition function:
 - $\delta_i^{\mathbb{W}}(y^\gamma, \sigma) = y^\gamma, \sigma \in \Sigma_e - \{c_i\}$: τ_i is not in process, and the time unit can be taken by other tasks or idle,
 - $\delta_i^{\mathbb{W}}(y^\gamma, \gamma_i) = y_0$: τ_i is released,
 - $(0 \leq k < \mathbb{W}_i, \sigma \in \Sigma_e - \{l\}) \delta_i^{\mathbb{W}}(y_k, \sigma) = y_{k+1}$: task τ_j is in process, and
 - $(0 < k < \mathbb{W}_i) \delta_i^{\mathbb{W}}(y_k, \beta_i) = y_{k+1}$: the execution of τ_i is completed,
- $q_{0,i}^{\mathbb{W}} = y^\gamma$: the initial state, and
- $Q_{m,i}^{\mathbb{W}} = \{y^\gamma\}$: the subset of marker states.

The DES model of specification $\mathbf{S}_i^{\mathbb{W}}$ for τ_i is shown in Fig. 5.7, where * and ** represents the events in $\Sigma_e - \{c_i\}$ and $\Sigma_e - \{l\}$, respectively.

Example For the RTS tasks shown in Table 5.1, a possible set of WCRT-based specifications are:

- SR1: the WCRT of task τ_1 is $\mathbb{W}_1 = 4$, and
- SR4: the WCRT of task τ_4 is $\mathbb{W}_4 = 2$.

The corresponding TCT operations to create specifications are listed below.

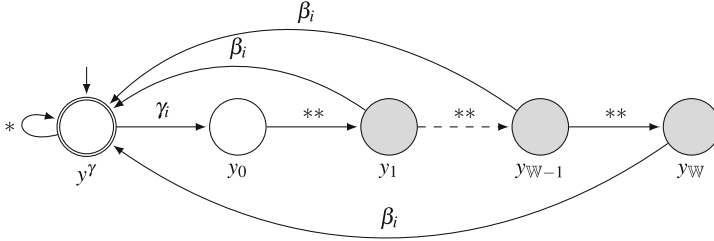


Fig. 5.7 WCRT-based conditional-preemption specification

SR1 = **create** (SR1, [mark 0], [tran [0, 0, 0], [0, 10, 1], [0, 29, 0], [0, 39, 0], [0, 49, 0], [1, 19, 2], [1, 29, 2], [1, 39, 2], [1, 49, 2], [2, 12, 0], [2, 19, 3], [2, 29, 3], [2, 39, 3], [2, 49, 3], [3, 12, 0], [3, 19, 4], [3, 29, 4], [3, 39, 4], [3, 49, 4], [4, 12, 0], [4, 19, 5], [4, 29, 5], [4, 39, 5], [4, 49, 5], [5, 12, 0]]) (6, 25)

SR4 = **create** (SR4, [mark 0], [tran [0, 0, 0], [0, 19, 0], [0, 29, 0], [0, 39, 0], [0, 40, 1], [1, 19, 2], [1, 29, 2], [1, 39, 2], [1, 49, 2], [2, 19, 3], [2, 29, 3], [2, 39, 3], [2, 42, 0], [2, 49, 3], [3, 42, 0]]) (4, 16) \square

5.5 Case Study I: Supervisor Synthesis of Motor Network

So far, the priority-free scheduling policy with conditional-preemption is described in regular languages that can be represented by DES. It is well known that SCT can be used to find the supremal controllers that provide the *minimally restricted controller* of the systems. By utilizing the procedure **sync** in TCT, all the specifications can be integrated into a unique one. The procedure **supcon** in TCT finds all the safe execution sequences within an RTS satisfying the synchronized specification. Users need not be concerned with the mathematical calculations; by utilizing TCT, all the safe execution sequences are provided in the supervisor. Each sequence can be utilized by users to schedule the RTS. All the EDF, FP, and other sequences can be found in the synthesized supervisor.

As illustrated in Fig. 5.8, an example *motor network* similar to the one studied in [3] is considered as an RTS. Suppose that four electric motors are controlled by a uni-processor. Their deadlines and periods are represented by D and T , respectively. These parameters coincide with those of the tasks in the previous examples as

- Motor 1: τ_1 ,
- Motor 2: τ_2 ,
- Motor 3: τ_3 , and
- Motor 4: τ_4 .

The work plans of the motor network also coincide with the RTS models presented in Sect. 5.3. Since $R_2 = 3$, Motor 2 will be ready three ms later than other tasks. We

Fig. 5.8 A motor network example

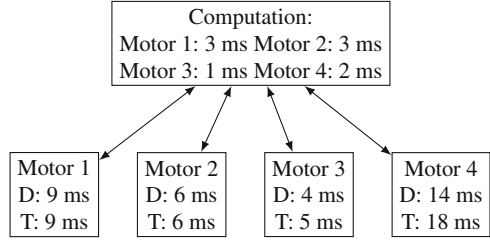


Table 5.2 Uni-processor scheduler behaviors of \mathbb{S}_1

Plan	Spec	Super	LM	SM
1	A_1	(71, 98)	(6, 7)	(24, 33)
2	A_2	(58, 70)	(6, 7)	(20, 21)
3	A_3	(69, 87)	(6, 7)	(24, 27)
4	A_4	(69, 87)	(6, 7)	(24, 27)
5	$A_1, ST1, ST2$	(68, 90)	(6, 7)	(23, 27)
6	$A_3, ST1$	(65, 82)	(6, 7)	(22, 24)

take work plans I and II of \mathbb{S} , denoted by \mathbb{S}_1 and \mathbb{S}_2 , to find all the safe execution sequences under priority-free conditionally-preemptive scheduling.

5.5.1 Work Plan I

Since $\tau_1, \tau_2 \in \mathbb{S}_1$, we only consider the specifications corresponding to tasks τ_1 and τ_2 . The synthesized supervisor is shown in Table 5.2, in which LM and SM represent the release map and the scheduling map, respectively. The numbers of the states and transitions are recorded in the form (number of states, number of transitions). Based on supervisory control of DES, several other examples of different specifications are also listed in Table 5.2. For \mathbb{S}_1 , we have

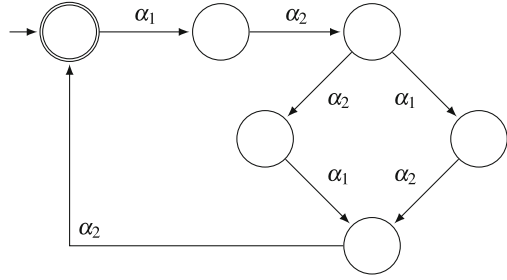
- A_1 : SN1 and SN2,
- A_2 : 1NP, 2NP, SN1, and SN2,
- A_3 : 2NP, SN1, and SN2, and
- A_4 : 1NP, SN1, and SN2.

RTS Scheduling Plan 1

The scheduling corresponding to Plan 1 (A_1) is preemptive. Thus, we only need to control the RTS to be nonblocking. Denote this specification by PS1. In TCT, it is calculated by

$$PS1 = \mathbf{sync} (SN1, SN2) (1, 11).$$

All the safe preemptive execution sequences are calculated by the TCT procedure **supcon**, i.e.,

Fig. 5.9 Release map of \mathbb{S}_1 

$$\text{SUPER1} = \mathbf{supcon} (\text{SYS1}, \text{PS1}) (71, 98).$$

The safe execution sequences in SUPER1 are represented by a DES with 71 states and 98 transitions. By projecting out all events but α_i , i.e.,

$$\text{PMSUPER1} = \mathbf{project} (\text{SUPER1}, \text{Image} [11, 21]) (6, 7).$$

We obtain the *preemptive release map* of \mathbb{S}_1 in the trimmed version of PMSUPER1 as follows:

$$\text{PMSUPER1} = \mathbf{trim} (\text{PMSUPER1}) (6, 7).$$

All the *safe release sequences* are shown in Fig. 5.9. In SUPER1, by projecting out all events but c_i , i.e.,

$$\text{PJSUPER1} = \mathbf{project} (\text{SUPER1}, \text{Image} [19, 29]) (24, 33).$$

In the trimmed version below, i.e.,

$$\text{PJSUPER1} = \mathbf{trim} (\text{PJSUPER1}) (24, 33),$$

we obtain the *preemptive scheduling map* of \mathbb{S}_1 , as shown in Fig. 5.10a, that contains all the safe execution sequences of c_1 and c_2 . The other marker states except the initial states in PMSUPER1 and PJSUPER1 represent that the corresponding task is ready to be released, which is redundant information for the users. Thus they are unmarked, similar to such diagrams of other scheduling plans. PMSUPER1 and PJSUPER1 are unmarked below.

$$\text{PMSUPER1} = \mathbf{edit} (\text{PMSUPER1}, [\text{mark} -[\text{all}]]) (6, 7)$$

$$\text{PMSUPER1} = \mathbf{edit} (\text{PMSUP1}, [\text{mark} +[0]]) (6, 7)$$

$$\text{PJSUPER1} = \mathbf{edit} (\text{PMSUPER1}, [\text{mark} -[\text{all}]]) (24, 33)$$

$$\text{PJSUPER1} = \mathbf{edit} (\text{PJSUP1}, [\text{mark} +[0]]) (24, 33)$$

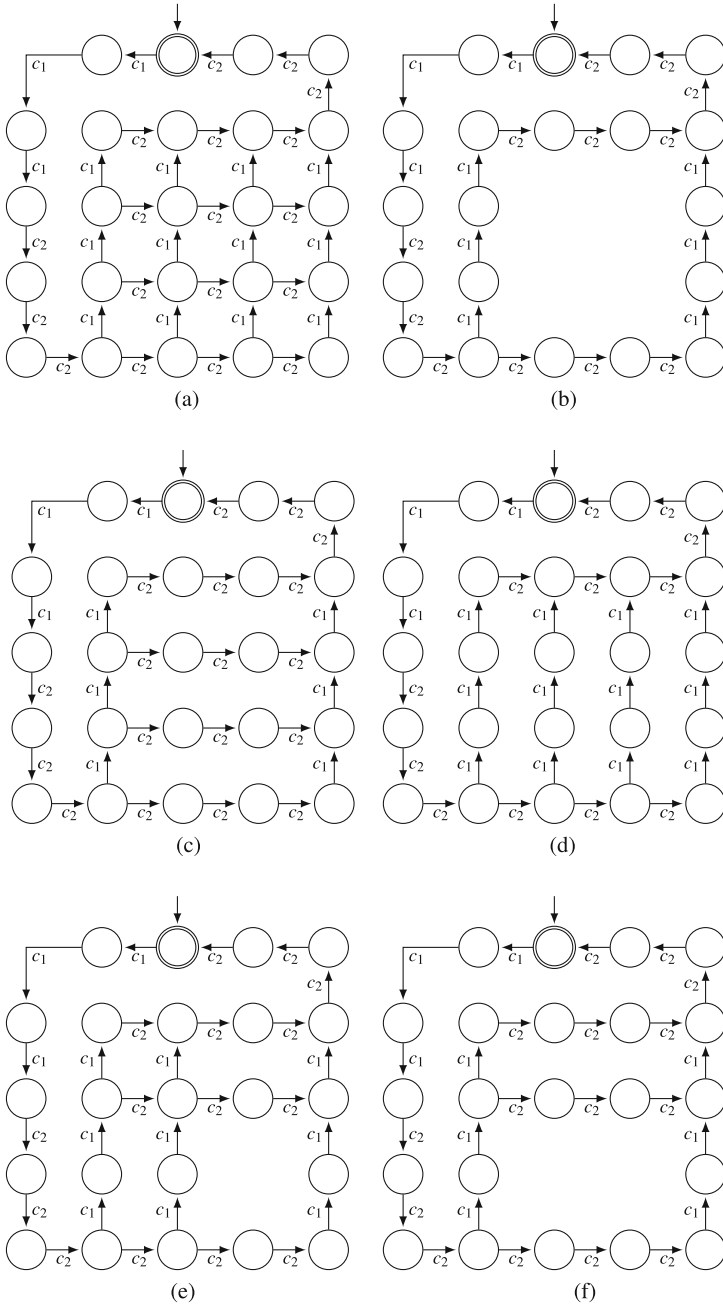


Fig. 5.10 Scheduling map of S_1 . (a) A_1 . (b) A_2 . (c) A_3 . (d) A_4 . (e) A_1 , SC1, and SC2. (f) A_3 and SC1

RTS Scheduling Plan 2

The scheduling corresponding to Plan 2 (A_2) shown in Table 5.2 is non-preemptive. The corresponding TCT operations are listed below.

PS2 = **sync** (1NP, 2NP, SN1, SN2) (4, 21)

SUPER2 = **supcon** (SYS1, PS2) (58, 70)

PMSUPER2 = **project** (SUPER2, Image [11, 21]) (6, 7)

true = **isomorph** (PMSUPER1, PMSUPER2; identity)

PJSUPER2 = **project** (SUPER2, Image [19, 29]) (20, 21)

RTS Scheduling Plan 3

The corresponding TCT operations for Plan 3 (A_3) shown in Table 5.2 are listed below.

PS3 = **sync** (2NP, SN1, SN2) (2, 15)

SUPER3 = **supcon** (SYS1, PS3) (69, 87)

PMSUPER3 = **project** (SUPER3, Image [11, 21]) (6, 7)

true = **isomorph** (PMSUPER1, PMSUPER3; identity)

PJSUPER3 = **project** (SUPER3, Image [19, 29]) (24, 27)

RTS Scheduling Plan 4

The corresponding TCT operations for Plan 4 (A_4) shown in Table 5.2 are listed below.

PS4 = **sync** (1NP, SN1, SN2) (2, 15)

SUPER4 = **supcon** (SYS1, PS4) (69, 87)

PMSUPER4 = **project** (SUPER4, Image [11, 21]) (6, 7)

true = **isomorph** (PMSUPER4, PMSUPER1; identity)

PJSUPER4 = **project** (SUPER4, Image [19, 29]) (24, 27)

false = **isomorph** (PJSUPER4, PJSUPER3)

RTS Scheduling Plan 5

The corresponding TCT operations for Plan 5 shown in Table 5.2 are listed below.

$$\text{PS5} = \mathbf{sync} (\text{PS1}, \text{ST1}, \text{ST2}) (19, 116)$$

$$\text{SUPER5} = \mathbf{supcon} (\text{SYS1}, \text{PS5}) (68, 90)$$

$$\text{PMSUPER5} = \mathbf{project} (\text{SUPER5}, \text{Image [11, 21]}) (6, 7)$$

$$\text{true} = \mathbf{isomorph} (\text{PMSUPER5}, \text{PMSUPER1}; \text{identity})$$

$$\text{PJSUPER5} = \mathbf{project} (\text{SUPER5}, \text{Image [19, 29]}) (23, 27)$$

RTS Scheduling Plan 6

The corresponding TCT operations for Plan 6 shown in Table 5.2 are listed below.

$$\text{PS6} = \mathbf{sync} (\text{PS3}, \text{ST1}) (8, 43)$$

$$\text{SUPER6} = \mathbf{supcon} (\text{SYS1}, \text{PS6}) (65, 82)$$

$$\text{PMSUPER6} = \mathbf{project} (\text{SUPER6}, \text{Image [11, 21]}) (6, 7)$$

$$\text{true} = \mathbf{isomorph} (\text{PMSUPER6}, \text{PMSUPER1}; \text{identity})$$

$$\text{PJSUPER6} = \mathbf{project} (\text{SUPER6}, \text{Image [19, 29]}) (22, 24)$$

Supcon Results

All the release maps are isomorphic with the release map depicted in Fig. 5.9. Moreover, all the scheduling maps are depicted in Fig. 5.10. This means that, based on a supremal release map, according to different preemption plans, the priority-free conditionally-preemptive scheduling policy can provide different results. To the best of our knowledge, no other scheduling algorithms can schedule an RTS by considering A_3 and $ST1$ simultaneously.

In an RTS, under EDF, the tasks with the earliest deadlines are assigned with the highest priority. EDF scheduling chooses only one task among them to execute without considering other possibilities. Other released tasks have no chance to be executed by the processor.

As a comparison, the preemptive EDF scheduling result of S_1 by Cheddar [17] is depicted in Fig. 5.11, which can also be found in Fig. 5.10a. The execution sequence in Fig. 5.11 looks like a non-preemptive scheduling sequence. However, none of the exact preemptive scheduling sequences in Fig. 5.10a can be generated by EDF. Hence, as a reconfiguration scenario, in Fig. 5.11, if task τ_2 (with the earliest deadline) cannot arrive on time at the ninth time unit, according to the multiple

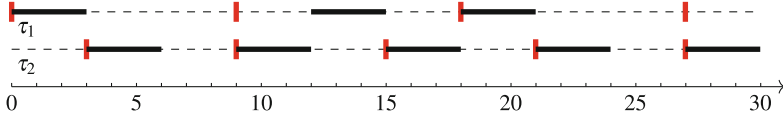


Fig. 5.11 Preemptive scheduling map of \mathbb{S}_1

Table 5.3 Uni-processor scheduler behaviors of \mathbb{S}_1

Plant	Spec	Super	LM	SM
SYS1	PS3	SUPER1 (69, 87)	(6, 7)	(24, 27)
SUPER1	ST1, PS1	SUPER2 (65, 82)	(6, 7)	(22, 24)
SUPER2	SR1, PS1	SUPER3 (45, 52)	(5, 5)	(15, 15)

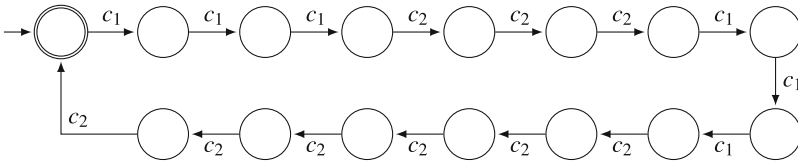


Fig. 5.12 Preemptive scheduling with $\mathbb{W}_1 = 4$

sequences, users can choose another available sequence shown in Fig. 5.10a to schedule task τ_1 first. Thus, recalculating the scheduling sequences is unnecessary. However, at the ninth time unit, there is no EDF sequence to execute τ_1 first. If τ_2 cannot arrive on time, the EDF scheduling cannot schedule \mathbb{S}_1 successfully. The supervisory control technique provides a larger number of safe execution sequences, compared with EDF scheduling.

The WCRT of τ_1 in the scheduling sequence shown in the Gantt chart depicted in Fig. 5.11 is $\mathbb{W}_1 = 6$. By comparison, in the scheduling map shown in Fig. 5.10a, when tasks τ_1 and τ_2 are released simultaneously, one can preempt another randomly. In all of these scheduling sequences, we have $\mathbb{W}_1 \leq 6$.

By following the speeding up approach presented in Chap. 1, the scheduling of \mathbb{S}_1 based on \mathbf{A}_3 and ST1 can be calculated in the first two steps as listed in Table 5.3. Moreover, if we require that the WCRT of task τ_1 be $\mathbb{W}_1 = 4$, only one such sequence exists, as stated in SUPER3 in Table 5.3, which is shown in Fig. 5.12. Accordingly, \mathbb{S}_1 can be scheduled in the order $\tau_1 \tau_2 \tau_1 \tau_2 \tau_2$. Notice that specifications ST1 and SR1 are synchronized with specification PS1 since it guarantees that the synthesis procedure is nonblocking.

5.5.2 Work Plan II

By $\tau_1, \tau_2, \tau_4 \in \mathbb{S}_2$, we only consider the specifications corresponding to tasks τ_1, τ_2 , and τ_4 . Several supervisors are calculated and listed in Table 5.4. The release maps are isomorphic with each other, as shown in Fig. 5.13. The scheduling maps of RTS

Table 5.4 Uni-processor scheduler behaviors of \mathbb{S}_2

Plan	Spec	Super	LM	SM
7	\mathbf{A}_5	(128, 186)	(9, 12)	(34, 41)
8	$\mathbf{A}_5, \text{ST3}, \text{ST4}$	(119, 173)	(9, 12)	(32, 37)
9	\mathbf{A}_6	(145, 215)	(9, 12)	(44, 54)
10	$\mathbf{A}_6, \text{ST1}$	(140, 204)	(9, 12)	(39, 47)

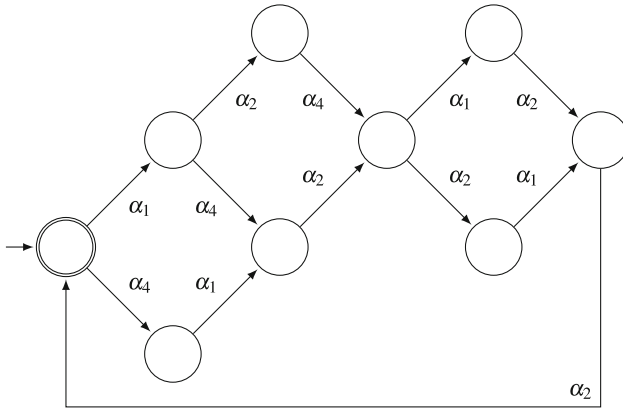


Fig. 5.13 Release map of \mathbb{S}_2

\mathbb{S}_2 are depicted in Figs. 5.14 and 5.15. To the best of our knowledge, no sequence in Figs. 5.14b and 5.15b can be achieved by other scheduling algorithms.

RTS Scheduling Plan 7

The corresponding TCT operations for Plan 7 shown in Table 5.4 are listed below.

$$\text{PS7} = \text{sync} (\text{SN1}, \text{SN2}, \text{SN4}, \text{1B34}, \text{2B4}, \text{4NP}) (8, 57)$$

$$\text{SUPER7} = \text{supcon} (\text{SYS2}, \text{PS7}) (128, 186)$$

$$\text{PMSUPER7} = \text{project} (\text{SUPER7}, \text{Image} [11, 21, 41]) (9, 12)$$

$$\text{PMSUPER7} = \text{trim} (\text{PMSUPER7}) (9, 12)$$

$$\text{PJSUPER7} = \text{project} (\text{SUPER7}, \text{Image} [19, 29, 49]) (34, 41)$$

RTS Scheduling Plan 8

The corresponding TCT operations for Plan 8 shown in Table 5.4 are listed below.

$$\text{PS8} = \text{sync} (\text{PS7}, \text{ST3}, \text{ST4}) (28, 159) \text{Blocked_events} = \text{None}$$

$$\text{SUPER8} = \text{supcon} (\text{SYS2}, \text{PS8}) (119, 173)$$

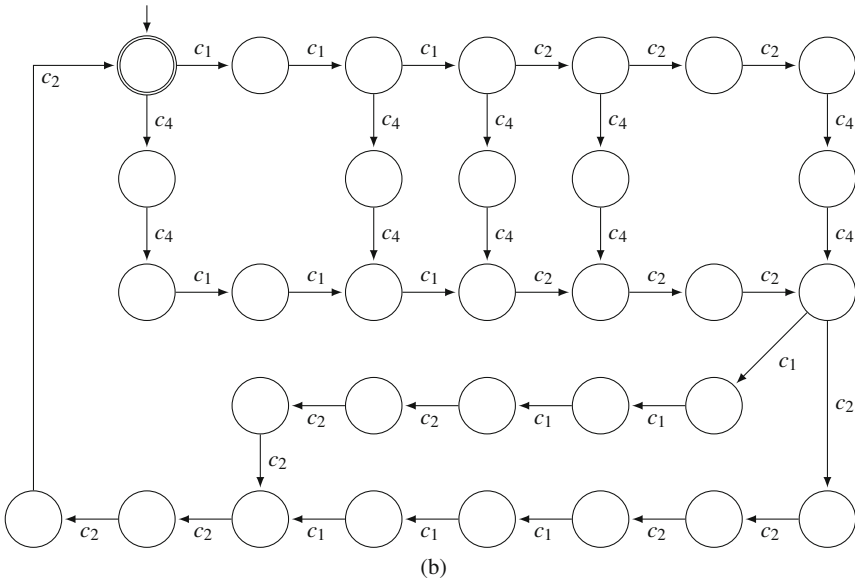
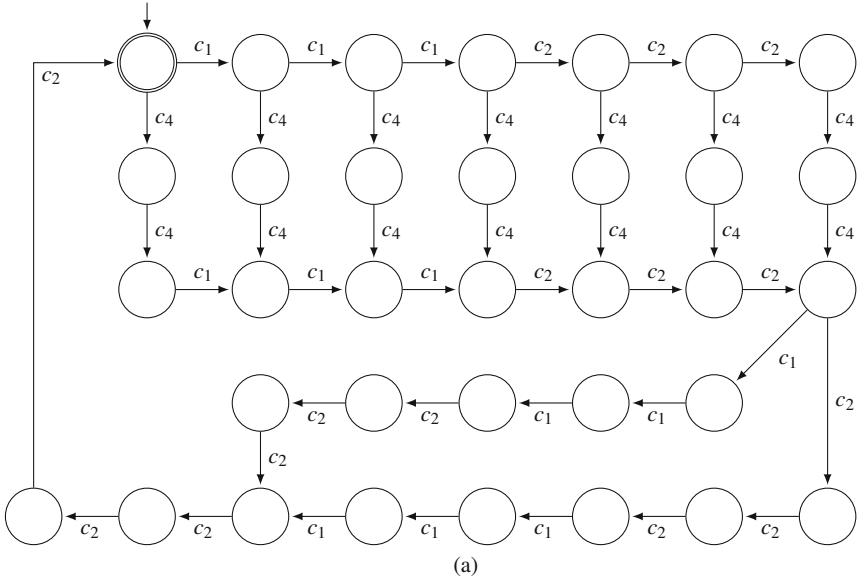


Fig. 5.14 Scheduling map of $\mathbb{S}_2(1)$. (a) A_5 . (b) A_5 , SC3, and SC4

$PMSUPER8 = \mathbf{project} (SUPER8, \text{Image } [11, 21, 41]) (9, 12)$

$\mathbf{true} = \mathbf{isomorph} (PMSUPER8, PMSUPER7; \text{identity})$

$PJSUPER8 = \mathbf{project} (SUPER8, \text{Image } [19, 29, 49]) (32, 37)$

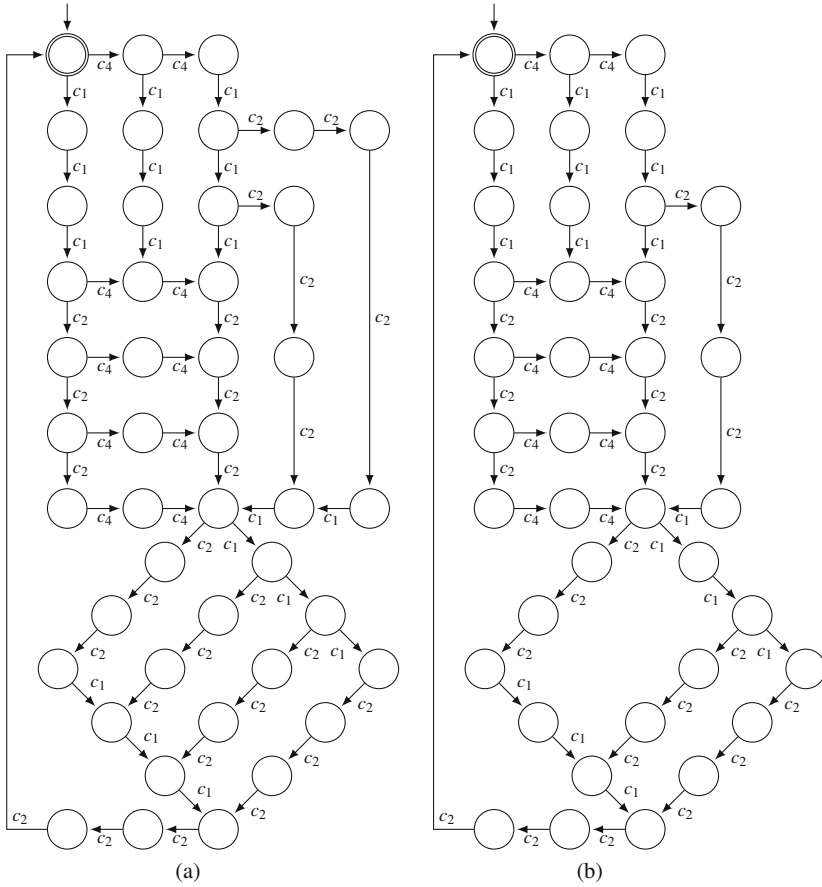


Fig. 5.15 Scheduling map of $\mathbb{S}_2(2)$. (a) A_6 . (b) A_6 and SC1

RTS Scheduling Plan 9

The corresponding TCT operations for Plan 9 shown in Table 5.4 are listed below.

$$PS9 = \text{sync} (SN1, SN2, SN4, SP12, 2B4, 4B1) (8, 56)$$

$$SUPER9 = \text{supcon} (SYS2, PS9) (149, 215)$$

$$PMSUPER9 = \text{project} (SUPER9, \text{Image} [11, 21, 41]) (9, 12)$$

$$\text{true} = \text{isomorph} (PMSUPER9, PMSUPER7; \text{identity})$$

$$PJSUPER9 = \text{project} (SUPER9, \text{Image} [19, 29, 49]) (44, 54)$$

RTS Scheduling Plan 10

The corresponding TCT operations for Plan 10 shown in Table 5.4 are listed below.

$$PS10 = \mathbf{sync} (PS9, ST1) (16, 101)$$

$$SUPER10 = \mathbf{supcon} (SYS2, PS10) (140, 204)$$

$$PMSUPER10 = \mathbf{project} (SUPER10, \text{Image } [11, 21, 41]) (9, 12)$$

$$\mathbf{true} = \mathbf{isomorph} (PMSUPER10, PMSUPER9; \text{identity})$$

$$PJSUPER10 = \mathbf{project} (SUPER10, \text{Image } [19, 29, 49]) (39, 47)$$

Supcon Results

The supervisor synthesized according to Plan 7 provides all the partial preemption scheduling sequences. For instance, the Gantt chart representing a partial preemption scheduling result of \mathbb{S}_2 is depicted in Fig. 5.16, which can also be found in Fig. 5.14a. Evidently, SCT provides a greater number of safe execution sequences w.r.t. partial preemption scheduling. By also considering SR4 as a specification, we obtain a supervisor with 83 states and 118 transitions. The corresponding release map is isomorphic with the one shown in Fig. 5.13. Two scheduling sequences are depicted in Fig. 5.17; they form a subset of the safe executions shown in Fig. 5.14a. The sequence in Fig. 5.16 can be found in Fig. 5.17.

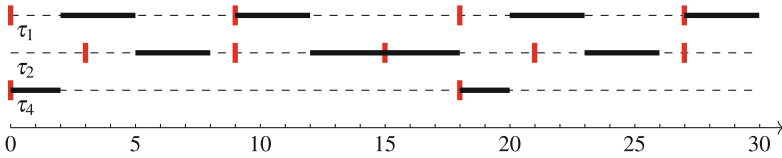


Fig. 5.16 PTS scheduling map of \mathbb{S}_2

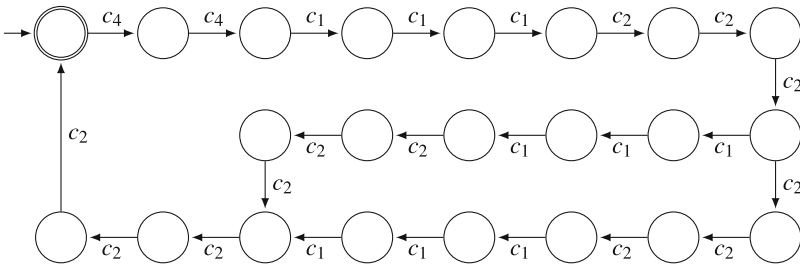


Fig. 5.17 Partial preemption and WCRT scheduling map of \mathbb{S}_2

5.6 Case Study II: Supervisor Synthesis of Manufacturing Cell

Consider a manufacturing cell as an example. As shown in Fig. 5.18, a robot R is utilized to transport two types of workpieces, W1 and W2, to a conveyor. Two pieces of W1 (resp., W2) are released to the input buffer B_1 (resp., B_2) simultaneously in every six (resp., three) seconds. The robot R has capacity one; transporting a piece takes 1 second. Thus, we define task $\tau_1 = (0, 2, 6, 6)$ (resp., $\tau_2 = (0, 2, 3, 3)$) to represent the transportation of the two pieces of W1 (resp., W2) by R. Consequently, we have a system $\mathbb{S} = \{\tau_1, \tau_2\}$.

Suppose that the preemption matrix with respect to \mathbb{S} is

$$A_1 = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}.$$

The scheduling can be considered as an FP scheduling, i.e., task τ_1 cannot be preempted by τ_2 , and τ_2 is allowed to be preempted by τ_1 . In other words, the robot must transport the two pieces of W1 in two adjacent seconds, but it is not necessary for transporting W2. By utilizing SCT, the supervisor calculated by **supcon** is represented by a DES with 21 states and 27 transitions. As shown in Fig. 5.19, the release (resp., scheduling) map is represented by a generator with three (resp., six) states and three (resp., six) transitions. As shown in Fig. 5.19b, no preemption

Fig. 5.18 Manufacturing cell

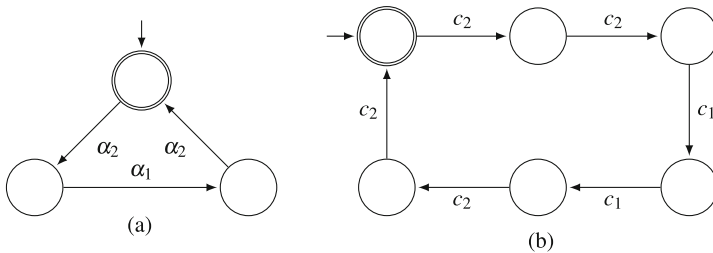
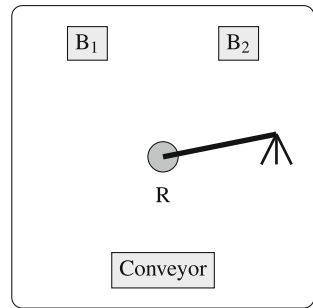


Fig. 5.19 FP scheduling. (a) Release map. (b) Scheduling map

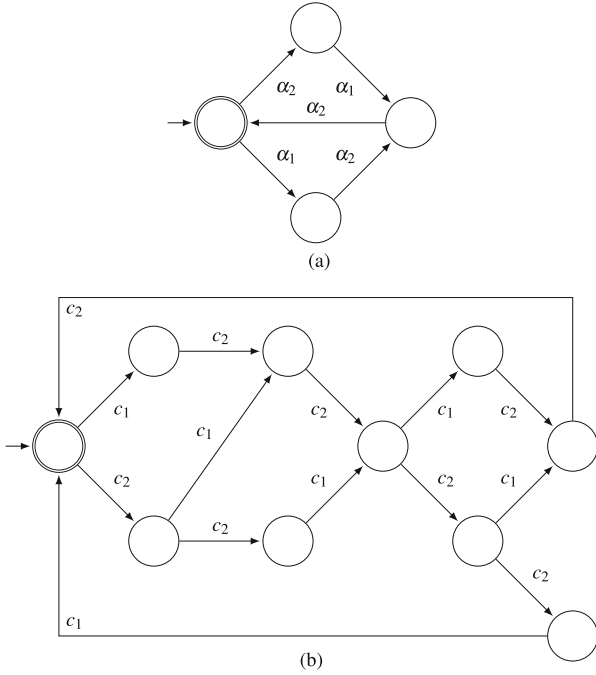


Fig. 5.20 Preemptive scheduling. (a) Release map. (b) Scheduling map

scheduling sequences are found. This requires that the robot R should transport the workpieces in the following order periodically:

$$W2, W2, W1, W1, W2, W2.$$

Suppose that the preemption matrix is

$$A_2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

i.e., the real-time scheduling is preemptive. This means that the robot can transport $W1$ and $W2$ in any order. SCT is used to calculate the supervisor, and it is represented by a DES with 37 states and 55 transitions. As shown in Fig. 5.20, the release map (resp., scheduling map) is represented by a generator with four (resp., ten) states and five (resp., 14) transitions. The scheduling map in Fig. 5.20b provides nine safe execution sequences.

If we also require that, based on A_2 , the WCRT of task τ_1 be $\mathbb{W}_1 = 5$, i.e., the two pieces of $W1$ must be transported in the first 5 seconds after their release, then we obtain a supervisor that is represented by a DES with 63 states and 91 transitions.

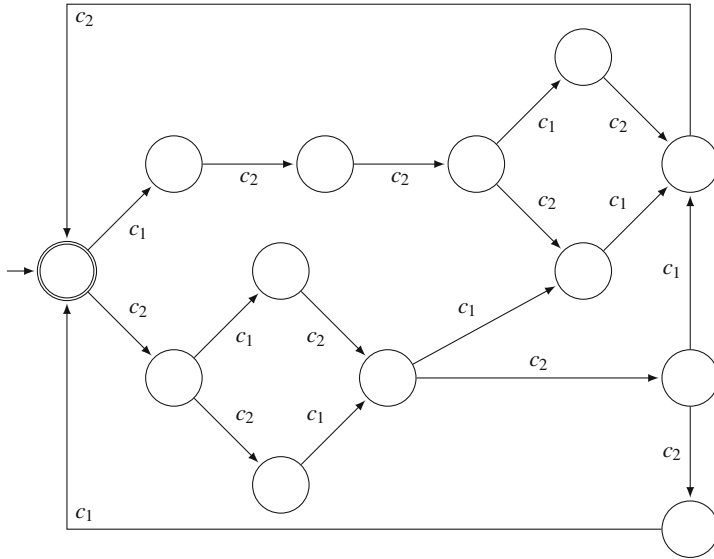


Fig. 5.21 Preemptive scheduling with $W_1 = 5$

The release map is isomorphic with the map depicted in Fig. 5.20a; the scheduling map is visualized in Fig. 5.21.

5.7 Conclusion

This chapter reports a formal constructive method for real-time periodic tasks via a DES model. The behavior of a processor can be established by the synchronous product of the DES models of all tasks running in it. The tasks can be scheduled without considering their priorities. This chapter presents two sets of conditional-preemption specifications, i.e., PFCP specifications, and task-centered conditional-preemption specifications. Moreover, in order to control the system to be nonblocking and also limit the WCRT of the tasks, two corresponding sets of specifications are presented. The formal supervisory control of DES can be considered as a rigorous analysis and synthesis tool to schedule the RTS satisfying the deadlines. The procedure **sync** in TCT is utilized to generate the plant and global specifications. By utilizing the procedure **supcon**, all the conditionally-preemptive safe execution sequences can be calculated. These sequences can provide more choices than the classical scheduling algorithms and the real-time scheduling proposed in Chap. 4 based on the supervisory control of TDES. The offline scheduling algorithm presented in this chapter can be applied to a practical context to schedule the real-world uni-processor systems.

The multi-period proposed in Chap. 4 aims to reconfigure the RTS when no safe execution sequences can be found. Based on this idea, by considering the exact execution time of real-time tasks, a general modular DES model is presented in Chap. 6. Finally, a hierarchical RTS model is presented in Chap. 7. Based on nonblocking supervisory control of state-tree structures [13, 14], both conditionally-preemptive and dynamic priority scheduling are addressed in the SCT-based real-time scheduling. Moreover, the three-step speeding up approach stated in Chap. 1 can be applied to this chapter.

Appendix

In this chapter, *regular languages* are utilized to describe the processor behavior related to any periodic *real-time task execution*. Thereafter, each language will be followed by a DES generator representation. For any task $\tau_i \in \mathbb{S}$ arriving periodically, its execution is represented by a DES generator \mathbf{G}_i with marked language $L_m(\mathbf{G}_i)$ and (*prefix*) *closed language* $L(\mathbf{G}_i)$ satisfying

$$L(\mathbf{G}_i) = \overline{L_m(\mathbf{G}_i)} \quad (5.2)$$

that describes all possible executions and random preemptions of task τ_i .

For any execution event $\sigma \in \Sigma_e$, the occurrence of σ takes a single processor time unit. The marked language $L_m(\mathbf{G}_i)$ describes all the possible execution sequences of task τ_i 's execution within a period T_i . We have

$$L_m(\mathbf{G}_i) = L_i^R(\gamma_i L_i^T)^*. \quad (5.3)$$

This expression contains two parts:

- L_i^R : the processor behavior before the first time release of task τ_i , and
- L_i^T : the processor behavior within a period T_i between the occurrence of two adjacent γ_i 's.

The *event occurrences* within a string s_r in L_i^R take R_i time units in total, i.e.,

$$L_i^R = \{s_r \mid |s_r| = R_i\}. \quad (5.4)$$

If τ_i is associated with $R_i = 0$, then L_i^R is empty. Otherwise, L_i^R represents that the processor could be idle or taken by other tasks before the first time release of task τ_i . For any $\sigma \in \Sigma_i$, let $\#\sigma(s_r)$ represent the number of occurrences of σ in a string s_r . We have

$$L_i^R = \{s_r \mid \#\sigma(s_r) + \sum_{j=1, j \neq i}^n \#c_j(s_r) = R_i\}. \quad (5.5)$$

In a period T_i , events α_i and β_i must occur only once. Since their occurrences are instantaneous (taking no time), and the events occurrences within $s \in L_i^T$ take T_i time units in total, the length of every string in a period equals $T_i + 2$. Formally,

$$L_i^T = \{s \mid |s| = T_i + 2\}. \quad (5.6)$$

For any $s \in L_i^T$ representing the complete execution of task τ_i , the number of c_i 's is C_i , i.e.,

$$\#c_i(s) = C_i. \quad (5.7)$$

In a period T_i , i.e., in L_i^T , the processor runs T_i processor time units that are occupied by all the execution events. Formally, s satisfies

$$\#l(s) + \sum_{j=1}^n \#c_j(s) = T_i. \quad (5.8)$$

Any string $s \in \gamma_i L_i^T$ contains a substring $s' = \gamma_i s^e \beta_i$, in which s^e represents the system behavior since the arrival of τ_i until its execution is completed. Thus the response time of task τ_i is the processor time spent between the occurrences of γ_i and β_i . Since the occurrence of α_i is instantaneous, the response time of τ_i in s is

$$\mathbb{P}_i = |s^e| - 1. \quad (5.9)$$

The set of strings, $S_1 \subseteq (\Sigma_e - \{l, c_i\})^*$, occurring earlier than α_i is utilized to represent the busy time. Moreover, the preemption time, occurring between α_i and β_i randomly, is represented by a set of strings $S_2 \subseteq (\Sigma_e - \{l, c_i\})^*$. Furthermore, a set of strings $S_3 \subseteq (\Sigma_e - \{c_i\})^*$ that occur later than β_i , is utilized to represent the free time. Consequently, for any $s \in L_i^T$, s is structurally represented by

$$s = s_1 \alpha_i (s_2 c_i s_2) \dots (s_2 c_i) \beta_i s_3 \quad (5.10)$$

with $s_1 \in S_1$, $s_2 \in S_2$, and $s_3 \in S_3$. The strings in $(\Sigma_e - \{l, c_i\})^*$ represent the system behavior corresponding to the random preemption by other tasks. String s_2 occurs between α_i and c_i or any two adjacent c_i 's and represents that the execution of τ_i can be preempted at any time. After the occurrence of the last c_i , β_i occurs immediately in order not to delay the response time.

The strings in $(\Sigma_e - \{c_i\})^*$ represent the system behavior in the free time. These processor time units can be idle, or utilized to execute other tasks. In order to satisfy the deadline D_i , all the c_i 's must occur before β_i . Thus, before the occurrence of β_i , the preemption time cannot be longer than $D_i - C_i$. Formally, strings s_1 and s_2 in Eq. (5.10) form sublanguages S_1 and S_2 that also satisfy Eqs. (5.11) and (5.12) listed below.

$$S_1 = \{s_1 | 0 \leq \sum_{j=1, j \neq i}^n \#c_j(s_1) \leq D_i - C_i\} \quad (5.11)$$

$$S_2 = \{s_2 | 0 \leq \sum_{j=1, j \neq i}^n \#c_j(s_2) \leq D_i - C_i\} \quad (5.12)$$

By Eq. (5.8), the free time cannot be longer than $T_i - C_i$. Formally, sublanguage S_3 in Eq. (5.10) must also satisfy

$$S_3 = \{s_3 | 0 \leq \#l(s_3) + \sum_{j=1, j \neq i}^n \#c_j(s_3) \leq T_i - C_i\}. \quad (5.13)$$

Example The closed and marked languages to describe the processor behavior to execute task τ_1 are respectively

$$L(\mathbf{G}_1) = \overline{L_m(\mathbf{G}_1)}$$

and

$$L_m(\mathbf{G}_1) = L_1^R(\gamma_1 L_1^T)^*.$$

Sublanguage L_1^T satisfies

$$L_1^R = \epsilon, \forall s \in L_1^T, |s| = 8, \#c_1(s) = 2, \text{ and}$$

$$\#l(s) + \sum_{j=1}^3 \#c_j(s) = 6.$$

String s is structured as

$$s = s_1 \alpha_1 s_2 c_1 s_2 c_1 \beta_1 s_3$$

with $s_1 \in S_1$, $s_2 \in S_2$, and $s_3 \in S_3$ as follows:

- $S_1 = \{s_1 \in \{c_2, c_3\}^* | 0 \leq \sum_{j=2}^3 \#c_j(s_1) \leq 4\}$,
- $S_2 = \{s_2 \in \{c_2, c_3\}^* | 0 \leq \sum_{j=2}^3 \#c_j(s_2) \leq 4\}$, and
- $S_3 = \{s_3 \in \{c_2, c_3, l\}^* | 0 \leq \#l(s_3) + \sum_{j=2}^3 \#c_j(s_3) \leq 4\}$. □

References

1. Baruah, S.K., Rosier, L.E., Howell, R.R.: Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Syst.* **2**(4), 301–324 (1990)
2. Buttazzo, G.C., Bertogna, M., Yao, G.: Limited preemptive scheduling for real-time systems: a survey. *IEEE Trans. Indust. Inf.* **9**(1), 3–15 (2013)
3. Chen, P.C.Y., Wonham, W.M.: Real-time supervisory control of a processor for non-preemptive execution of periodic tasks. *Real-Time Syst.* **23**, 183–208 (2002)
4. Davis, R.I.: A review of fixed priority and EDF scheduling for hard real-time uniprocessor systems. *ACM SIGBED Rev.* **11**(1), 8–19 (2014)
5. Dertouzos, M.L.: Control robotics: the procedural control of physical processes. In: IF IP Congress, pp. 807–813 (1974)
6. Fineberg, M.S., Serlin, O.: Multiprogramming for hybrid computation. In: AFIPS Fall Joint Computing Conference, pp. 1–13 (1967)
7. Howell, R.R., Venkatrao, M.K.: On non-preemptive scheduling of recurring tasks using inserted idle times. *Inf. Comput.* **117**(1), 50–62 (1995)
8. Krishna, C.M.: Real-time systems. Wiley Encyclopedia of Electrical and Electronics Engineering. McGraw-Hill, New York (2001)
9. Leung, J.Y.T., Whitehead, J.: On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Perform. Eval.* **2**(4), 237–250 (1982)
10. Li, J., Shu, L., Chen, J., Li, G.: Energy-efficient scheduling in nonpreemptive systems with real-time constraints. *IEEE Trans. Syst. Man Cybern. Syst.* **43**(2), 332–344 (2013)
11. Li, D., Li, M., Meng, X., Tian, Y.: A hyperheuristic approach for intercell scheduling with single processing machines and batch processing machines. *IEEE Trans. Syst. Man Cybern. Syst.* **45**(2), 315–325 (2015)
12. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM* **20**(1), 46–61 (1973)
13. Ma, C., Wonham, W.M.: Nonblocking Supervisory Control of State Tree Structures, vol. 317. Springer-Verlag, Berlin (2005)
14. Ma, C., Wonham, W.M.: Nonblocking supervisory control of state tree structures. *IEEE Trans. Autom. Control* **51**(5), 782–793 (2006)
15. Mok, A.K.: Fundamental design problems of distributed systems for the hard-real-time environment. Ph.D. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, (1983)
16. Sha, L., Abdelzaher, T., Cervin, A., Baker, T., Burns, A., Buttazzo, G., Caccamo, M., Lehoczky, J., Mok, A.K.: Real time scheduling theory: a historical perspective. *Real-Time Syst.* **28**(2), 101–155 (2004)
17. Singhoff, F., Legrand, J., Nana, L., Marcé, L.: Cheddar: a flexible real time scheduling framework. *ACM SIGAda Ada Lett.* **4**, 1–8 (2004)
18. Wang, X., Li, Z., Wonham, W.M.: Dynamic multiple-period reconfiguration of real-time scheduling based on timed DES supervisory control. *IEEE Trans. Indust. Inf.* **12**(1), 101–111 (2016)
19. Wang, X., Li, Z., Wonham, W.M.: Optimal priority-free conditionally-preemptive real-time scheduling of periodic tasks based on DES supervisory control. *IEEE Trans. Syst. Man Cybern. Syst.* **47**(7), 1082–1098 (2017)
20. Wonham, W.M., Cai, K.: Supervisory Control of Discrete-Event Systems. Monograph Series Communications and Control Engineering. Springer, Berlin (2018)
21. Xia, Y., Zhou, M., Luo, X., Pang, S., Zhu, Q.: A stochastic approach to analysis of energy-aware DVS-enabled cloud datacenters. *IEEE Trans. Syst. Man Cyber. Syst.* **45**(1), 73–83 (2015)

Chapter 6

Modular Scheduling/Reconfiguration with Exact Execution Time Based on R-W Method



6.1 Introduction

Supervisory control theory (SCT)-based real-time scheduling and reconfiguration [3, 6, 13, 16, 17, 19] are a newly-identified research topic. As summarized in Chap. 4, the study in [3] proposes a timed discrete-event systems (TDES) model; based on SCT, periodic real-time systems (RTS) are scheduled non-preemptively. In [3], the RTS modelled by TDES assume that the processors are *resources* that are available to execute several real-time tasks concurrently. Thereafter, the TDES-based real-time scheduling approach is extended to:

- schedule the RTS preemptively or non-preemptively [6],
- schedule the RTS processing *sporadic tasks* [13], and
- dynamically reconfigure the periodic RTS when no safe execution sequences are found [16].

The TDES-based non-preemptive scheduling is studied in [3, 6, 13] and [16], and the discrete-event systems (DES)-based sequential RTS scheduling is investigated in [17], which are reported in Chap. 5. By assigning different execution events for different tasks, the RTS model proposed in Chap. 5 is utilized to describe the sequential executions of *independent tasks*. The RTS models presented in Chap. 5 are more realistic.

Based on a modular modelling approach, this chapter deals with RTS processing sporadic and/or *multi-period tasks* simultaneously. A sporadic task running in an RTS may have an *irregular arrival time*, with or without a *deadline*; a periodic task has a *regular arrival time* and a deadline. The period of a periodic task could be:

- equal to the corresponding deadline [10],
- greater than or equal to the deadline [12], or
- multiple [16].

According to the study in [2], the execution instance of a real-time task is referred to as a *job*. In real world, the execution time of a job always varies over time. In order to describe such variations, the lower bounds and upper bounds of such a task are estimated, which are referred to as its *best-case execution time* (BCET) and *worst-case execution time* (WCET), respectively. The *exact execution time* of a task lies between its BCET and WCET.

Based on DES, an RTS can be built via a three-step approach:

- the *parameters* of the tasks are represented by modular DES models,
- the *task behavior* is constrained by the necessary modular DES models, and
- the global RTS model is constrained by the DES real-time task models corresponding to the running tasks.

A scheduling principle, namely *priority-free conditionally-preemptive (PFCP) real-time scheduling*, is presented in Chap. 5 by considering that both preemptive and non-preemptive scheduling policies are conservative. Based on the PFCP real-time scheduling principle, all the safe execution sequences in an RTS processing both multi-period and sporadic tasks can be found. The proposed modelling framework and the PFCP scheduling principle are general in the following respects:

- the task behavior is modelled by DES, and thus the processing of different tasks is represented by different events instead of a unique time event,
- the priorities of tasks are not treated and the preemption relations are described by a matrix, and
- the specifications are imposed on both processor and task levels.

Based on the presented modelling framework and scheduling principle, classic *fixed priorities* (FP) real-time scheduling policies [1, 4, 5, 7–11, 14, 21] can be considered as consequences of the presented specifications.

For the purpose of integrating real-time scheduling and reconfiguration in one general model, a DES-based multi-period model is presented in this chapter. The only difference of a period's model before and after its reconfiguration is the upper bound of its multi-period. A DES model depicting the RTS is synchronized by the DES representing these tasks. The related details and examples for dynamic reconfigurations of RTS can be found in Chap. 4. For the RTS tasks with multi-periods, they are endowed with the following features:

- a task arrives periodically but the length of its period is selected randomly between a lower bound and an upper bound, and
- a task is associated with or without a deadline that is not earlier than the WCET and no later than the longest period.

Two approaches are presented in this chapter to obtain the local behavior of any real-time task. Users are free to choose any one at will. On the one hand, given a real-time task, one approach views its deadline as a parameter that is used to compute the synchronous product of the necessary modular generators. On the other hand, by viewing the deadline of a real-time task as a timing specification, the local behavior of a real-time task is synthesized as a supervisor. In other words, the latter applies

a *formal specification* to enforce the deadlines in order to obtain local supervisors, one for each individual task. Afterwards, it is ready to construct a candidate for the RTS overall behavior by the synchronous product of all real-time task modules. This candidate may be blocking and does not yet implement a PFCP scheduling policy. Both issues are then resolved by a global supervisor that we synthesize at the second stage. The presented RTS model is implemented in a manufacturing example.

The rest of this chapter is structured as follows. The presented RTS model is discussed in Sect. 6.2. Two approaches are detailed in Sect. 6.3 to obtain the global RTS execution models based on the modular RTS models. The developed formal specifications and supervisor synthesis are reported in Sect. 6.4. The PFCP is applied to a real-world RTS in Sect. 6.5. Conclusions and possible extensions are given in Sect. 6.6.

6.2 Modular RTS Models

A DES-based periodic real-time task model is proposed in Chap. 5. By the supervisory control of DES, all the safe execution sequences of an RTS are found by PFCP real-time scheduling. However, this model is conservative:

- the model can only describe periodic tasks,
- the model cannot be reconfigured dynamically when no safe execution sequences can be found, and
- the model is monolithic (global), i.e., from the perspective of an RTS task, all the possible processor behaviors are enumerated in a unique DES *generator*. One must rebuild a monolithic DES generator whenever the parameters of the task are edited by the user.

Motivated by these drawbacks, this chapter presents a formal and unified modular DES-based real-time task model, in which DES generators represent the parameters of a sporadic or multi-period task. The corresponding monolithic task model is the synchronous product of these DES models.

A DES plant is a *generator*

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

where

- Q is the finite *state set*,
- Σ is the finite *event set (alphabet)*,
- $\delta : Q \times \Sigma \rightarrow Q$ is the *partial state transition function*,
- q_0 is the *initial state*, and
- $Q_m \subseteq Q$ is the subset of *marked states*.

6.2.1 RTS Tasks

In accordance with [15–18] and [2], an RTS is assumed to process a set of *non-repetitive*, sporadic, and/or *multi-period tasks*. All the real-time tasks run in an RTS simultaneously and the unpredictable exact execution time of a task falls between its BCET and WCET. For a periodic task, its *arrival time* is regular, which is considered as a period that could be:

- equal to its deadline [10],
- greater than or equal to its deadline [12], or
- multiple [16, 18], i.e., a period set that has a lower bound and an upper bound.

A non-repetitive task arrives only once, and a sporadic or periodic task arrives periodically. A non-repetitive or sporadic task may arrive at any time, which means that no periodical constraint is assigned. As defined in Chap. 4, a multi-period is a period set that has a lower bound and an upper bound. Any period of a real-time task, i.e., the difference between any two adjacent arrivals, lies between the two bounds. If the lower bound is equal to the upper bound, the task is a traditional periodic one. We consider a traditional periodic task as a special case of a multi-period. Consequently, our analysis is based on multi-period tasks. Unless otherwise stated, in the rest of this chapter, “period task” means “*multi-period task*”.

An RTS \mathbb{S} is assumed to process a set of real-time tasks. Formally, write $\mathbb{S} = \{\tau_1, \tau_2, \dots, \tau_i, \dots, \tau_n\}$. For $i \in \mathbf{n} = \{1, 2, \dots, n\}$, a periodic task $\tau_i \in \mathbb{S}$ is represented by a four-tuple

$$\tau_i = (R_i, \mathbf{C}_i, D_i, \mathbf{T}_i)$$

with

- a *release time* R_i ,
- an *execution time interval* \mathbf{C}_i ,
- a *deadline* D_i , and
- a *multi-period* \mathbf{T}_i ,

where R_i and D_i are non-negative integers. As stated in Chap. 4, a multi-period

$$\mathbf{T}_i = [T_i^l, T_i^u] \in \mathbb{N} \times \mathbb{N},$$

is specified by a non-empty interval where the number of time units that elapse between any two successive releases lies within \mathbf{T}_i . Hence a multi-period has a *lower bound* (i.e., shortest one) represented by T_i^l and an *upper bound* (i.e., longest one) represented by T_i^u .

Only an exact execution time C (resp., period T) in \mathbf{C}_i (resp., \mathbf{T}_i) of task τ_i is actually taken. For deadline $D_i \in (T_i^l, T_i^u]$, we impose an additional assumption that the task can arrive only when the previously instantiated job is completed. The

lower (resp., upper) bound of the execution interval \mathbf{C}_i is the BCET (resp., WCET) of task τ_i , denoted by C_i^l (resp., C_i^u), i.e.,

$$\mathbf{C}_i = [C_i^l, C_i^u] \in \mathbb{N} \times \mathbb{N}.$$

For simplification, we write C_i (resp., T_i , in accordance with Chap. 5) instead of \mathbf{C}_i (resp., \mathbf{T}_i) in the case of $C_i^l = C_i^u$ (resp., $T_i^l = T_i^u$).

Any execution period of a real-time task, i.e., the difference between any two adjacent arrivals, lies between T_i^l and T_i^u . For deadline $D_i \in (T_i^l, T_i^u]$, we impose an additional assumption that the task can arrive only when the previously instantiated job is completed. An RTS \mathbb{S} is an asynchronous task set if there exist at least two different tasks τ_i and τ_j in \mathbb{S} with $R_i \neq R_j$; otherwise \mathbb{S} is synchronous.

The following different types of real-time tasks are addressed in this chapter:

- a sporadic/non-repetitive task without a deadline: $\tau_i = (\mathbf{C}_i)$,
- a sporadic/non-repetitive task with a deadline: $\tau_i = (\mathbf{C}_i, D_i)$,
- a periodic task with its deadline not equal to T_i^u : $\tau_i = (R_i, \mathbf{C}_i, D_i, \mathbf{T}_i)$, and
- a periodic task with its deadline equal to T_i^u : $\tau_i = (R_i, \mathbf{C}_i, \mathbf{T}_i)$.

A periodic or sporadic task produces an infinite sequence of *jobs* that arrive repetitively. And a non-repetitive task produces only one job. As stated in Chap. 3, a periodic task τ_i consists of an infinite sequence of jobs repeated periodically that are represented by a corresponding four-tuple

$$J_{i,j} = (r_{i,j}, C_i, d_{i,j}, p_{i,j}).$$

The subscript “ i, j ” of $J_{i,j}$ represents the j -th execution of task τ_i .

Leaving out the selected scheduling policies, when the real-time tasks are under execution, the processing of each individual job (belong to a real-time task) falls into the following two categories:

- *preemptive*: a running task can be interrupted by the execution of other released tasks;
- *non-preemptive*: the execution of a running task cannot be interrupted.

Based on Chap. 5, the *alphabet* (set of *event labels*) Σ_i describing the processor’s behavior to execute task τ_i is:

- γ_i : task τ_i is *released*,
- α_i : the execution of τ_i is *started*,
- β_i : the execution of τ_i is *completed*,
- ρ_i : the execution of τ_i is not completed,
- c_i ($i \in \mathbf{n}$): the processor starts to execute τ_i for one *processor time unit*, and
- l : *empty action*, i.e., the processor is in an idle operation for one time unit.

Let $\sigma = c_i$ (resp., $\sigma = l$). The occurrence of σ represents that one processor time unit is utilized to execute task τ_i (resp., in an idle operation for one processor time unit while it stays in state q').

Formally, for an RTS, its event set is defined as

$$\Sigma = \Sigma_{con} \dot{\cup} \Sigma_{unc},$$

with

- $\Sigma_{con} = \{\alpha_i, c_i | i \in \mathbf{n}\}$: the *controllable event subset*, and
- $\Sigma_{unc} = \{\beta_i, \rho_i, \gamma_i, l | i \in \mathbf{n}\}$: the *uncontrollable event subset*.

Moreover, Σ is also partitioned into

- $\Sigma_o = \{\gamma_i, \alpha_i, \beta_i, \rho_i, | i \in \mathbf{n}\}$: the *operation event set*, and
- $\Sigma_e = \{c_i, l | i \in \mathbf{n}\}$: the *execution event set*.

6.2.2 Periodic/Sporadic Task Execution Time Models

An RTS \mathbb{S} is assumed to process n sporadic/periodic/non-repetitive tasks. A sporadic (resp., periodic) task τ_i describes an *infinite stream* of jobs arriving at irregular (resp., regular) time intervals. Suppose that n sporadic/periodic tasks are running in an RTS. We present a DES model for \mathbf{C}_i of periodic and sporadic tasks. Let $i \in \mathbf{n}$ and $\tau_i \in \mathbb{S}$. This model mainly shows the processor behavior while it is executing task τ_i . Naturally, before the execution of τ_i is started and after its execution is finished, the processor is allowed to be in an idle operation. The execution time \mathbf{C}_i of a sporadic or periodic task is described by a DES generator

$$\mathbf{G}_i^{\mathbf{C}} = (Q_i^{\mathbf{C}}, \Sigma_i^{\mathbf{C}}, \delta_i^{\mathbf{C}}, q_{0,i}^{\mathbf{C}}, Q_{m,i}^{\mathbf{C}}).$$

Suppose that $s \in \Sigma_i^{\mathbf{C}*}$ is a string over the event set $\Sigma_i^{\mathbf{C}}$. Then we have s^* representing $\epsilon + s + s^2 + \dots$. Let C_i^l and C_i^u represent the lower bound (BCET) and upper bound (WCET) of task τ_i , respectively. The possible execution sequences of task τ_i are described by the *marked language* $L_m(\mathbf{G}_i^{\mathbf{C}})$ over $\Sigma_i^{\mathbf{C}}$.

This process can be represented as a regular expression

$$L_i^{\mathbf{C}} := L_l(L_a L_l)^* \subseteq \Sigma_i^{\mathbf{C}*} \quad (6.1)$$

with

- $L_l = l^*$, and
- $L_a = \sum_{k=C_i^l}^{C_i^u} \gamma_i \alpha_i c_i^{C_i^l} (\rho_i c_i)^{k-C_i^l} \beta_i$.

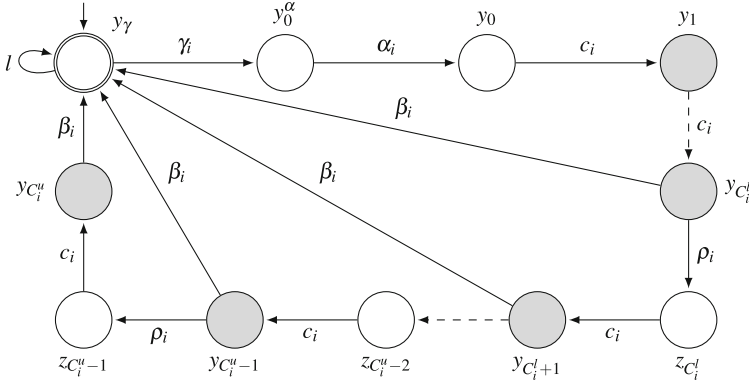
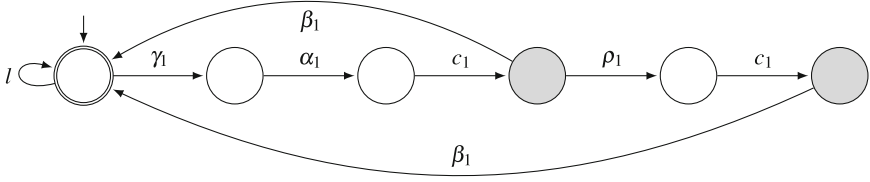
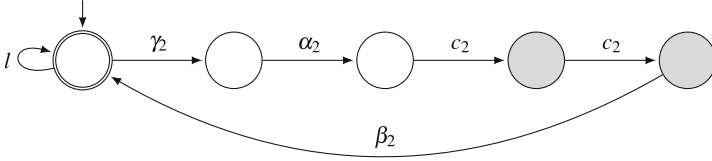


Fig. 6.1 Execution model for sporadic or periodic tasks

In the above formula L_a represents that the execution of a job takes exactly $k \in [C_i^l, C_i^u]$ time units. We provide a realization of L_i^C as the marked behavior of a DES \mathbf{G}_i^C , i.e., we have $L_m(\mathbf{G}_i^C) = L_i^C$ for the DES illustrated in Fig. 6.1.

Technically, \mathbf{G}_i^C is defined over the alphabet $\Sigma_i^C = \{\alpha_i, c_i, \beta_i, \rho_i, \gamma_i\}$ with

- state y^γ : before the arrival of task τ_i ,
- state y_0^α : the processor is ready to process τ_i ,
- state y_0 : the execution of τ_i is *started*,
- state $(0 < k \leq C_i^u)$ y_k : τ_i has been executed for k time units,
- state $(C_i^l \leq k < C_i^u)$ z_k : the execution of τ_i is not completed after being executed for k time units,
- the state with an *entering arrow* is the *initial state* y^γ ,
- the state set $\{y^\gamma\}$ is the singleton *marker state set* represented by a *double circle*, and
- the transition function δ_i^C satisfies
 - $\delta_i^C(y^\gamma, l) = y^\gamma$: the processor is in an idle operation,
 - $\delta_i^C(y^\gamma, \gamma_i) = y_0^\alpha$: τ_i is *released*,
 - $\delta_i^C(y_0^\alpha, \alpha_i) = y_0$: the processor starts to execute τ_i ,
 - $(0 \leq k < C_i^l)$ $\delta_i^C(y_k, c_i) = y_{k+1}$: τ_i is being executed for the $(k + 1)$ -th time unit,
 - $(C_i^l \leq k < C_i^u)$ $\delta_i^C(y_k, \rho_i) = z_k$: the execution of τ_i is not *completed* in the k -th time unit,
 - $(C_i^l \leq k < C_i^u)$ $\delta_i^C(z_k, c_i) = y_{k+1}$: τ_i is being executed for the $(k + 1)$ -th time unit, and
 - $(C_i^l \leq k \leq C_i^u)$ $\delta_i^C(y_k, \beta_i) = y^\gamma$: the execution of τ_i is *completed* in k processor time units.

Fig. 6.2 Generator G_1^C Fig. 6.3 Generator G_2^C

Example Suppose that there are two tasks τ_1 and τ_2 running in a uni-processor RTS, i.e., $\mathbb{S} = \{\tau_1, \tau_2\}$. Let $\mathbf{C}_1 = [1, 2]$. Then G_1^C is shown in Fig. 6.2. Let $C_2^l = C_2^u = 2$. Then G_2^C is shown in Fig. 6.3. \square

6.2.3 Non-Repetitive Execution Time Models

The task execution model for non-repetitive tasks is the counterpart of the task execution model for periodic and sporadic tasks. The only difference is that the execution process is non-repetitive.

Let $\tau_i \in \mathbb{S}$, $i \in \mathbf{n}$, denote a non-repetitive task with an execution time parameter

$$\mathbf{C}_i = [C_i^l, C_i^u]$$

to represent the lower bound and upper bound. The formal language to model task execution is then obtained as

$$L_i^{\mathbf{C},nr} := L_i^{\mathbf{C}} - \Sigma_i^* \gamma_i \Sigma_i^* \gamma_i \Sigma_i^* \subseteq \Sigma_i^* \quad (6.2)$$

with $L_i^{\mathbf{C}}$ as defined in Eq. (6.1). We provide a realization of $L_i^{\mathbf{C},nr}$ as the marked behavior of a DES $G_i^{\mathbf{C},nr}$, i.e., we have

$$L_m(G_i^{\mathbf{C},nr}) = L_i^{\mathbf{C},nr}$$

for the DES illustrated in Fig. 6.4.

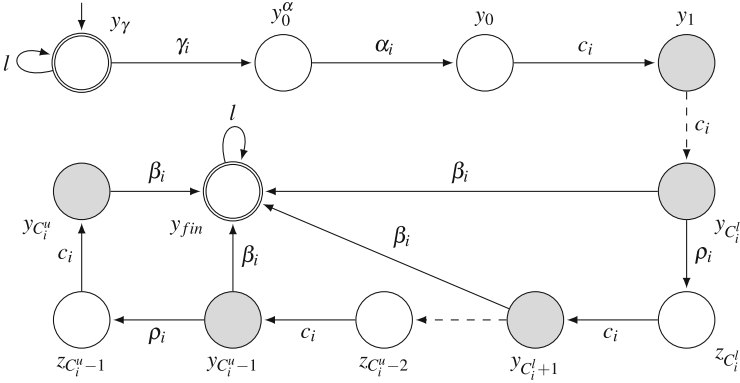


Fig. 6.4 Non-repetitive execution model

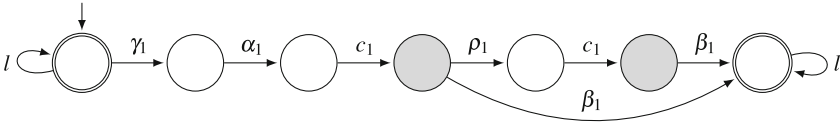


Fig. 6.5 Generator $G_1^{C,nr}$

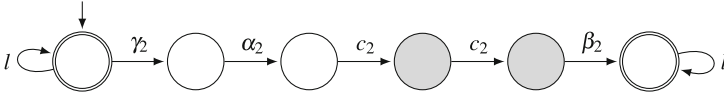


Fig. 6.6 Generator $G_2^{C,nr}$

Technically, $G_i^{C,nr}$ is defined exactly as G_i^C , except that

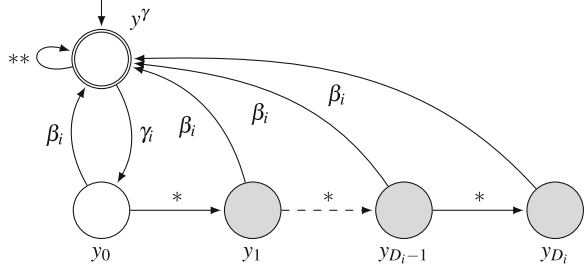
- there is an additional marked state y_{fin} with a self-loop $\delta_i^C(y_{fin}, l) = y_{fin}$, and
- any transition $\delta_i^C(y_k, \beta_i) = y^\alpha$ defined in G_i^C for k with $C_i^l \leq k \leq C_i^u$, is substituted by a transition $\delta_i^C(y_k, \beta_i) = y_{fin}$ in $G_i^{C,nr}$.

Example Let $C_1 = [1, 2]$. Then $G_1^{C,nr}$ is shown in Fig. 6.5. Let $C_2^l = C_2^u = 2$. Then $G_2^{C,nr}$ is shown in Fig. 6.6. \square

6.2.4 Deadline Models

Generally, for any periodic or sporadic real-time task τ_i , $C_i^h \leq D_i$ is assigned. Let $C_i^l \leq C \leq C_i^h$. The execution of τ_i in an execution period takes C processor time units. This process must be completed with the deadline D_i satisfied. This means

Fig. 6.7 Deadline model



that during the time interval between the occurrence of events γ_i and β_i , C time units are utilized to process τ_i ; the other (at most) $D_i - C$ time units are in idle operations or occupied by the execution of other tasks. For a periodic or sporadic task τ_i with a deadline D_i , the deadline D_i is represented by a DES generator \mathbf{G}_i^D that recognizes the following marked language

$$L_i^D = L_f(\gamma_i L_d \beta_i L_f)^* \subseteq (\Sigma_e \cup \{\gamma_i, \beta_i\})^* \quad (6.3)$$

with

- $L_f = \Sigma_e^*$, and
- $L_d = \{s | s \in (\Sigma_e - \{l\})^* \ \& \ |s| \leq D_i\}$.

Language L_f represents that, prior to the release of task τ_i , an arbitrary amount of time units may elapse. Furthermore, L_d represents that after a release of task τ_i (the occurrence of event α_i), no more than D_i time units may elapse until the job completion (the occurrence of event β_i). We require that between the occurrences of α and β_i , the processor should not be in an idle operation. The release time and period are realized by the DES

$$\mathbf{G}_i^D = (Q_i^D, \Sigma_i^D, \delta_i^D, q_{0,i}^D, Q_{m,i}^D)$$

shown in Fig. 6.7, which illustrates a realization \mathbf{G}_i^D with

$$L_i^D = L_m(\mathbf{G}_i^D).$$

In Fig. 6.7, “*” and “**” represent events in $\Sigma_e - \{l\}$ and $\Sigma_e - \{c_i\}$, respectively.

Technically, \mathbf{G}_i^D is defined over the alphabet $\Sigma_i^D := \Sigma_e \cup \{\gamma_i, \beta_i\}$ with

- state y^γ : the processor is ready to execute τ_i ,
- state $(0 \leq k \leq D_i) y_k$: k time units in an execution period have elapsed,
- the state with an entering arrow is the initial state y^γ ,
- the state set $\{y^\gamma\}$ is the singleton marker state set represented by a double circle, and

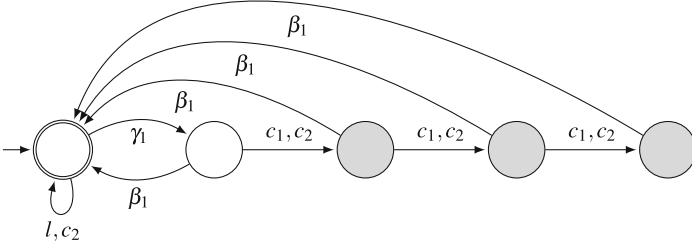


Fig. 6.8 Generator G_1^D

- let $i, j \in \mathbf{n}$, the transition function δ_i^D satisfies
 - $\delta_i^D(y^\gamma, l) = y^\gamma$: the processor is in an idle operation,
 - $(i \neq j) \delta_i^D(y^\gamma, c_j) = y^\gamma$: task τ_j is under execution,
 - $\delta_i^D(y^\gamma, \gamma_i) = y_0$: task τ_i releases,
 - $(0 \leq k < D_i) \delta_i^D(y_k, c_j) = y_{k+1}$: the $(k + 1)$ -th time unit is occupied by τ_j ,
 - and
 - $(0 \leq k \leq D_i) \delta_i^D(y_k, \beta_i) = y^\gamma$: the execution of τ_i is completed.

Example Suppose that there are two tasks τ_1 and τ_2 running in a uni-processor RTS, i.e., $\mathbb{S} = \{\tau_1, \tau_2\}$. Let $D_1 = 3$. Generator G_1^D is depicted in Fig. 6.8. □

6.2.5 Release and Multi-Period Models

In this section, *DES diagrams* are utilized to describe the release time and the periods of periodic real-time tasks. As stated previously, an RTS is assumed to process sporadic/periodic/non-repetitive tasks. In the release time and a period, the processor time units can be occupied by all the tasks running in the RTS or in an idle operation.

As defined in [16], a period of task τ_i is a period set that has a lower bound and an upper bound. Formally, $\mathbf{T}_i = [T_i^l, T_i^u]$. Let T be an arbitrary period, i.e., $T_i^l \leq T \leq T_i^u$. Let C be an exact execution time, i.e., $C_i^l \leq C \leq C_i^u$. Generally, we have $C_i^u \leq T_i^l$. In a period, after task τ_i is released, the processor will use C time units to execute τ_i . Moreover, in the other $T - C$ time units, the processor is in an idle operation or occupied by the execution of other tasks. For task τ_i , its release time and period are described by a DES generator G_i^T . Formally, we have its marked language as

$$L_i^{\mathbf{T}} = \overline{L_r} + L_r(\gamma_i L_t)^* \subseteq (\Sigma_e \cup \{\gamma_i\})^* \tag{6.4}$$

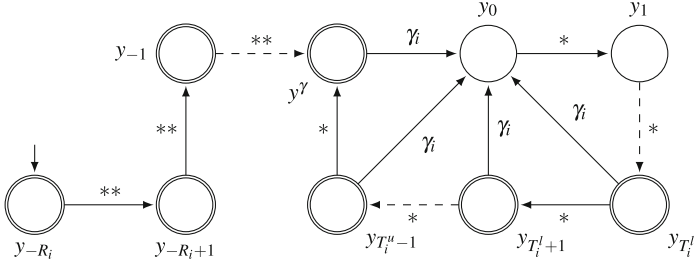


Fig. 6.9 Release time and period model

with

- $L_r := \{s | s \in (\Sigma_e - \{c_i\})^* \ \& \ |s| = R_i\}$, and
- $L_t := \{s | s \in \Sigma_e^* \ \& \ T_i^l \leq |s| \leq T_i^u\}$.

Language L_r represents that, prior to the first release of τ_i , the R_i processor time units are allowed to be in an idle operation or alternatively occupied to execute other tasks. Furthermore, L_t represents that in a processing period of τ_i , the processor can be in an idle operation or alternatively occupied by the execution of any task in \mathbb{S} . The execution of task τ_i takes no less than T_i^l time units and no more than T_i^u time units.

The release time and period are realized by the DES

$$\mathbf{G}_i^{\mathbf{T}} = (Q_i^{\mathbf{T}}, \Sigma_i^{\mathbf{T}}, \delta_i^{\mathbf{T}}, q_{0,i}^{\mathbf{T}}, Q_{m,i}^{\mathbf{T}})$$

as illustrated by Fig. 6.9, where “*” and “**” represent the events in Σ_e and $\Sigma_e - \{c_i\}$, respectively, and we have

$$L_i^{\mathbf{T}} = L_m(\mathbf{G}_i^{\mathbf{T}}).$$

Technically, $\mathbf{G}_i^{\mathbf{T}}$ is defined over the alphabet $\Sigma_i^{\mathbf{T}} := \Sigma_e \cup \{\gamma_i\}$ with

- state y_{-k} , $0 < k \leq R_i$: k time units until the first release of τ_i ,
- state y^γ : the processor is ready to execute τ_i ,
- state $(0 \leq k < T_i^u)$ y_k : k time units in a period have elapsed,
- the states represented by double circles are the marker states,
- the state with an entering arrow is the initial state y_{-R_i} ,
- the state set $\{y_{-R_i}, y_{-R_i+1}, \dots, y_{-1}, y^\gamma, T_i^l, \dots, T_i^u\}$ is the marker state set, in which each state is represented by a double circle, and
- let $i, j \in \mathbf{n}$, the transition function $\delta_i^{\mathbf{T}}$ satisfies
 - $(1 < k \leq R_i, i \neq j) \delta_i^{\mathbf{T}}(y_{-k}, c_j) = y_{-k+1}$: the processor is occupied by τ_j ,
 - $(1 < k \leq R_i) \delta_i^{\mathbf{T}}(y_{-k}, l) = y_{-k+1}$: the processor is in an idle operation,
 - $(i \neq j) \delta_i^{\mathbf{T}}(y_{-1}, c_j) = y^\gamma$: the processor is occupied by τ_j ,

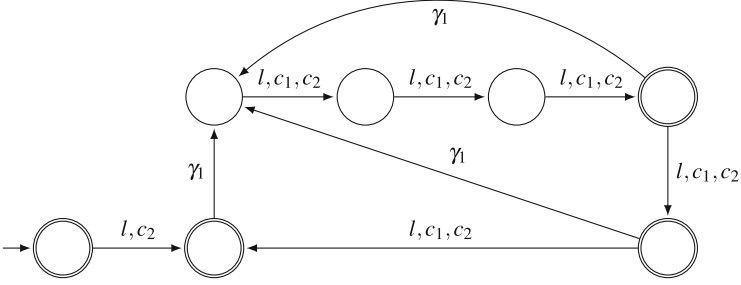


Fig. 6.10 Generator G_1^T

- $(i \neq j)\delta_i^T(y_{-1}, l) = y^\nu$: the processor is in an idle operation,
- $\delta_i^T(y^\nu, \gamma_i) = y_0$: τ_i is released,
- $(0 \leq k < T_i^u - 1) \delta_i^T(y_k, c_j) = y_{k+1}$: the $(k + 1)$ -th time unit is occupied by τ_j ,
- $(0 \leq k < T_i^u - 1) \delta_i^T(y_k, l) = y_{k+1}$: the $(k + 1)$ -th time unit is in an idle operation,
- $\delta_i^T(y_{T_i^u-1}, c_j) = y^\nu$: the processor is occupied by τ_j ,
- $\delta_i^T(y_{T_i^u-1}, l) = y^\nu$: the processor is in an idle operation, and
- $(T_i^l \leq k < T_i^u) \delta_i^T(y_k, \gamma_i) = y_0$: τ_i releases for the next time.

Remark The naming rule for all the states in Fig. 6.9 is similar to that for the states in Fig. 6.1. The only differences are: (1) state $(0 < k \leq R_i) y_{-k}$ represents that there is still k processor time units until the first arrival of τ_i ; (2) state $(0 \leq k < T_i^u)$ represents the k -th processor time unit in a period. \square

Example Suppose that there are two tasks τ_1 and τ_2 running in a uni-processor RTS, i.e., $\mathbb{S} = \{\tau_1, \tau_2\}$. Let $R_1 = 1$ and $\mathbf{T}_1 = [3, 5]$. Generator G_1^T for task τ_1 is depicted in Fig. 6.10. \square

6.3 Global RTS Execution Models

This section presents two approaches to obtain the global RTS execution models based on the modular RTS models presented above.

6.3.1 Approach I

Similar to the execution time models, we address the deadline model as a parameter of a real-time task, which is used to compute the synchronous product of the

necessary modular generators. Then, it is ready to construct a candidate for the RTS overall behavior by the synchronous product of all real-time task modules. This candidate may be blocking but it can be resolved by the final global supervisor synthesized later.

The global behavior of an RTS is represented by a monolithic DES generator \mathbf{G} that is the synchronous product of all the running tasks' parameters. Generator \mathbf{G}_i representing a real-time task τ_i is constrained by the synchronous product of the modular generators defined above, which falls into one of the following six categories generated by TCT¹ as follows:²

- sporadic without a (hard) deadline, $L_m(\mathbf{G}_i) = L_m(\mathbf{G}_i^C)$:

$$\mathbf{G}_i = \mathbf{G}_i^C,$$

- non-repetitive without a deadline, $L_m(\mathbf{G}_i) = L_m(\mathbf{G}_i^{C,nr})$:

$$\mathbf{G}_i = \mathbf{G}_i^{C,nr},$$

- sporadic with a deadline, $L_m(\mathbf{G}_i) = L_m(\mathbf{G}_i^C) || L_m(\mathbf{G}_i^D)$:

$$\mathbf{G}_i = \text{sync}(\mathbf{G}_i^C, \mathbf{G}_i^D),$$

- non-repetitive with a deadline, $L_m(\mathbf{G}_i) = L_m(\mathbf{G}_i^{C,nr}) || L_m(\mathbf{G}_i^D)$:

$$\mathbf{G}_i = \text{sync}(\mathbf{G}_i^{C,nr}, \mathbf{G}_i^D),$$

- periodic with a deadline $D_i < T_i^u$, $L_m(\mathbf{G}_i) = L_m(\mathbf{G}_i^C) || L_m(\mathbf{G}_i^T) || L_m(\mathbf{G}_i^D)$:

$$\mathbf{G}_i = \text{sync}(\mathbf{G}_i^C, \mathbf{G}_i^T, \mathbf{G}_i^D),$$

- periodic with a deadline $D_i = T_i^u$, $L_m(\mathbf{G}_i) = L_m(\mathbf{G}_i^C) || L_m(\mathbf{G}_i^T)$:

$$\mathbf{G}_i = \text{sync}(\mathbf{G}_i^C, \mathbf{G}_i^T).$$

The procedures utilized in this chapter are introduced in [20].

¹ <http://www.control.utoronto.ca/DES>.

² For simplification, unlike in the previous chapters, we write the generator names, say \mathbf{G}_i^C , as the input of TCT procedures directly.

Example Suppose that $\mathbb{S} = \{\tau_1, \tau_2\}$. For simplification, we assume $C_1^l = C_1^u = 2$. Several possible models for task τ_1 (assigned with different parameters) in the previous examples are displayed in Fig. 6.11, in which the corresponding parameters are listed in the captions. Accordingly, we have

- Fig. 6.11a: a sporadic task with a deadline,
- Fig. 6.11b: a periodic task with its deadline less than or equal to its period,
- Fig. 6.11c: a multi-periodic task with its deadline less than or equal to its longest period,
- Fig. 6.11d: a multi-periodic task with its deadline equal to its longest period,
- Fig. 6.11e: a traditional periodic task with its deadline equal to its period, and
- Fig. 6.11f: a non-repetitive task with a deadline. □

6.3.2 Approach II

In order to describe the behavior of an RTS \mathbb{S} by DES diagrams, the behavior of a task τ_i is specified as a set of regular languages that are represented by modular DES generators, in which the deadline is considered as a specification. Based on the nonblocking supervisory control of DES, the real-time tasks are represented by a DES diagram. Suppose that an RTS processes a set of real-time tasks. The corresponding DES diagram can be obtained by the synchronous product procedure.

In this approach, DES diagrams are utilized to describe the deadline of a real-time task τ_i , which requires that, after the arrival of τ_i , the execution of task τ_i should be completed in D_i time units. This model applies to both periodic/sporadic and non-repetitive task models.

After creating all the modular DES models, the real-time task models are obtained. All the possible behavior of a processor to execute a real-time task τ_i is described by a generator \mathbf{G}_i that is the *synchronous product* of the DES generators corresponding to the defined parameters, which falls into one of the six categories generated by TCT as follows:

- sporadic without a deadline, $L_m(\mathbf{G}_i) = L_m(\mathbf{G}_i^C)$:

$$\mathbf{G}_i = \mathbf{G}_i^C,$$

- non-repetitive without a deadline, $L_m(\mathbf{G}_i) = L_m(\mathbf{G}_i^{C,nr})$:

$$\mathbf{G}_i = \mathbf{G}_i^{C,nr},$$

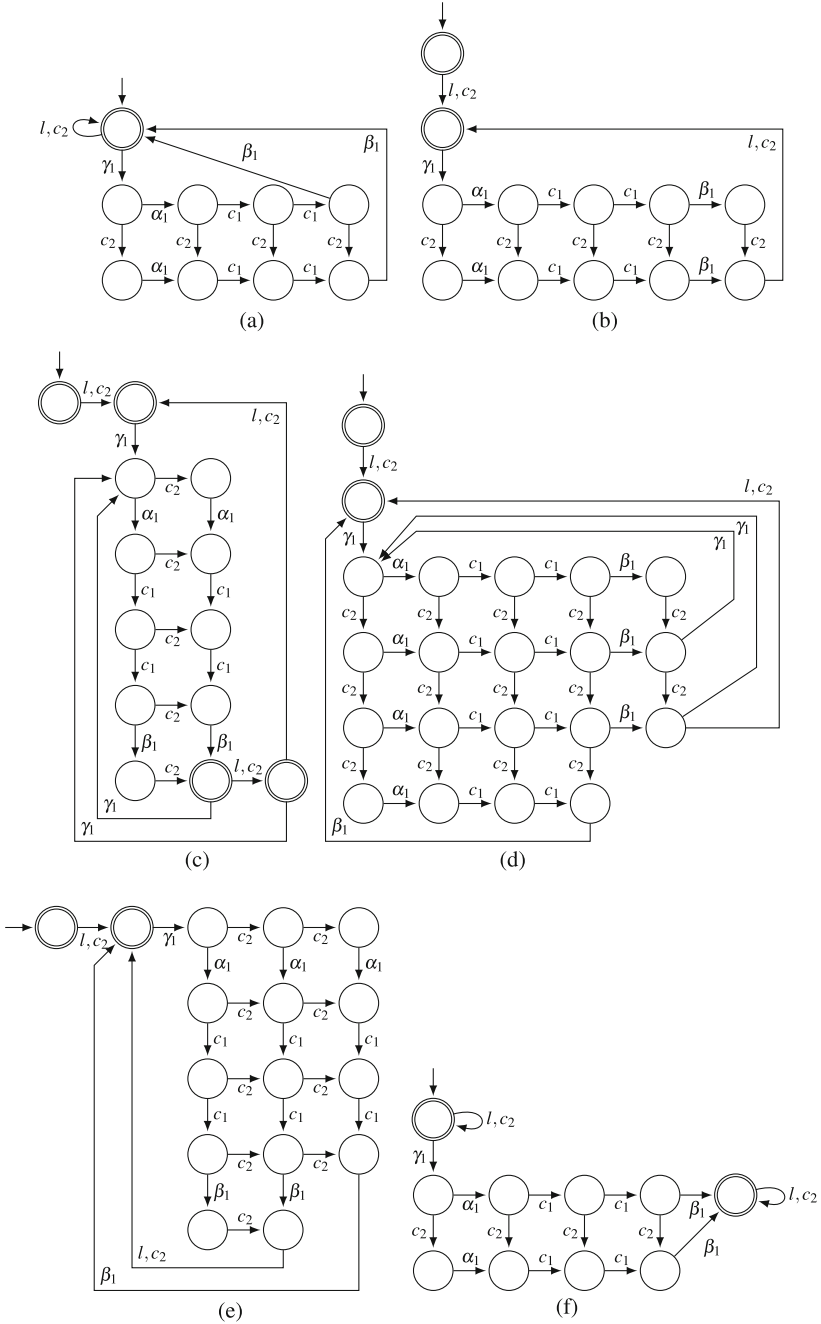


Fig. 6.11 Generators for τ_1 . (a) Sporadic, $C_1 = 2$, and $D_1 = 3$. (b) Periodic, $R_1 = 1$, $C_1 = 2$, $D_1 = 3$, and $T_1 = [4, 4]$. (c) Periodic, $R_1 = 1$, $C_1 = 2$, $D_1 = 3$, and $T_1 = [3, 5]$. (d) Periodic, $R_1 = 1$, $C_1 = 2$, and $T_1 = [3, 5]$. (e) Periodic, $R_1 = 1$, $C_1 = 2$, and $T_1 = [4, 4]$. (f) Non-repetitive, $C_1^{nr} = 2$, and $D_1 = 3$

- sporadic with a deadline, $L_m(\mathbf{G}_i) = L_m(\mathbf{G}_i^C) || L_m(\mathbf{G}_i^D)$:

$$\mathbf{G}_{ALLC} = \mathbf{allevent}(\mathbf{G}_i^C),$$

$$\mathbf{G}_{ALLD} = \mathbf{allevent}(\mathbf{G}_i^D),$$

$$\mathbf{G}_{ALL} = \mathbf{sync}(\mathbf{G}_{ALLC}, \mathbf{G}_{ALLD}),$$

$$\mathbf{G}_i^C = \mathbf{sync}(\mathbf{G}_i^C, \mathbf{G}_{ALL}),$$

$$\mathbf{G}_i^D = \mathbf{sync}(\mathbf{G}_i^D, \mathbf{G}_{ALL}),$$

$$\mathbf{G}_i = \mathbf{supcon}(\mathbf{G}_i^C, \mathbf{G}_i^D),$$

$$\mathbf{G}_i = \mathbf{minstate}(\mathbf{G}_i),$$

- non-repetitive with a deadline, $L_m(\mathbf{G}_i) = L_m(\mathbf{G}_i^{C,nr}) || L_m(\mathbf{G}_i^D)$:

$$\mathbf{G}_{ALLC} = \mathbf{allevent}(\mathbf{G}_i^{C,nr}),$$

$$\mathbf{G}_{ALLD} = \mathbf{allevent}(\mathbf{G}_i^D),$$

$$\mathbf{G}_{ALL} = \mathbf{sync}(\mathbf{G}_{ALLC}, \mathbf{G}_{ALLD}),$$

$$\mathbf{G}_i^{C,nr} = \mathbf{sync}(\mathbf{G}_i^{C,nr}, \mathbf{G}_{ALL}),$$

$$\mathbf{G}_i^D = \mathbf{sync}(\mathbf{G}_i^D, \mathbf{G}_{ALL}),$$

$$\mathbf{G}_i = \mathbf{supcon}(\mathbf{G}_i^{C,nr}, \mathbf{G}_i^D),$$

$$\mathbf{G}_i = \mathbf{minstate}(\mathbf{G}_i),$$

- periodic with a deadline $D_i < T_i^u$, $L_m(\mathbf{G}_i) = L_m(\mathbf{G}_i^C) || L_m(\mathbf{G}_i^T) || L_m(\mathbf{G}_i^D)$:

$$\mathbf{G}_i = \mathbf{sync}(\mathbf{G}_i^C, \mathbf{G}_i^T),$$

$$\mathbf{G}_{ALL} = \mathbf{allevent}(\mathbf{G}_i),$$

$$\mathbf{G}_i^D = \mathbf{sync}(\mathbf{G}_i^D, \mathbf{G}_{ALL}),$$

$$\mathbf{G}_i = \mathbf{supcon}(\mathbf{G}_i, \mathbf{G}_i^D),$$

$$\mathbf{G}_i = \mathbf{minstate}(\mathbf{G}_i),$$

- periodic with a deadline $D_i = T_i^u$, $L_m(\mathbf{G}_i) = L_m(\mathbf{G}_i^C) || L_m(\mathbf{G}_i^T)$:

$$\mathbf{G}_i = \text{sync}(\mathbf{G}_i^C, \mathbf{G}_i^T),$$

$$\mathbf{G}_{ALL} = \text{allevent}(\mathbf{G}_i),$$

$$\mathbf{G}_i = \text{supcon}(\mathbf{G}_i, \mathbf{G}_{ALL}),$$

$$\mathbf{G}_i = \text{minstate}(\mathbf{G}_i).$$

Remark For a periodic task τ_i with a deadline $D_i = T_i^u$, the obtained generator

$$\mathbf{G}_i = \text{sync}(\mathbf{G}_i^C, \mathbf{G}_i^T)$$

may contain blocked behavior that should be removed by the supervisory control of DES. Formally, it is possible that $L(\mathbf{G}_i) \neq \overline{L_m(\mathbf{G}_i)}$ holds. In order to provide a neat DES diagram, the operations $\mathbf{G}_i = \text{supcon}(\mathbf{G}_i, \mathbf{G}_{ALL})$ and $\mathbf{G}_i = \text{minstate}(\mathbf{G}_i)$ are addressed in the modelling process of \mathbf{G}_i to provide its optimal behavior. \square

Finally, generator \mathbf{G}_i for the task τ_i has *marked language* $L_m(\mathbf{G}_i)$ and (prefix) closed language $L(\mathbf{G}_i)$, satisfying

$$L(\mathbf{G}_i) = \overline{L_m(\mathbf{G}_i)}.$$

Example Suppose that $\mathbb{S} = \{\tau_1, \tau_2\}$. Six possible models for task τ_1 with $\mathbf{C}_1 = [1, 2]$ (assigned with different parameters) in the previous examples are displayed in Figs. 6.12 and 6.13, in which the corresponding parameters are listed in the captions. Accordingly, we have

- Fig. 6.12a: a sporadic task with a deadline,
- Fig. 6.12b: a periodic task with its deadline less than or equal to its period,
- Fig. 6.12c: a multi-periodic task with its deadline less than or equal to its longest period,
- Fig. 6.12d: a multi-periodic task with its deadline equal to its longest period,
- Fig. 6.13a: a traditional periodic task with its deadline equal to its period, and
- Fig. 6.13b: a non-repetitive task with a deadline. \square

6.3.3 Global RTS Behavior

Assume that n real-time tasks τ_i , $i \in \mathbf{n}$, are executed in an RTS \mathbb{S} ; its behavior is denoted by

$$L_m(\mathbf{G}) = L_m(\mathbf{G}_1) || L_m(\mathbf{G}_2) || \cdots || L_m(\mathbf{G}_n).$$

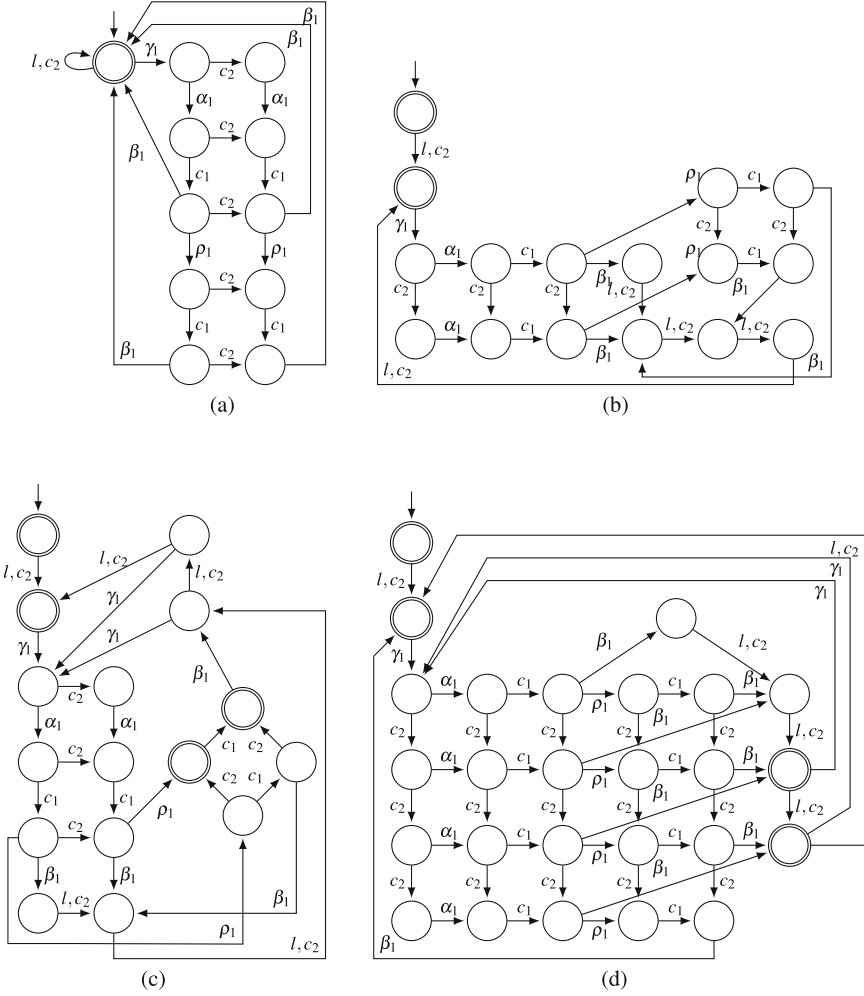


Fig. 6.12 Local closed-loop models for τ_1 (1). **(a)** Sporadic, $\mathbf{C}_1 = [1, 2]$, and $D_1 = 3$. **(b)** Periodic, $R_1 = 1$, $\mathbf{C}_1 = [1, 2]$, $D_1 = 3$, and $\mathbf{T}_1 = [5, 5]$. **(c)** periodic, $R_1 = 1$, $\mathbf{C}_1 = [1, 2]$, $D_1 = 3$, and $\mathbf{T}_1 = [3, 5]$. **(d)** Periodic, $R_1 = 1$, $\mathbf{C}_1 = [1, 2]$, $D_1 = 5$, and $\mathbf{T}_1 = [3, 5]$

This results in the respective closed and marked behaviors

$$L_m(\mathbf{G}) := L_m(\mathbf{G}_1) || L_m(\mathbf{G}_2) || \cdots || L_m(\mathbf{G}_n) \subseteq \Sigma^*$$

$$L(\mathbf{G}) := L(\mathbf{G}_1) || L(\mathbf{G}_2) || \cdots || L(\mathbf{G}_n) \subseteq \Sigma^*$$

where $\Sigma := \Sigma_1 \cup \Sigma_2 \cup \cdots \cup \Sigma_n$ denotes the overall alphabet.

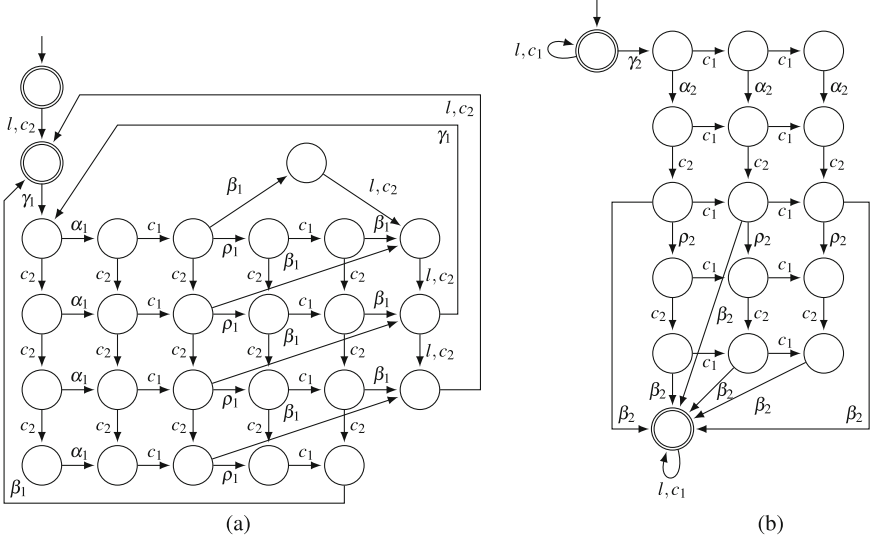


Fig. 6.13 Local closed-loop models for τ_1 (2). (a) Periodic, $R_1 = 1$, $C_1 = [1, 2]$, and $T_1 = [5, 5]$. (b) Non-repetitive, $C_2^{nr} = [1, 2]$, and $D_2 = 4$

Example Suppose that the task behavior of the two tasks in the previous example is represented by $L_m(\mathbf{G}_1)$ and $L_m(\mathbf{G}_2)$. We have the system behavior of \mathbb{S} , denoted by \mathbf{G} , as

$$L_m(\mathbf{G}) = L_m(\mathbf{G}_1) || L_m(\mathbf{G}_2). \quad \square$$

The software package TCT is a tool utilized to create the modular DES generator related to an RTS task. By following Chap. 5, for all $i \in \mathbf{n}$, events γ_i , α_i , β_i , ρ_i , and c_i are renamed $i0$, $i1$, $i2$, $i8$, and $i9$, respectively. Moreover, event l is represented by 0. More operations to edit the models and/or compute the supervisor can also be executed in TCT. The utilized procedures are introduced in [20].

An overall model of the RTS is denoted by

$$\mathbb{S} = \{\tau_1, \tau_2, \dots, \tau_n\}.$$

By construction, the effective set of shared events in the above synchronous product is the set of execution events, and, hence, is a subset of the controllable events. Consequently, the generated behavior $L(\mathbf{G})$ is controllable w.r.t. the synchronous product of the local behavior generated by the relevant plant components \mathbf{G}_i^C , $\mathbf{G}_i^{C,nr}$, \mathbf{G}_i^D , and/or \mathbf{G}_i^T , depending on the respective task type. In this sense, controllability is not a concern at this point. However, the synchronous product may introduce blocking, i.e., there may exist strings $s \in L(\mathbb{S})$ such that $st \notin L_m(\mathbf{G})$ for all $t \in \Sigma^*$. This implies there might be jobs that are never completed, which is not

acceptable. Also, the composed RTS model may include schedules where tasks might be preempted by any other task. We resolve both issues by setting up a specification to address a preemption relation and then synthesizing a nonblocking supervisor for that specification.

6.4 Scheduling Based on Supervisory Control

Previously, PFCP real-time scheduling is proposed in Chap. 5. By PFCP, a periodic task is independently assigned to a subset of tasks that are allowed to preempt its execution, based on which the released periodic tasks in a processor can be scheduled according to the predefined preemption relations assigned in the preemption matrix. In Chap. 5, three types of specifications are defined as follows:

- Nonblocking specification: it requires that the RTS should be nonblocking and it should be synchronized with other specifications (if any).
- Matrix-based conditional-preemption specification: it defines the preemption relation among the periodic tasks running in the same processor, represented by a matrix \mathbf{A} , where $\mathbf{A}_{i,j} = 1$ represents that task τ_i can be preempted by τ_j . According to matrix \mathbf{A} , Chap. 5 creates a specification for a task τ_i that describes all the tasks which preempt its execution.
- WCET-based conditional-preemption specification: it defines task sets that can preempt the execution of τ_i between its any two adjacent running time units (c_i s).

The first two specification types are assigned from the perspective of the processor; and the last one is assigned for an RTS task during its execution. All the detailed definitions of these specifications can be found in Chap. 5. Generally, based on SCT, they can be utilized to find the supervisors for an RTS processing sporadic tasks and periodic tasks simultaneously. The scheduling sequences contained in the found supervisors can be utilized to schedule the RTS offline.

As defined in Chap. 5, the *preemption matrix* \mathbf{A} of \mathbb{S} is in the form

$$\mathbf{A} = \begin{pmatrix} 0 & * & \cdots & * \\ * & 0 & \cdots & * \\ \vdots & \vdots & \ddots & \vdots \\ * & * & \cdots & 0 \end{pmatrix} \quad (6.5)$$

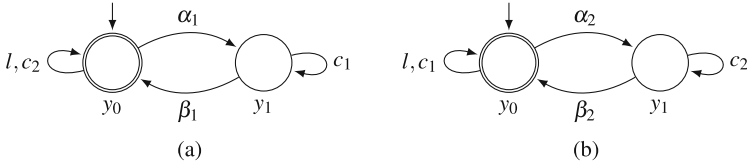
where $*$, either 0 or 1, describes the *preemption relations* among tasks. A task τ_i cannot preempt itself in default. According to the rows in \mathbf{A} , a DES specification can be generated.

Example Suppose that we have the following two tasks that are running in an RTS \mathbb{S} :

$$\tau_1 = (1, [1, 2], 3, [3, 5])$$

Table 6.1 Task configurations

Task parameters	Task type
$\tau_1 = (1, [1, 2], 3, [3, 5])$	Periodic
$\tau_2 = ([1, 2], 4)$	Non-repetitive with deadline

**Fig. 6.14** Specifications corresponding to \mathbf{A}_1 . (a) τ_1 can not be preempted. (b) τ_2 can not be preempted

and

$$\tau_2 = ([1, 2], 4).$$

They are a periodic task and a non-repetitive task, respectively. The task configurations are given in Table 6.1. Their behavior is represented by $L_m(\mathbf{G}_1)$ and $L_m(\mathbf{G}_2)$; we obtain the RTS behavior as follows:

$$L_m(\mathbf{G}) = L_m(\mathbf{G}_1) || L_m(\mathbf{G}_2)$$

and

$$L(\mathbf{G}) = L(\mathbf{G}_1) || L(\mathbf{G}_2).$$

We assume the following preemption matrices

$$\mathbf{A}_1 = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \text{ and } \mathbf{A}_2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

describing that the real-time scheduling of the RTS is non-preemptive or preemptive, respectively. According to Chap. 5, the specifications can be generated accordingly. The DES generators corresponding to \mathbf{A}_1 and \mathbf{A}_2 are depicted in Figs. 6.14 and 6.15, respectively. Since matrix \mathbf{A}_2 describes that the scheduling of both τ_1 and τ_2 are preemptive, the specifications depicted in Figs. 6.15a and 6.15b can be ignored.

Supervisory controller synthesis is then carried out using the same procedures as synthesizing the local models. After state minimization, we obtain overall closed-loop realizations with 53 states and 84 transitions (when addressing P_1) and with 151 states and 298 transitions (when addressing P_2). The transition relation for the former supervisor is given below.

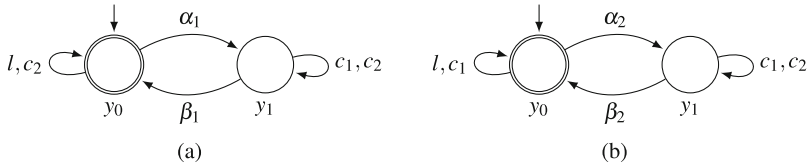
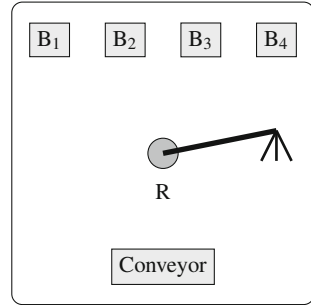


Fig. 6.15 Specifications corresponding to A_2 . (a) τ_1 can be preempted. (b) τ_2 can be preempted

Fig. 6.16 Manufacturing cell



SUPER = (SUPER, [mark, 0, 1, 6, 14, 46, 48, 52], [tran [0, l, 52], [0, γ_2 , 27], [1, l, 52], [1, γ_1 , 20], [1, γ_2 , 39], [2, c_2 , 4], [3, l, 15], [3, γ_2 , 33], [4, α_1 , 37], [4, β_2 , 13], [5, β_1 , 44], [7, γ_1 , 38], [7, β_2 , 14], [8, β_1 , 33], [8, ρ_1 , 9], [9, c_1 , 11], [10, α_2 , 44], [11, β_1 , 10], [11, α_2 , 5], [12, β_2 , 35], [13, α_1 , 40], [14, l, 46], [14, γ_1 , 36], [15, l, 6], [15, γ_2 , 10], [16, l, 23], [17, β_1 , 15], [17, γ_2 , 11], [18, β_1 , 3], [18, ρ_1 , 45], [18, γ_2 , 8], [19, c_1 , 18], [19, γ_2 , 51], [20, α_1 , 19], [20, γ_2 , 41], [21, γ_1 , 22], [21, β_2 , 48], [21, ρ_2 , 24], [22, β_2 , 36], [22, ρ_2 , 2], [23, l, 48], [23, ρ_2 , 2], [24, γ_1 , 2], [24, c_2 , 7], [25, γ_1 , 22], [25, β_2 , 46], [25, ρ_2 , 42], [26, c_2 , 25], [27, α_2 , 26], [28, β_1 , 23], [29, β_1 , 23], [29, ρ_1 , 30], [30, c_1 , 47], [31, β_2 , 23], [31, ρ_2 , 50], [32, c_2 , 31], [33, α_2 , 32], [34, β_1 , 16], [34, ρ_1 , 43], [35, c_1 , 34], [36, α_1 , 35], [37, β_2 , 40], [38, α_1 , 12], [38, β_2 , 36], [39, γ_1 , 41], [40, c_1 , 29], [41, α_1 , 51], [42, γ_1 , 2], [43, c_1 , 28], [44, c_2 , 21], [45, c_1 , 17], [45, γ_2 , 9], [46, γ_1 , 36], [47, β_1 , 48], [48, l, 14], [48, γ_1 , 36], [49, γ_1 , 38], [49, β_2 , 48], [50, c_2 , 49], [51, c_1 , 8], [52, γ_1 , 20], [52, γ_2 , 39], [6, l, 1], [6, γ_1 , 20], [6, γ_2 , 39]]) (53, 84) \square

6.5 Case Study: Manufacturing Cell

Consider an extension of the manufacturing cell studied in Chap. 5 as an example, which is viewed as an RTS processing multi-period periodic tasks, traditional periodic tasks, and sporadic tasks. The robot depicted in Fig. 6.16 is considered as a processor, which transports four types of workpieces W_1 , W_2 , W_3 , and W_4 to a conveyor. The four processes are modelled as four real-time tasks. Hence, we obtain a system $\mathbb{S} = \{\tau_1, \tau_2, \tau_3, \tau_4\}$.

Table 6.2 Tasks with WCET

Task	R_i	C_i	D_i	T_i
τ_1	–	2	–	–
τ_2	–	2	6	–
τ_3	1	2	7	[8, 8]
τ_4	0	2	6	[4, 6]

6.5.1 Task Models with Worst Case Execution Time

Suppose that in the manufacturing cell, the production rate varies, i.e., every time one or two pieces of workpieces W1, W2, W3, and W4 arrive at buffers B₁, B₂, B₃, and B₄, respectively. Workpieces W1 and W2 arrive irregularly; the arrivals of W3 and W4 are represented by period sets [8, 8] and [4, 6], respectively. Transporting each workpiece costs one second; we assign the deadlines of the tasks representing the transportations of W2, W3, and W4 to be six, seven, and six seconds, respectively. As a consequence, the parameters of the four tasks are visualized in Table 6.2. Suppose that we have three work plans, i.e.,

- $\mathbb{S}_1 = \{\tau_1, \tau_2, \tau_3\}$: workpieces W1, W2, and W3 are in production,
- $\mathbb{S}_2 = \{\tau_1, \tau_2, \tau_4\}$: workpieces W1, W2, and W4 are in production, and
- $\mathbb{S}_3 = \{\tau_1, \tau_2, \tau_3, \tau_4\}$: workpieces W1, W2, W3, and W4 are in production.

The DES model for a task in the RTS is established as the synchronous product of the generated DES models. By following Approach I presented in Sect. 6.3.1, the parameters of the RTS tasks are created as follows.

C1 = create (C1, [mark 0], [tran [0, 0, 0], [0, 10, 1], [1, 11, 2], [2, 19, 3], [3, 19, 4], [4, 12, 0]]) (5, 6)

C2 = relabel (C1, [[10, 20], [11, 21], [12, 22], [19, 29]]) (5, 6)

C3 = relabel (C2, [[20, 30], [21, 31], [22, 32], [29, 39]]) (5, 6)

C4 = relabel (C3, [[30, 40], [31, 41], [32, 42], [39, 49]]) (5, 6)

D2 = create (D2, [mark 0], [tran [0, 0, 0], [0, 19, 0], [0, 20, 1], [0, 39, 0], [0, 49, 0], [1, 19, 2], [1, 22, 0], [1, 39, 2], [1, 49, 2], [2, 19, 3], [2, 22, 0], [2, 39, 3], [2, 49, 3], [3, 19, 4], [3, 22, 0], [3, 39, 4], [3, 49, 4], [4, 19, 5], [4, 22, 0], [4, 39, 5], [4, 49, 5], [5, 22, 0]]) (6, 22)

D3 = relabel (D2, [[20, 30], [22, 32], [39, 29]]) (6, 22)

D3 = edit (D3, [trans +[5, 19, 6], +[5, 29, 6], +[5, 49, 6], +[6, 32, 0]]) (7, 26)

D4 = **relabel** (D2, [[20, 40], [22, 42], [49, 29]]) (6, 22)

T3 = **create** (T3, [mark 0, 1], [tran [0, 0, 1], [0, 19, 1], [0, 29, 1], [0, 49, 1], [1, 30, 2], [2, 0, 3], [2, 19, 3], [2, 29, 3], [2, 49, 3], [3, 0, 4], [3, 19, 4], [3, 29, 4], [3, 49, 4], [4, 0, 5], [4, 19, 5], [4, 29, 5], [4, 49, 5], [5, 0, 6], [5, 19, 6], [5, 29, 6], [5, 49, 6], [6, 0, 7], [6, 19, 7], [6, 29, 7], [6, 49, 7], [7, 0, 1], [7, 19, 1], [7, 29, 1], [7, 49, 1]]) (8, 29)

T4 = **create** (T4, [mark all], [tran [0, 41, 1], [1, 0, 2], [1, 19, 2], [1, 29, 2], [1, 39, 2], [2, 0, 3], [2, 19, 3], [2, 29, 3], [2, 39, 3], [3, 0, 4], [3, 19, 4], [3, 29, 4], [3, 39, 4], [3, 41, 1], [4, 0, 0], [4, 19, 0], [4, 29, 0], [4, 41, 1], [4, 49, 0]]) (5, 19)

The DES model for the RTS is the synchronous product of that for all tasks. More operations to edit the models and/or compute the supervisor can also be executed in TCT. The DES models representing a task's behavior are listed below.

TASK2 = **sync** (C2, D2) (21, 73)

TASK3 = **sync** (C3, D3, T3) (32, 113)

TASK4 = **sync** (C4, D4, T4) (25, 87)

Suppose that four preemption matrices are considered as specifications:

$$\mathbf{A}_1 = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}, \mathbf{A}_2 = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

$$\mathbf{A}_3 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}, \mathbf{A}_4 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

The nonblocking specifications and matrix-based conditional-preemption specifications are assigned for the manufacturing cell example, which are created in TCT as listed below.

Nonblocking Specifications

In order to guarantee that the RTS is nonblocking, the nonblocking specification \mathbf{S}_i^N for τ_i should allow the occurrence of any string $s \in \Sigma_i^*$, i.e., $L(\mathbf{S}_i^N) = \Sigma_i^*$. The procedure **allevents** can be utilized to generate a DES representing Σ_i^* . For instance, as depicted in Fig. 6.17, the nonblocking specification for task τ_i is represented by a generator with $*$ = Σ_i allowing all the events in Σ_i to occur at the only state.

Fig. 6.17 Nonblocking specification for τ_i

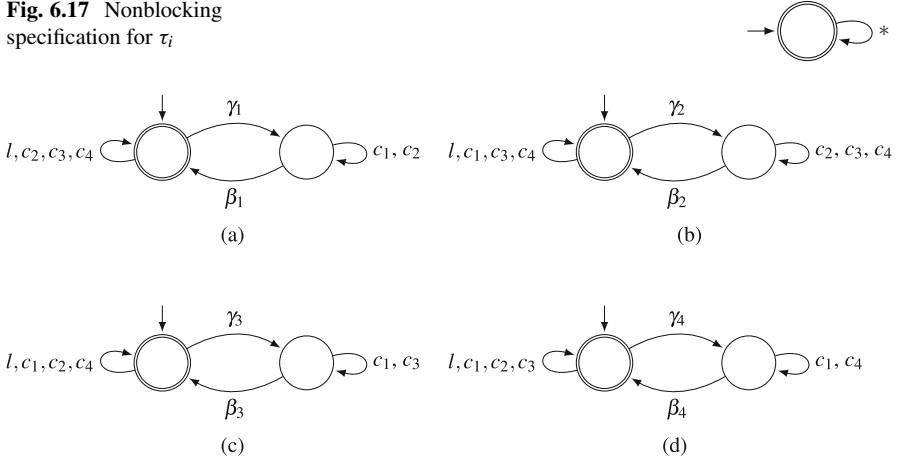


Fig. 6.18 Specification for A_3 . (a) $A_{1,2} = 1$. (b) $A_{2,3} = 1$ and $A_{2,4} = 1$. (c) $A_{31} = 1$. (d) $A_{41} = 1$

The corresponding TCT operations to create such specifications are listed below. SN1, SN2, SN3, and SN4 are the nonblocking specifications S_1^N , S_2^N , S_3^N , and S_4^N , respectively. In TCT, the monolithic nonblocking specification for an RTS \mathcal{S} is denoted by SN. The corresponding TCT operations are given below.

SN1 = **allevents** (C1) (1, 5)

SN2 = **allevents** (TASK2) (1, 8)

SN3 = **allevents** (TASK3) (1, 8)

SN4 = **allevents** (TASK4) (1, 8)

SN = **sync** (SN1, SN2, SN3, SN4) (1, 17)

Matrix-Based PFCP Conditional-Preemption Specifications

According to [17], the matrix-based PFCP conditional-preemption specifications can be generated by TCT. For example, matrix-based specification A_3 represents that task τ_1 is allowed to be preempted by task τ_2 . The corresponding specification is depicted in a DES diagram 1B2. As depicted in Fig. 6.18, all such specifications are created below.

1B2 = **create** (1B2, [mark 0], [tran [0, 0, 0], [0, 11, 1], [0, 29, 0], [0, 39, 0], [0, 49, 0], [1, 12, 0], [1, 19, 1], [1, 29, 1]]) (2, 8)

$2B3 = \mathbf{create}$ ($2B3$, [mark 0], [tran [0, 0, 0], [0, 19, 0], [0, 21, 1], [0, 39, 0], [0, 49, 0], [1, 22, 0], [1, 29, 1], [1, 39, 1]]) (2, 8)

$3B1 = \mathbf{create}$ ($3B1$, [mark 0], [tran [0, 0, 0], [0, 19, 0], [0, 29, 0], [0, 31, 1], [0, 49, 0], [1, 19, 1], [1, 32, 0], [1, 39, 1]]) (2, 8)

$4B1 = \mathbf{create}$ ($4B1$, [mark 0], [tran [0, 0, 0], [0, 19, 0], [0, 29, 0], [0, 39, 0], [0, 41, 1], [1, 19, 1], [1, 42, 0], [1, 49, 1]]) (2, 8)

The DES generators corresponding to A_4 are depicted in Fig. 6.19. The corresponding TCT operations are given below. Each specification represents that the corresponding task cannot be preempted. For example, 1NP represents that the execution of task τ_1 cannot be preempted.

$1NP = \mathbf{create}$ (1NP, [mark 0], [tran [0, 0, 0], [0, 11, 1], [0, 29, 0], [0, 39, 0], [0, 49, 0], [1, 12, 0], [1, 19, 1]]) (2, 7)

$2NP = \mathbf{create}$ (2NP, [mark 0], [tran [0, 0, 0], [0, 19, 0], [0, 21, 1], [0, 39, 0], [0, 49, 0], [1, 22, 0], [1, 29, 1]]) (2, 7)

$3NP = \mathbf{create}$ (3NP, [mark 0], [tran [0, 0, 0], [0, 19, 0], [0, 29, 0], [0, 31, 1], [0, 49, 0], [1, 32, 0], [1, 39, 1]]) (2, 7)

$4NP = \mathbf{create}$ (4NP, [mark 0], [tran [0, 0, 0], [0, 19, 0], [0, 29, 0], [0, 39, 0], [0, 41, 1], [1, 42, 0], [1, 49, 1]]) (2, 7)

By supervisory control of DES, for different scheduling plans with different specifications, we obtain the states and transitions of the corresponding supervisors as listed in Table 6.3 under “Super 1”, in which notation (\cdot, \cdot) represents the number

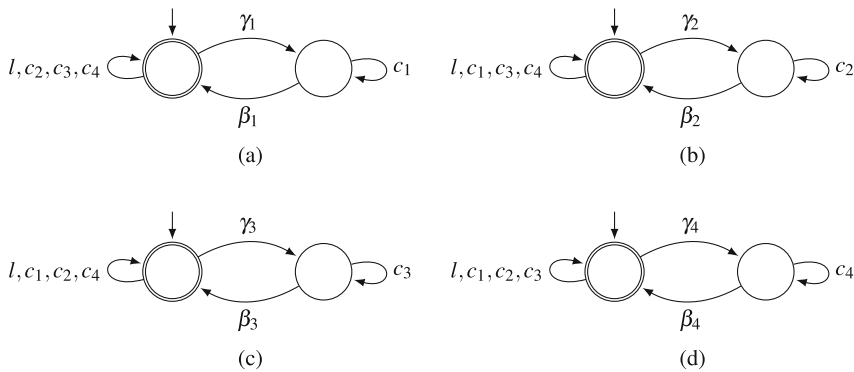


Fig. 6.19 Specification for A_4 . (a) τ_1 cannot be preempted. (b) τ_2 cannot be preempted. (c) τ_3 cannot be preempted. (d) τ_4 cannot be preempted

Table 6.3 Supervisors of RTS

G	S	Super 1	Super 2
\mathbb{S}^1	A₁	(3055, 8363)	(2791,7310)
\mathbb{S}^1	A₂	(2725, 6533)	(2512,5759)
\mathbb{S}^1	A₃	(2132, 4815)	(2083,4454)
\mathbb{S}^1	A₄	(1069, 2252)	(1126,2219)
\mathbb{S}^2	A₁	(2425, 6559)	(2335,6034)
\mathbb{S}^2	A₂	(2170, 5131)	(2092,4723)
\mathbb{S}^2	A₃	(1632, 3635)	(1695,3559)
\mathbb{S}^2	A₄	(810, 1674)	(903,1736)
\mathbb{S}^3	A₁	(62952, 197655)	(72397,218029)
\mathbb{S}^3	A₂	(36057, 90381)	(42006,100880)
\mathbb{S}^3	A₃	(13300, 30071)	(17043,36174)
\mathbb{S}^3	A₄	(10076, 21868)	(13819,27971)

of the states and transitions in the supervisor. For each supervisor in Table 6.3, one can use the method developed in [16] (stated in Chap. 4) to view the release and scheduling maps that represent their task release orders and scheduling processes, respectively. Suppose that task τ_1 in Table 6.3 is replaced by a non-repetitive task $\tau'_1 = ([2, 2], 4)$. The corresponding supervisors are listed in Table 6.3 under “Super 2”.

The DES generators corresponding to the specifications of an RTS are synchronized into a monolithic one. Some specifications listed in Table 6.3 under **S** are synchronized as follows.

$$\text{SPEC2} = \mathbf{sync} (3\text{NP}, \text{SN1}, \text{SN2}, \text{SN}) (2, 27)$$

$$\text{SPEC3} = \mathbf{sync} (\text{SN}, 1\text{B2}, 2\text{B3}, 3\text{B1}) (8, 80)$$

$$\text{SPEC4} = \mathbf{sync} (1\text{NP}, 2\text{NP}, 3\text{NP}, \text{SN}) (8, 77)$$

$$\text{SPEC6} = \mathbf{sync} (\text{SN}, 4\text{NP}) (2, 27)$$

$$\text{SPEC7} = \mathbf{sync} (\text{SN}, 1\text{B2}, 2\text{B4}, 4\text{B1}) (8, 80)$$

$$\text{SPEC8} = \mathbf{sync} (\text{SN}, 1\text{NP}, 2\text{NP}, 4\text{NP}) (8, 77)$$

$$\text{SPEC10} = \mathbf{sync} (\text{SN}, 3\text{NP}, 4\text{NP}) (4, 45)$$

$$\text{SPEC11} = \mathbf{sync} (\text{SN}, 1\text{B2}, 2\text{B3}, 2\text{B4}, 3\text{B1}, 4\text{B1}) (16, 137)$$

$$\text{SPEC12} = \mathbf{sync} (\text{SN}, 1\text{NP}, 2\text{NP}, 3\text{NP}, 4\text{NP}) (16, 133)$$

Finally, the supervisors under “Super 1” are calculated as follows.

$$\text{SUPER1} = \text{supcon} (\text{SYS1}, \text{SN}) (3055, 8363)$$

$$\text{SUPER2} = \text{supcon} (\text{SYS1}, \text{SPEC2}) (2725, 6533)$$

$$\text{SUPER3} = \text{supcon} (\text{SYS1}, \text{SPEC3}) (2132, 4815)$$

$$\text{SUPER4} = \text{supcon} (\text{SYS1}, \text{SPEC4}) (1069, 2252)$$

$$\text{SUPER5} = \text{supcon} (\text{SYS2}, \text{SN}) (2425, 6559)$$

$$\text{SUPER6} = \text{supcon} (\text{SYS2}, \text{SPEC6}) (2170, 5131)$$

$$\text{SUPER7} = \text{supcon} (\text{SYS2}, \text{SPEC7}) (1632, 3635)$$

$$\text{SUPER8} = \text{supcon} (\text{SYS2}, \text{SPEC8}) (810, 1674)$$

$$\text{SUPER9} = \text{supcon} (\text{SYS3}, \text{SN}) (62952, 197655)$$

$$\text{SUPER10} = \text{supcon} (\text{SYS3}, \text{SPEC10}) (36057, 90381)$$

$$\text{SUPER11} = \text{supcon} (\text{SYS3}, \text{SPEC11}) (13300, 30071)$$

$$\text{SUPER12} = \text{supcon} (\text{SYS3}, \text{SPEC12}) (10076, 21868)$$

6.5.2 Task Models with Exact Execution Time

Suppose that, as stated in Table 6.4, the BCETs and WCETs of all the tasks are equal to one and two time units, respectively. For different specifications of a scheduling plan, the numbers of the states and transitions of the supervisors are listed in Table 6.5 under “Super 1” in the form (number of states, number of transitions). Suppose that task τ_1 in Table 6.5 is replaced by a non-repetitive task $\tau'_1 = ([1, 2], 4)$. The corresponding supervisors are listed in Table 6.5 under “Super 2”. The supervisor (0, 0) in Table 6.5 represents that no safe execution sequences are found.

Table 6.4 Task parameters

Task	R_i	C_i	D_i	T_i
τ_1	–	[1, 2]	–	–
τ_2	–	[1, 2]	6	–
τ_3	1	[1, 2]	7	[8, 8]
τ_4	0	[1, 2]	6	[4, 6]

Table 6.5 Supervisors

Plant	Spec	Super 1	Super 2
S^1	$A_1, D_2 = 6, D_3 = 7$	(5593, 17392)	(6196, 18170)
S^1	$A_2, D_2 = 6, D_3 = 5$	(3577, 9630)	(2615, 6559)
S^1	$A_3, D_2 = 6, D_3 = 5$	(1129, 2728)	(861, 1804)
S^1	$A_4, D_2 = 6, D_3 = 7$	(745, 1502)	(541, 1051)
S^2	$A_1, D_2 = 6, D_4 = 6$	(4500, 14317)	(7770, 23301)
S^2	$A_2, D_2 = 6, D_4 = 5$	(3264, 9221)	(3537, 9241)
S^2	$A_3, D_2 = 6, D_4 = 5$	(1533, 4023)	(1755, 4264)
S^2	$A_4, D_2 = 6, D_4 = 6$	(582, 1258)	(525, 1076)
S^3	$A_1, D_2 = 6, D_3 = 7, D_4 = 6$	(147277, 574215)	(171171, 607542)
S^3	$A_2, D_2 = 6, D_3 = 5, D_4 = 5$	(0, 0)	(0, 0)
S^3	$A_3, D_2 = 6, D_3 = 5, D_4 = 5$	(0, 0)	(0, 0)
S^3	$A_4, D_2 = 6, D_3 = 7, D_4 = 6$	(8646, 20307)	(4167, 8220)

As stated in Chap. 1, a method that speeds up the calculation reduces the number of states in the plant and specification. The presented synthesis speeding up approach can be applied to this chapter.

6.6 Conclusion

This chapter reports a unified DES-based framework to build RTS by modular models and scheduling/reconfiguring RTS by SCT. This framework can be utilized to model an RTS that processes multi-period and sporadic tasks, in which a multi-period task is assigned with a set of possible periods between a minimum period and a maximum period. The proposed modular models are taken to be generic entities, which are utilized to model a problem domain such as “hard real-time manufacturing or reconfigurations” and manage its manufacturing process.

In practice, the execution time of a real-time task is expected to vary over time, within guaranteed bounds referred to as the BCET and WCET respectively. This motivates our further development of SCT-based RTS scheduling of real-time tasks previously proposed in Chap. 5. Building on the idea of exact execution time, this chapter provides a modular scheduling/reconfiguration methodology for RTS processing non-repetitive, sporadic, and (multi-period) periodic tasks, subject to a PFCP scheduling policy. Moreover, the three-step speeding up algorithm stated in Chap. 1 can be applied to this chapter. We illustrate our modelling framework in the context of a realistic manufacturing system.

A hierarchical RTS model is presented in Chap. 7, based on nonblocking supervisory control of *state-tree structures* (STS), where both conditionally-preemptive and dynamic priority scheduling are addressed in the SCT-based real-time scheduling. The task release/arrival, starting, and finishing are on the higher level, and the

task execution is on the lower level. This modelling mechanism provides the possibility of assigning dynamic priorities for the real-time tasks under execution. The computational complexity of the STS framework is polynomial with respect to the number of the nodes in a *binary decision diagram* that describes the RTS's behavior. Hence, the nonblocking supervisory control of STS may return results for the cases where the TCT algorithm fails.

References

1. Baruah, S.K., Rosier, L.E., Howell, R.R.: Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Syst.* **2**(4), 301–324 (1990)
2. Buttazzo, G.C.: *Hard Real-time Computing Systems: Predictable Scheduling Algorithms and Applications*, vol. 24. Springer Science & Business Media (2011)
3. Chen, P.C.Y., Wonham, W.M.: Real-time supervisory control of a processor for non-preemptive execution of periodic tasks. *Real-Time Syst.* **23**, 183–208 (2002)
4. Davis, R.I.: A review of fixed priority and EDF scheduling for hard real-time uniprocessor systems. *ACM SIGBED Rev.* **11**(1), 8–19 (2014)
5. Howell, R.R., Venkatrao, M.K.: On non-preemptive scheduling of recurring tasks using inserted idle times. *Inf. Comput.* **117**(1), 50–62 (1995)
6. Janarthanan, V., Gohari, P., Saffar, A.: Formalizing real-time scheduling using priority-based supervisory control of discrete-event systems. *IEEE Trans. Autom. Control* **51**(6), 1053–1058 (2006)
7. Leung, J.Y.T., Whitehead, J.: On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Perform. Eval.* **2**(4), 237–250 (1982)
8. Li, J., Shu, L., Chen, J., Li, G.: Energy-efficient scheduling in nonpreemptive systems with real-time constraints. *IEEE Trans. Syst. Man Cybern. Syst.* **43**(2), 332–344 (2013)
9. Li, D., Li, M., Meng, X., Tian, Y.: A hyperheuristic approach for intercell scheduling with single processing machines and batch processing machines. *IEEE Trans. Syst. Man Cybern. Syst.* **45**(2), 315–325 (2015)
10. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM* **20**(1), 46–61 (1973)
11. Mok, A.K.: *Fundamental design problems of distributed systems for the hard-real-time environment*. Ph.D. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts (1983)
12. Nasser, E., Bres, G.: Hard real-time sporadic task scheduling for fixed priority schedulers. In: *International Workshop on Responsive Systems*, pp. 44–47 (1991)
13. Park, S.J., Cho, K.H.: Real-time preemptive scheduling of sporadic tasks based on supervisory control of discrete event systems. *Inf. Sci.* **178**(17), 3393–3401 (2008)
14. Sha, L., Abdelzaher, T., Cervin, A., Baker, T., Burns, A., Buttazzo, G., Caccamo, M., Lehoczky, J., Mok, A.K.: Real time scheduling theory: a historical perspective. *Real-Time Syst.* **28**(2), 101–155 (2004)
15. Wang, X., Khemaissia, I., Khalgui, M., Li, Z., Mosbahi, O., Zhou, M.: Dynamic low-power reconfiguration of real-time systems with periodic and probabilistic tasks. *IEEE Trans. Autom. Sci. Eng.* **12**(1), 258–271 (2015)
16. Wang, X., Li, Z., Wonham, W.M.: Dynamic multiple-period reconfiguration of real-time scheduling based on timed DES supervisory control. *IEEE Trans. Ind. Inf.* **12**(1), 101–111 (2016)

17. Wang, X., Li, Z., Wonham, W.M.: Optimal priority-free conditionally-preemptive real-time scheduling of periodic tasks based on DES supervisory control. *IEEE Trans. Syst. Man Cybern. Syst.* **47**(7), 1082–1098 (2017)
18. Wang, X., Li, Z., Wonham, W.M.: Priority-free conditionally-preemptive scheduling of modular sporadic real-time systems. *Automatica* **89**, 392–397 (2018)
19. Wang, X., Li, Z., Wonham, W.M.: Real-time scheduling based on nonblocking supervisory control of state-tree structures. *IEEE Trans. Autom. Control* **66**(9), 4230–4237 (2021)
20. Wonham, W.M., Cai, K.: *Supervisory Control of Discrete-Event Systems*. Monograph Series Communications and Control Engineering, Springer, Berlin (2018)
21. Xia, Y., Zhou, M., Luo, X., Pang, S., Zhu, Q.: A stochastic approach to analysis of energy-aware DVS-enabled cloud datacenters. *IEEE Trans. Syst. Man Cybern. Syst.* **45**(1), 73–83 (2015)

Chapter 7

Scheduling/Reconfiguration Based on Supervisory Control of STS



7.1 Introduction

Supervisory control theory (SCT)-based real-time scheduling and reconfiguration [6, 7, 12, 15–18] are a newly-identified research topic. Based on timed discrete-event systems (TDES) or discrete-event systems (DES), *multi-period tasks* have been proposed in Chaps. 4 and 6, which can be utilized to model periodic tasks processed in *real-time systems* (RTS) with periods varying between a *lower bound* and an *upper bound*. As a consequence, it is a uniformed model which integrates real-time scheduling and reconfiguration. The only difference of a task's model before and after its reconfiguration is the upper bound of its multi-period.

Chapter 5 shows that both *preemptive* and *non-preemptive* scheduling policies may be conservative. As a solution, based on supervisory control of DES, a real-time scheduling principle is presented, namely *priority-free conditionally-preemptive (PFCP) scheduling*. Based on the PFCP real-time scheduling principle and the specifications provided by users, all the safe execution sequences of an RTS processing both multi-period periodic and sporadic tasks can be found.

As a top-down state-based modelling framework with the *state explosion problem* managed, *state-tree structures* (STS) are defined in [8] and [9] for the purpose of incorporating the *hierarchical (vertical)* and *concurrent (horizontal)* structures of complex DES into a natural and compact model. The holons in STS represent *hierarchical* and *concurrent transition structures* of DES with *structured state spaces*; for details see Chap. 2, [8], and [9].

In this chapter, RTS are modelled starting from holons. The RTS model is converted into an STS automatically. The *controller* for each *controllable event* in the STS is obtained by the supervisory control of STS to provide the expected safe execution sequences. A task is associated with a constant *worst-case execution time* (WCET). WCET and the corresponding deadlines, release time, and periods (if any) are modelled by child-state-trees and holons in STS. As a consequence, a sporadic RTS is modelled by an STS. Based on this unified STS-based framework,

the scheduling requirements of an RTS are described by STS specifications that require only a small-sized state space.

The PFCP specifications proposed in Chap. 5 are converted to STS specifications in a compact form such that the state explosion problem is effectively managed. Moreover, a *partially non-preemptive specification* is given to partially define the *preemption relation* among tasks. As an optimal dynamic priority scheduling, partially preemptive and non-preemptive *earliest deadline first* (EDF) scheduling is also addressed in this chapter. From a large number of safe execution sequences, a set of optimal sequences can be found by the EDF scheduler.

The STS model and its specifications are represented by predicates. Prior to calculating the predicate representing the optimal controlled behavior, *maximally permissive predicates* are presented for disabling controllable events in the transition structure. The PFCP scheduling reported in this chapter is applied to a practical context to schedule a real-world RTS. For a controllable event, the controller is represented by a *binary decision diagram* (BDD) [2, 3] representing a boolean function. A BDD is a rooted *directed acyclic graph* that has two *terminal nodes* 1 and 0 that represent *true* and *false*, respectively.

In this chapter, the RTS are modelled according to the following principles:

- an RTS is modelled in a hierarchical (vertical) and concurrent (horizontal) structure by using STS,
- the conditionally-preemptive and dynamic priority scheduling requirements are converted into STS specifications which need much less *storage space* than the *automata* of the DES framework, and
- with the state explosion problem managed significantly, the nonblocking supervisory control of STS finds all the safe execution sequences.

The methodology in this chapter guarantees that:

- the presented scheduling framework can find the *optimal behavior* (all the safe execution sequences) of an RTS by the supervisory control of STS, and
- a few sequences are selected, which rank at the top according to some specified optimality criteria.

The remainder of this chapter is organized as follows. The STS-based real-time task model is described in Sect. 7.2. The conditionally-preemptive and dynamic priority scheduling specifications are reported in Sects. 7.3 and 7.4, respectively. As an example, the nonblocking supervisory control of Manufacturing Cell and a large example are presented in Sects. 7.5 and 7.6. Finally, conclusions are drawn in Sect. 7.7.

7.2 RTS Modelled by State-Tree Structures

This chapter presents an approach to model RTS processing sporadic and/or periodic tasks by STS, which can be viewed as an STS counterpart of the DES model

presented in Chap. 6. For simplification, we consider the WCET of each real-time task only. Nevertheless, there is no technical problem to extend the approach to the exact execution time (studied in Chap. 6) with both the best case execution time (BCET) and WCET addressed. The preliminaries on STS can be found in Sect. 2.3.

7.2.1 RTS Tasks

Suppose that a periodic RTS \mathbb{S} processes n tasks, i.e., $\mathbb{S} = \{\tau_1, \tau_2, \dots, \tau_i, \dots, \tau_n\}$, $i \in \mathbf{n} = \{1, 2, \dots, n\}$. Generally, a periodic task τ_i is specified as a four-tuple

$$\tau_i = (R_i, C_i, D_i, \mathbf{T}_i)$$

with

- a *release time* R_i ,
- a *WCET* C_i ,
- a *deadline* D_i , and
- a *multi-period* \mathbf{T}_i ,

where R_i , C_i , and D_i are non-negative integers. A deadline D_i is *hard* if its violation is unacceptable; otherwise it is *soft*, which is not treated in this chapter. As stated in Chap. 4, a multi-period is specified by a non-empty interval

$$\mathbf{T}_i = [T_i^l, T_i^u] \in \mathbb{N} \times \mathbb{N},$$

where the number of time units that elapse between any two successive releases lies within \mathbf{T}_i . Hence a multi-period has a *lower bound* (i.e., shortest one) represented by T_i^l and an *upper bound* (i.e., longest one) represented by T_i^u . Only a period T in \mathbf{T}_i of task τ_i is selected in a scheduling period.

The following different types of real-time tasks are addressed in this chapter:

- a sporadic/non-repetitive task without a deadline: $\tau_i = (C_i)$,
- a sporadic/non-repetitive task with a deadline: $\tau_i = (C_i, D_i)$,
- a periodic task with its deadline not equal to T_i^u : $\tau_i = (R_i, C_i, D_i, \mathbf{T}_i)$, and
- a periodic task with its deadline equal to T_i^u : $\tau_i = (R_i, C_i, \mathbf{T}_i)$.

Example Suppose that four asynchronous tasks are running in an RTS \mathbb{S} . We consider the WCET of the tasks stated in the example of Chap. 6 only, and their parameters are listed in Table 7.1, i.e.,

- τ_1 : a sporadic task without a deadline,
- τ_2 : a sporadic task with a deadline,
- τ_3 : a traditional periodic task, and
- τ_4 : a multi-period periodic task.

Table 7.1 Parameters of four tasks

Task	R_i	C_i	D_i	T_i
τ_1	–	2	–	–
τ_2	–	2	6	–
τ_3	1	2	7	[8, 8]
τ_4	0	2	6	[4, 6]

Unless stated otherwise, all the exemplifications are based on this example. There is no technical problem to design STS counterparts for the RTS model presented in Chap. 6. \square

7.2.2 Execution Time Models

The execution time of a real-time task τ_i is modelled hierarchically by two holons H^{C_i} and H^{W_i} . Holon H^{C_i} is on a higher level, and hence holon H^{W_i} is considered as a superstate [8, 9] in holon H^{C_i} . The *initial* (resp., *terminal*) states of the holons without external states are marked by incoming (resp., outgoing) arrows.

Definition 7.1 [holon H^{C_i}] The execution time C_i of task τ_i is described by a holon $H^{C_i} := (X^{C_i}, \Sigma^{C_i}, \delta^{C_i}, X_0^{C_i}, X_m^{C_i})$ with state set $X^{C_i} = X_I^{C_i} = \{I_i, A_i, W_i\}$, event set $\Sigma^{C_i} = \Sigma_I^{C_i} := \{\gamma_i, \alpha_i, \beta_i\}$, transitions $\delta_I^{C_i}(I_i, l) = I_i$, $\delta_I^{C_i}(I_i, \gamma_i) = A_i$, $\delta_I^{C_i}(A_i, \alpha_i) = W_i$, $\delta_I^{C_i}(W_i, \beta_i) = I_i$, and the *initial* and *terminal state set* $X_0^{C_i} = X_m^{C_i} = \{I_i\}$. \diamond

Example By zooming into state W_i in Fig. 7.1, a lower-level holon H^{W_i} and the matching state-tree representing the inner structure of W_i are shown in Fig. 7.2. Clearly, before the arrival of τ_i and after the completion of its execution, the processor is allowed to be in an idle operation. Moreover, holon H^{W_i} represents the system behavior inside state W_i . The states in Fig. 7.1 are defined as:

- I_i : *idle*, i.e., before the arrival of task τ_i ,
- A_i : *arrival*, i.e., task τ_i has arrived in the system, and
- W_i : *working*, i.e., task τ_i is being executed by the processor. \square

The lower-level holon H^{W_i} is defined as follows.

Definition 7.2 [holon H^{W_i}] Let $0 \leq k < C_i$. The system behavior in state W_i is described by a holon $H^{W_i} := (X^{W_i}, \Sigma^{W_i}, \delta^{W_i}, X_0^{W_i}, X_m^{W_i})$ with state set $X^{W_i} := \{I_i, A_i, w_i^0, w_i^1, \dots, w_i^{C_i}\}$ partitioned into $X_E^{W_i} = \{I_i, A_i\}$ and $X_I^{W_i} = \{w_i^0, w_i^1, \dots, w_i^{C_i}\}$, event set $\Sigma^{W_i} := \{\alpha_i, \beta_i, c_i\}$ partitioned into $\Sigma_B^{W_i} := \{\alpha_i, \beta_i\}$ and $\Sigma_I^{W_i} := \{c_i\}$, transitions $\delta_{BI}(A_i, \alpha_i) = w_i^0$, $\delta_{BO}(w_i^{C_i}, \beta_i) = I_i$, and $\delta_I(w_i^k, c_i) = w_i^{k+1}$, the initial state set $X_0^{C_i} = \{w_i^0\}$, and the terminal state set $X_m^{C_i} = \{w_i^{C_i}\}$. \diamond

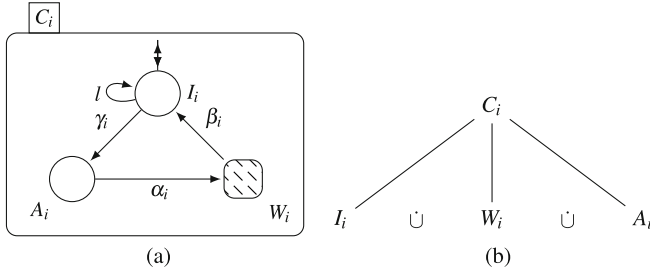


Fig. 7.1 Higher-level model of WCET. (a) Holon H^{C_i} . (b) Child-state-tree ST^{C_i}

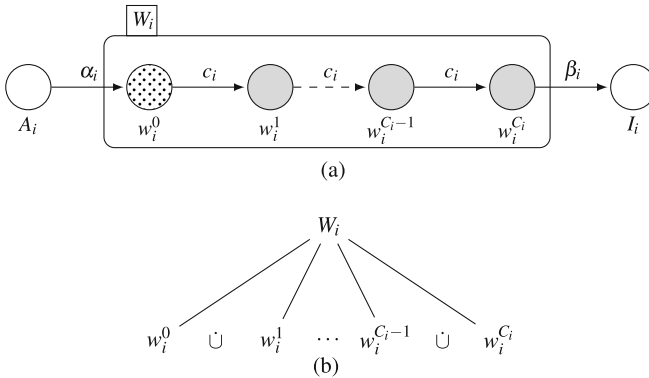


Fig. 7.2 Lower-level model of WCET. (a) Holon H^{W_i} . (b) Child-state-tree ST^{W_i} .

Example For task τ_i , as shown in Fig. 7.1, its WCET C_i is represented by a holon and a matching state-tree depicted in Figs. 7.1a and 7.1b, respectively. In accordance with Fig. 2.13, superstate W_i is represented by a square dashed with north west lines. □

The state spaces of the holons form child-state-trees. For instance, in Fig. 7.2, a holon H^{W_i} contains $C_i + 1$ states, which represents the system behavior inside state W_i . According to Sect. 2.3.2, a superstate in a holon is represented by a box. Let $0 \leq k \leq C_i$. State w_i^k represents that task τ_i has been processed for k time units.

Example By plugging the holon illustrated in Fig. 7.2a into Fig. 7.1a, a two-level holon that represents C_i is depicted in Fig. 7.3a. It is matched with a child-state-tree ST^{C_i} depicted in Fig. 7.3b. In accordance with both Sects. 2.3.2 and 4.2.2, the states filled with gray or crosshatch dots represent that they are in a lower level holon, and the states filled with crosshatch dots represent that the corresponding task is under execution. □

Remark The holon depicted in Fig. 7.1 is similar to the TDES active transition graph illustrated in Fig. 4.1 but more general. Their semantics is different. For

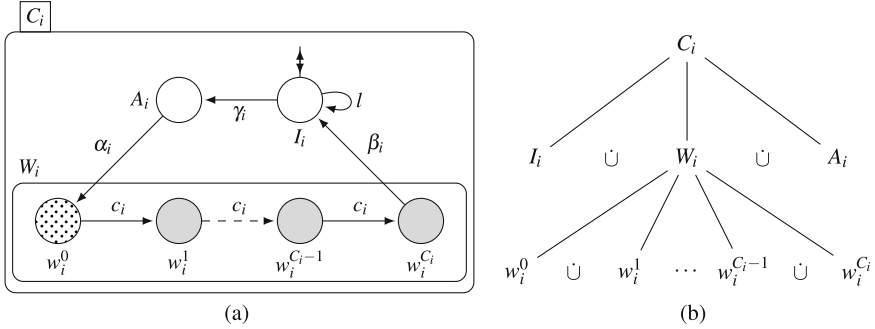


Fig. 7.3 Hierarchical STS model of C_i . (a) Holon H_{C_i} with two levels. (b) Child-state-tree ST^{C_i}

example, the occurrences of event β_i in Figs. 4.1 and 7.1 are “ticked down” by a global tick event t and a local event c_i representing task τ_i under execution, respectively. Clearly, event t represents any possible operation in a time unit; however, event c_i represents that a time unit is utilized to execute task τ_i . \square

7.2.3 Deadline Models

The deadline D_i of a task τ_i is an integer denoting the maximal time difference between its arrival (the occurrence of γ_i) and its *execution completion* (the occurrence of β_i). In STS, deadline D_i is represented by a holon H^{D_i} that matches a child-state-tree ST^{D_i} , which is defined below.

Definition 7.3 [deadline holon] Let $0 \leq k < D_i$, $C_i \leq p \leq D_i$, and $1 \leq j \leq n$. The deadline D_i of task τ_i is represented by a holon $H^{D_i} := (X^{D_i}, \Sigma^{D_i}, \delta^{D_i}, X_0^{D_i}, X_m^{D_i})$ with state set $X^{D_i} = X_I^{D_i} := \{d_i^R, d_i^0, d_i^1, \dots, d_i^{D_i}\}$, event set $\Sigma^{D_i} = \Sigma_I^{D_i} := \{\gamma_i, \beta_i, c_1, c_2, \dots, c_n, l\}$, transitions ($j \neq i$) $\delta_I^{D_i}(d_i^R, c_j) = d_i^R$, $\delta_I^{D_i}(d_i^R, l) = d_i^R$, $\delta_I^{D_i}(d_i^R, \gamma_i) = d_i^0$, $\delta_I^{D_i}(d_i^k, c_j) = d_i^{k+1}$, $\delta_I^{D_i}(d_i^k, l) = d_i^{k+1}$, and $\delta_I^{D_i}(d_i^p, \beta_i) = d_i^R$, and the initial and terminal state set $X_0^{D_i} = X_m^{D_i} = \{d_i^R\}$. \diamond

For task τ_i , the deadline D_i is represented by holon H^{D_i} shown in Fig. 7.4, in which “*” and “**” represent the events in Σ_e and $\Sigma_e - \{c_i\}$, respectively. For simplicity, the states between d_i^1 and $d_i^{D_i}$ (except $d_i^{C_i}$) are omitted in this figure. The omissions are represented by arrows with dashed lines. The states in Fig. 7.4 are defined as:

- d_i^R : the execution of task τ_i is completed or has not started, and
- d_i^k : the k -th time unit is being utilized to process real-time tasks or in an idle operation.

The deadline D_i of task τ_i is an integer denoting the maximal time difference between its arrival (the occurrence of γ_i) and its execution completion (the occurrence of β_i).

According to Chap. 5, the *alphabet* (set of *event labels*) Σ_i describing the processor's behavior to execute task τ_i is:

- γ_i : task τ_i is *released*,
- α_i : the *execution* of τ_i is *started*,
- β_i : the execution of τ_i is *completed*,
- c_i ($i \in \mathbf{n}$): the processor starts to execute τ_i for one *processor time unit*, and
- l : *empty action*, i.e., the processor is in an idle operation for one time unit.

Let $\sigma = c_i$ (resp., $\sigma = l$). The occurrence of σ represents that one processor time unit is utilized to execute task τ_i (resp., in an idle operation for one processor time unit while it stays in state q').

Formally, for an RTS, its event set is defined as

$$\Sigma = \Sigma_{con} \dot{\cup} \Sigma_{unc},$$

with

- $\Sigma_{con} = \{\alpha_i, c_i | i \in \mathbf{n}\}$: the controllable event subset, and
- $\Sigma_{unc} = \{\beta_i, \gamma_i, l | i \in \mathbf{n}\}$: the uncontrollable event subset.

Moreover, Σ is also partitioned into

- $\Sigma_o = \{\gamma_i, \alpha_i, \beta_i | i \in \mathbf{n}\}$: the *operation event set*, and
- $\Sigma_e = \{c_i, l | i \in \mathbf{n}\}$: the *execution event set*.

Example In Fig. 7.4, the transition

$$(C_i \leq p \leq D_i) \delta_i^{D_i} (d_i^p, \beta_i) = d_i^R$$

represents all the possible occurrences of event β_i . Figure 7.4a matches the child-state-tree ST^{D_i} as depicted in Fig. 7.4b. □

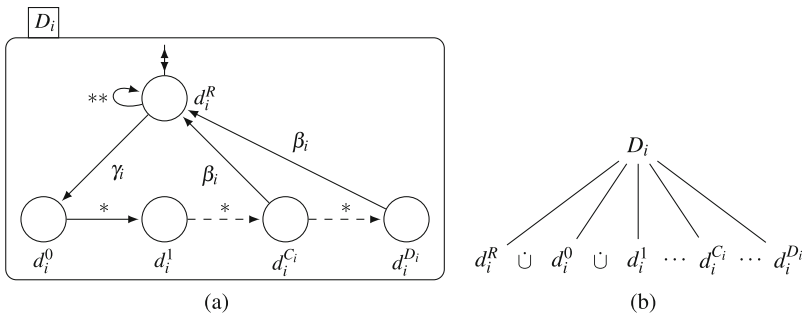


Fig. 7.4 STS model of D_i . (a) Holon H^{D_i} . (b) Child-state-tree ST^{D_i}

7.2.4 Release Time and Period Models

For a periodic task τ_i , the occurrence of event γ_i represents the end of its current processing period (if any) and the start of the next period. In STS, the release time R_i and period \mathbf{T}_i of a task τ_i are represented by a holon $H^{\mathbf{T}_i}$ that matches a child-state-tree $ST^{\mathbf{T}_i}$, which is defined below.

Definition 7.4 [release time and period holon] Let $0 < k \leq R_i$, $0 \leq p < T_i^u - 1$, $T_i^l \leq q < T_i^u$, and $1 \leq j \leq n$. The release time R_i and period \mathbf{T}_i of a task τ_i are represented by a holon $H^{\mathbf{T}_i} := (X^{\mathbf{T}_i}, \Sigma^{\mathbf{T}_i}, \delta^{\mathbf{T}_i}, X_0^{\mathbf{T}_i}, X_m^{\mathbf{T}_i})$ with state set $X^{\mathbf{T}_i} = X_I^{\mathbf{T}_i} := \{r_i^{R_i}, r_i^{R_i-1}, \dots, r_i^0, t_i^0, t_i^1, \dots, t_i^{T_i^u-1}\}$; event set $\Sigma^{\mathbf{T}_i} = \Sigma_I^{\mathbf{T}_i} := \{\gamma_i, c_1, c_2, \dots, c_n, l\}$; transitions ($j \neq i$) $\delta_I^{\mathbf{T}_i}(r_i^k, c_j) = r_i^{k-1}$, $\delta_I^{\mathbf{T}_i}(r_i^k, l) = r_i^{k-1}$, $\delta_I^{\mathbf{T}_i}(r_i^0, \gamma_i) = t_i^0$, $\delta_I^{\mathbf{T}_i}(t_i^p, c_j) = t_i^{p+1}$, $\delta_I^{\mathbf{T}_i}(t_i^p, l) = t_i^{p+1}$, $\delta_I^{\mathbf{T}_i}(t_i^{T_i^u-1}, c_j) = r_i^0$, $\delta_I^{\mathbf{T}_i}(t_i^{T_i^u-1}, l) = r_i^0$, and $\delta_I^{\mathbf{T}_i}(t_i^q, \gamma_i) = t_i^0$; the initial state set $X_0^{\mathbf{T}_i} = \{r_i^{R_i}\}$ and the terminal state set $X_m^{\mathbf{T}_i} = \{r_i^{R_i}, r_i^{R_i-1}, \dots, r_i^0\}$. \diamond

For a task τ_i , the release time R_i and period \mathbf{T}_i are represented by the holon shown in Fig. 7.5, in which “*” and “**” represent the events in Σ_e and $\Sigma_e - \{c_i\}$, respectively. Let $0 \leq k \leq R_i$ and $0 \leq p < T_i^u$. The states in Fig. 7.5 are defined as:

- r_i^k : k time units before the arrival of task τ_i , and
- t_i^p : the k -th time unit is being utilized to process real-time tasks or in an idle operation.

For a periodic task τ_i , the occurrence of event γ_i represents the end of its current processing period (if any) and the start of the next period. Let $T_i^l \leq q < T_i^u$. In Fig. 7.5, the transitions

$$\delta_I^{\mathbf{T}_i}(r_i^0, \gamma_i) = t_i^0$$

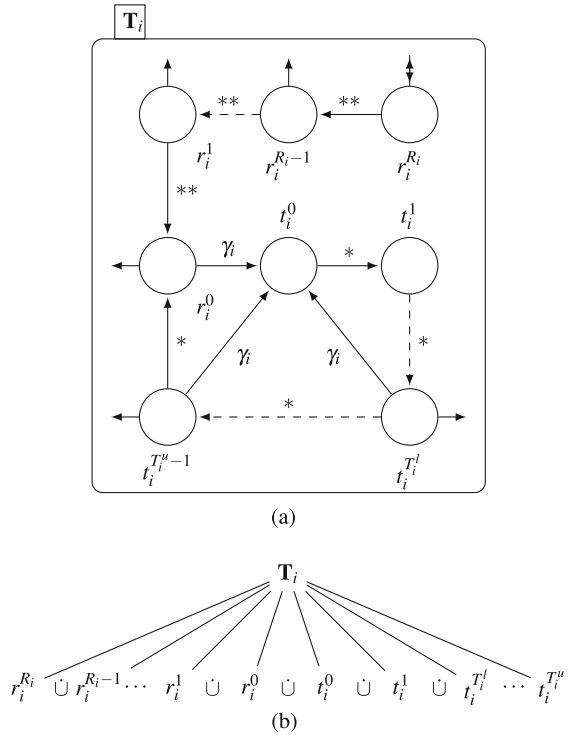
and

$$\delta_I^{\mathbf{T}_i}(t_i^q, \gamma_i) = t_i^0$$

represent all the possible occurrences of event γ_i .

Example Figure 7.5a represents the possible multi-period of task τ_i , which matches a child-state-tree $ST^{\mathbf{T}_i}$, as depicted in Fig. 7.5b. Since the RTS may process any task or remain idle within the period of a task τ_i or before its deadline, we allow event c_i ($i \in \mathbf{n}$) to occur if the system is at states d_i^k , $0 \leq k < D_i$ and t_i^p , $0 \leq p < T_i^u$, simultaneously. \square

Fig. 7.5 STS model of \mathbf{T}_i .
(a) Holon $H^{\mathbf{T}_i}$. **(b)** Child-state-tree $ST^{\mathbf{T}_i}$



7.2.5 Task Models

Finally, as depicted in Fig. 7.6a, the parameters of task τ_i are modelled by (at most) the following four holons:

- holon H^{C_i} (contains H^{W_i}): execution time C_i ,
- holon H^{W_i} : execution process of task τ_i ,
- holon H^{D_i} : deadline D_i , and
- holon $H^{\mathbf{T}_i}$: release time R_i and period \mathbf{T}_i .

In Fig. 7.6a, “*” and “**” represent the events in Σ_e and $\Sigma_e - \{c_i\}$, respectively. For task τ_i , superstates C_i , D_i , and \mathbf{T}_i are the expansions of an AND *superstate* TK_i . Hence, the main structure of the state-tree rooted by superstate TK_i is illustrated in Fig. 7.6b. The global child-state-tree representing real-time task τ_i is obtained by plugging the necessary holons (shown in Figs. 7.3b, 7.4b, and 7.5b) into Fig. 7.6b. Notice that state D_i (or \mathbf{T}_i) is deleted from Fig. 7.6b if the matching holon H^{D_i} (or $H^{\mathbf{T}_i}$) is not needed. The root state C_i is replaced by TK_i directly if only holon H^{C_i} is necessary. As a consequence, holon H^{C_i} is renamed to be H^{TK_i} .

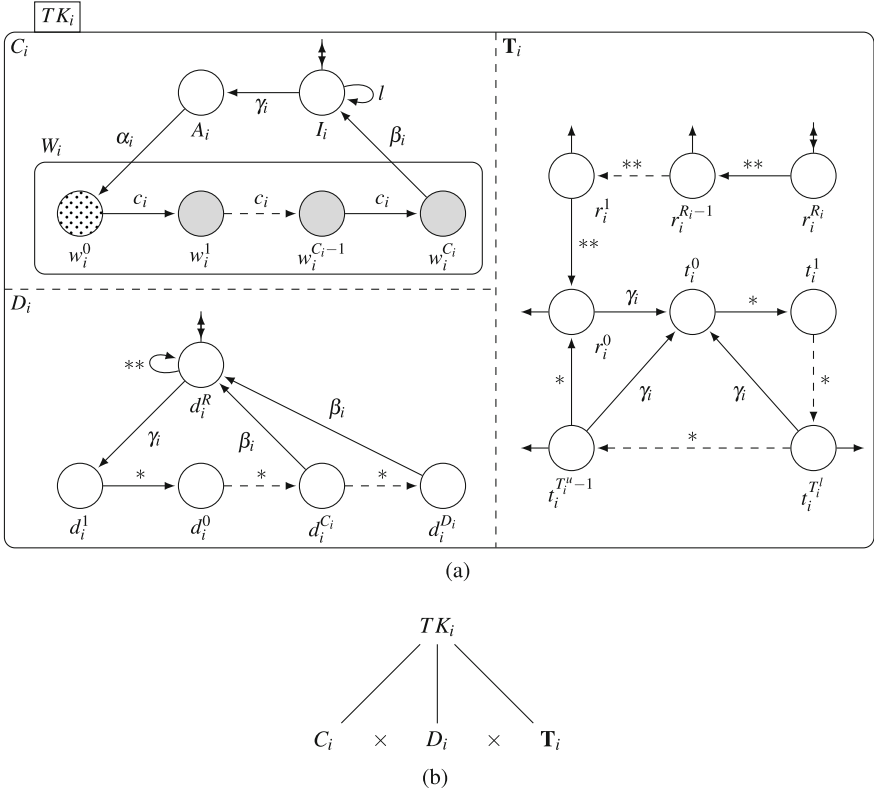


Fig. 7.6 Global model of real-time tasks. (a) Holons. (b) Child-state-tree

Remarks

1. The STS model depicted in Fig. 7.6 represents that all the holons are running synchronously.
2. The RTS model of task τ_i is the synchronous product of (at most) holons H^{C_i} , H^{D_i} , and H^{T_i} . Holons H^{D_i} or H^{T_i} in Fig. 7.6a can be removed in the case that they are unnecessary or not defined. For example, the deadline of periodic task τ_4 in Table 7.1 is equal to its maximum period T_i^μ . Thus holon H^{D_4} is ignored when building the model of τ_4 .
3. The condition for an event σ to occur is: σ is eligible to occur in all the holons where it appears. For example, event γ_i in Fig. 7.6a can occur only if the system is at states I_i, r_i^0 , and d_i^R simultaneously.
4. From the perspective of holon H^{C_i} , H^{W_i} is an internal state in $X_I^{C_i}$.

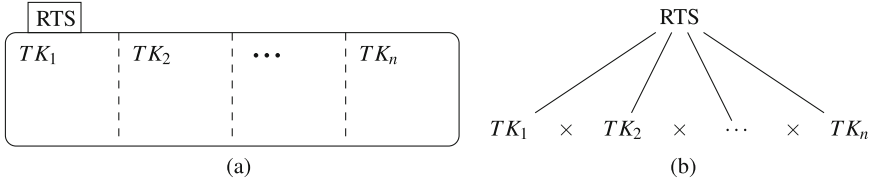


Fig. 7.7 Global model of \mathbb{S} . (a) Holons modelling \mathbb{S} . (b) Child-state-tree modelling \mathbb{S}

5. By following the presented approach, there is no technology barrier for addressing the exact execution time (discussed in Chap. 6) in holon H^{W_i} . Users only need to revise H^{W_i} accordingly.
6. In this section, $\{I_i, d_i^R, r_i^{R_i}\}$ is defined as the initial state set of the STS depicted in Fig. 7.6a. □

7.2.6 Global RTS Execution Models

The monolithic RTS execution model is the synchronous product of the models of all the RTS tasks under execution. In STS, the holons and state-tree representing RTS \mathbb{S} are depicted in Figs. 7.7a and 7.7b, respectively. By plugging the task models presented above into the RTS model we obtain the global RTS model, in which the global RTS behavior is represented by an AND superstate, namely RTS.

Example Suppose that the four tasks given in Table 7.1 are running in an RTS \mathbb{S} . The corresponding holons and the structure of the state-tree are depicted in Figs. 7.8 and 7.9, respectively. □

7.3 Conditionally-Preemptive Specifications

According to the STS framework, two types of *state-based specifications*¹ are defined:

Type 1: *mutual exclusion* $\{x_1, x_2, \dots, x_n\}$: a system should avoid occupying states x_1, x_2, \dots, x_n in an STS model simultaneously;

Type 2: forbidden events at *non-empty state sets* $(\{x_1, x_2, \dots, x_n\}, \sigma)$: at a state set $\{x_1, x_2, \dots, x_n\}$ in an STS model, event σ is not allowed to occur.

As a consequence, the following two types of RTS real-time scheduling specifications are defined.

¹ The states x_1, x_2, \dots, x_n in these specifications can belong to different holons.

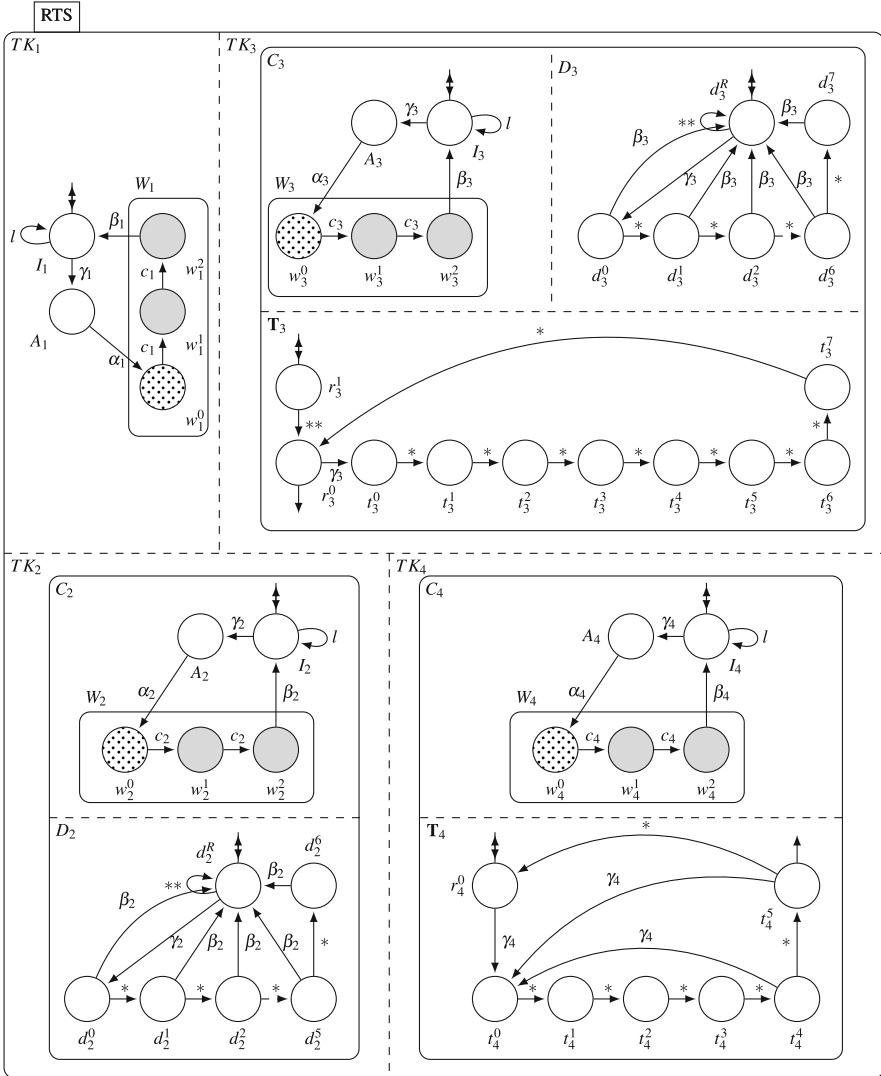
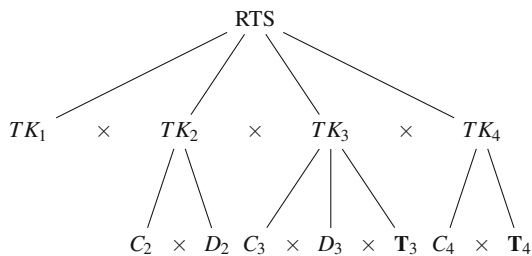


Fig. 7.8 Holons of RTS S

7.3.1 Matrix-Based Conditional-Preemption Specifications

In order to define the preemption relations among the tasks executed in a uni-processor, a *preemption matrix* A is presented in Chap. 5 taking the form

Fig. 7.9 State-tree of RTS S



$$\mathbf{A} = \begin{pmatrix} 0 & * & \cdots & * \\ * & 0 & \cdots & * \\ \vdots & \vdots & \ddots & \vdots \\ * & * & \cdots & 0 \end{pmatrix} \quad (7.1)$$

where $*$, equal to either 0 or 1, can be predefined by users. Accordingly, a task τ_i is (resp., is not) allowed to be interrupted by the execution of τ_j if $\mathbf{A}_{i,j} = 1$ (resp., $\mathbf{A}_{i,j} = 0$) for $i \neq j$.

In a preemption matrix \mathbf{A} , if $i \neq j$, $\mathbf{A}_{i,j} = 0$ is equivalently represented by a Type 2 specification $(\{W_i\}, \alpha_j)$. In particular, if two tasks τ_i and τ_j are not allowed to preempt each other, the specification is denoted by a Type 1 specification $\{W_i, W_j\}$. It forbids a system from visiting W_i and W_j simultaneously.

Example Suppose that the execution of task τ_3 is not allowed to be preempted by τ_1 , i.e., $\mathbf{A}_{3,1} = 0$. The PFCP specifications are represented by $(\{W_3\}, \alpha_1)$ and $(\{W_3\}, c_1)$, which represents that while task τ_3 is under execution, the occurrences of events α_1 and c_1 are prohibited. Moreover, if $\mathbf{A}_{1,3} = \mathbf{A}_{3,1} = 0$, we have a specification $\{W_1, W_3\}$, which shows that tasks τ_1 and τ_3 are not allowed to stay in holons H^{W_1} and H^{W_3} to process them simultaneously. \square

7.3.2 Task-Centered Specifications

Let $i, j \in \mathbf{n}$, $i \neq j$, and $0 \leq k \leq C_i$. More specifically, a specification $(\{w_i^k\}, c_j)$ represents that at state w_i^k the occurrence of event c_j is forbidden, which shows that at state w_i^k the execution of task τ_i cannot be preempted by task τ_j .

Example Suppose that, at states w_1^0 and w_1^1 , the execution of τ_1 is not allowed to be preempted by τ_3 . Then the state-based specifications are denoted by $(\{w_1^0\}, c_3)$ and $(\{w_1^1\}, c_3)$. \square

7.4 Dynamic Specifications

EDF scheduling is a dynamic priority scheduling strategy [14]. At every processor time unit, EDF dynamically assigns the highest priority to the tasks with the earliest deadlines. *Least laxity first* (LLF) scheduling, as another optimal algorithm, is proposed by Mok in [10], which assigns the processor to the active task with the smallest laxity. LLF has a larger overhead than EDF due to a larger number of context switches caused by laxity changes at run time. This section mainly focuses on the EDF specifications.

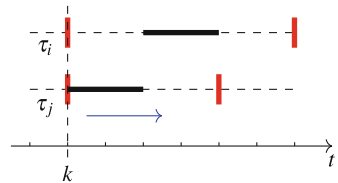
7.4.1 Earliest-Deadline First Task Selection at Arrival

Generally, suppose that two tasks τ_i and τ_j with $D_i > D_j$ arrive simultaneously at time unit $t \geq 0$. The tasks other than the one with the *earliest deadline* are prevented from entering the processor for execution. Such an example is depicted in the *Gantt chart* depicted in Fig. 7.10.² Before executing τ_i or τ_j , i.e., at state set $\{t_i^0, t_j^0\}$, the system should prevent task τ_i from entering the processor (at state set $\{w_i^0\}$) while task τ_j is at state set $\{A_j\}$. We have a Type 2 specification $\{A_j, w_i^0, t_i^0, t_j^0\} = \{A_j\} \cup \{w_i^0\} \cup \{t_i^0, t_j^0\}$. Consequently, the EDF task selection is described in Fact 7.1.

Fact 7.1 (EDF Task Selection) *Let $S \subseteq \mathbb{S}$ and $S \neq \emptyset$. The EDF task selection specification for the tasks in S is equivalently represented by a state-based specification set $\{\{A_j, w_i^0, t_i^0, t_j^0\} \mid \tau_i, \tau_j \in S, D_i > D_j\}$.* \diamond

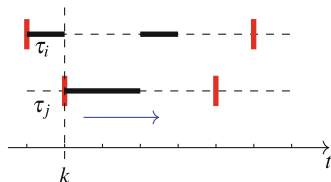
Example For the tasks given in Table 7.1, we have $D_2 < D_3$ and $D_4 < D_3$. The EDF task selection specification is $\{\{A_2, w_3^0, t_2^0, t_3^0\}, \{A_4, w_3^0, t_4^0, t_3^0\}\}$. Notice that no period is assigned to task τ_2 , and the specification $\{A_2, w_3^0, t_2^0, t_3^0\}$ is invalid. However, there is no problem for assigning such a specification for STS since state set $\{A_2, w_3^0, t_2^0, t_3^0\}$ is neither reachable nor coreachable. \square

Fig. 7.10 EDF task selection at arrival



² The two tasks in Figs. 7.10, 7.11, and 7.12 are assigned randomly as $\tau_i = (_, 2, 6, [6, 6])$ and $\tau_j = (_, 2, 4, [4, 4])$.

Fig. 7.11 Partially preemptive EDF



7.4.2 Partially Preemptive Earliest-Deadline First Scheduling

Let $S \subseteq \mathbb{S}$, $S \neq \emptyset$, $\tau_i, \tau_j \in S$, $0 \leq k \leq D_i$, and $0 \leq p \leq D_j$. States d_i^k and d_j^p represent that during the execution of tasks τ_i and τ_j , k and p time units have elapsed, respectively. As depicted in Fig. 7.11, during (or at least starting) the execution of any two tasks τ_i and τ_j (at state set $\{A_i, A_j, W_i, W_j, d_i^k, d_j^p\}$), i.e., at any time unit $t = k \geq 0$, the execution of τ_i should be preempted if the deadline of τ_j is earlier (i.e., $D_i - d_i^k > D_j - d_j^p$). Accordingly, the *partially preemptive EDF scheduling specification* is described in Fact 7.2.

Fact 7.2 (Partially Preemptive EDF) Let $S \subseteq \mathbb{S}$, $S \neq \emptyset$, $\tau_i, \tau_j \in S$, $0 \leq k \leq D_i$, and $0 \leq p \leq D_j$. The state-based specification set $\{(\{A_i, A_j, W_i, W_j, d_i^k, d_j^p\}, \alpha_i), (\{A_i, A_j, W_i, W_j, d_i^k, d_j^p\}, c_i) \mid \tau_i, \tau_j \in S, D_i - d_i^k > D_j - d_j^p\}$ specifies a partially preemptive EDF scheduling strategy for the tasks in S . \diamond

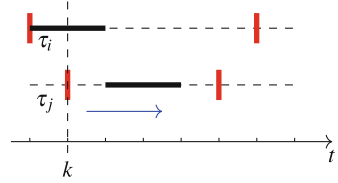
Remarks

1. The specifications generated in Facts 7.1 and 7.2 form the complete partially preemptive EDF scheduling strategy.
2. The partially preemptive EDF specifications can be combined with the PFCP specifications.
3. The specifications $\{(\{A_i, A_j, W_i, W_j, d_i^k, d_j^p\}, \alpha_i) \mid \tau_i, \tau_j \in S, D_i - d_i^k > D_j - d_j^p\}$ can be ignored if some scheduling scenarios place no constraints on the start of any task's execution. In this case, the EDF scheduling is still satisfied since in any execution period, event α_i occurs prior to the occurrence of event c_i (the execution of task τ_i). \square

7.4.3 Partially Non-Preemptive Earliest-Deadline First Scheduling

Non-preemptive EDF specifications are very common in industrial implementations. For instance a manufacturing process may require that, after a task τ_i enters the processor and before its execution is completed, the processing of τ_i cannot be preempted even if another task τ_j is assigned with a higher priority.

Fig. 7.12 Partially non-preemptive EDF



Let $S \subseteq \mathbb{S}$ with $S \neq \emptyset$, $\tau_i, \tau_j \in S$, $0 \leq k \leq D_i$, and $0 \leq p \leq D_j$. As depicted in Fig. 7.12, the system is not allowed to visit state set $\{W_i, W_j\}$. Moreover, after the arrival of any two tasks τ_i and τ_j (at state set $\{A_i, A_j, d_i^k, d_j^p\}$), i.e., at time $t = k \geq 0$, the execution of τ_i will be preempted if the deadline of τ_j comes earlier ($D_i - d_i^k > D_j - d_j^p$). Accordingly, the *partially non-preemptive EDF scheduling specification* is described in Fact 7.3.

Fact 7.3 (Partially Non-Preemptive EDF) *Let $S \subseteq \mathbb{S}$ with $S \neq \emptyset$, $\tau_i, \tau_j \in S$, $0 \leq k \leq D_i$, and $0 \leq p \leq D_j$. The state-based specification sets $\{\{W_i, W_j\} \mid \tau_i \in S, \tau_j \in \mathbb{S}, i \neq j\}$ and $\{(\{A_i, A_j, d_i^k, d_j^p\}, \alpha_i) \mid \tau_i, \tau_j \in S, D_i - d_i^k > D_j - d_j^p\}$ specify a partially non-preemptive EDF strategy for the tasks in S . \diamond*

Remarks

1. The specifications generated in Facts 7.1 and 7.3 form the complete partially non-preemptive EDF scheduling strategy.
2. The partially preemptive/non-preemptive EDF (dynamic) specifications can be combined with the PFCP specifications.
3. The partially preemptive/non-preemptive LLF scheduling specifications can be designed similarly, which is not touched upon in this chapter. \square

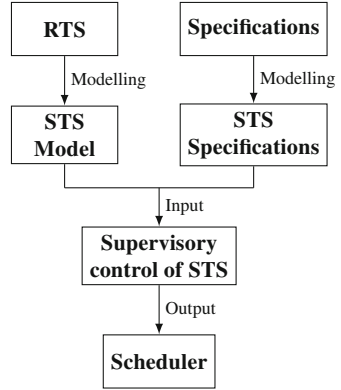
7.5 Supervisor Synthesis with a Case Study: Manufacturing Cell

By assigning dynamic priorities to RTS tasks, based on the STS modelling and supervisory control mechanism, a few safe execution sequences are selected, which rank at the top according to specified optimality criteria. The presented RTS models using STS can be synthesized in a software package STSLib,³ which utilizes BDD as the basis of efficient computation.

Thanks to the supervisory control of STS [8, 9, 19], the controllers for the controllable events are obtained to provide the expected safe execution sequences. Briefly, this process is depicted in the diagram in Fig. 7.13.

³ <https://github.com/chuanma/STSLib>.

Fig. 7.13 RTS scheduling diagram



Case studies on the manufacturing cell studied in Chaps. 5 and 6 are recalled in this section. Based on the nonblocking supervisory control of STS, the controller for each controllable event in the RTS is calculated, which provides all the expected safe execution sequences.

7.5.1 Compact Encoding of the Manufacturing Cell

Consider the manufacturing cell studied in Sect. 6.5 as an example. Two pieces of W1 (resp., W2) are released to the input buffer B1 (resp., B2) simultaneously every 6 (resp., 3) seconds. The robot R has capacity one; transporting each piece takes 1 second. Thus, we define a task $\tau_1 = (0, 2, 6, [6, 6])$ (resp., $\tau_2 = (0, 2, 3, [3, 3])$) to represent the transportation of the two pieces of W1 (resp., W2) by R. Consequently, we have a system $\mathbb{S} = \{\tau_1, \tau_2\}$, whose holons are shown in Fig. 7.14.

In the STS framework, the manufacturing cell is encoded into BDD. As proposed in [4] and [5], the states in the state set X^x of a holon H^x are encoded by *BDD nodes* (variables). Consider a state set X^x with a state space $|X^x| = N$. An element y in X^x is encoded as a *vector of n binary values*, where $n = \lceil \log_2 N \rceil$. The *encoding process* is denoted by a function $f : X^x \rightarrow \{0, 1\}^n$ that maps an element y in X^x to a distinct n -bit binary *vector*. According to [8], the n variables are denoted by x_i with $0 \leq i < n$.

As shown in Fig. 7.14, seven states form the state set of holon H^{T_1} , i.e., $X^{T_1} = \{t_1^0, t_1^0, t_1^1, \dots, t_1^5\}$. As a consequence, three BDD nodes $T1_i$ with $0 \leq i < 3$ are needed to encode the states in X^{T_1} . (In the encoding process, superstate T_1 is represented by RT_1 .) Let $T1_i : 0$ and $T1_i : 1$ denote that $T1_i$ is encoded as 0 and 1, respectively. The encoding pairs are shown in Table 7.2. Similarly, the states in holons H^{C_2} and H^{W_2} are encoded as shown in Table 7.3.

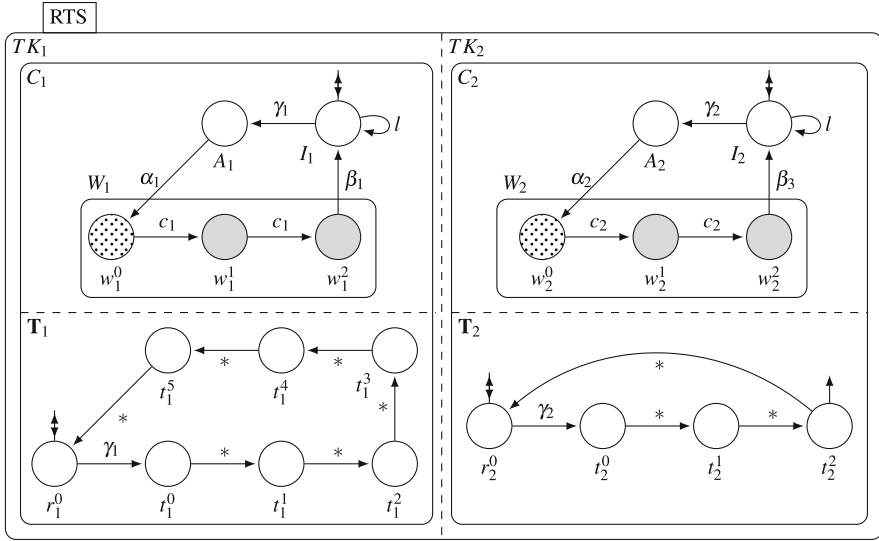


Fig. 7.14 Holons of manufacturing cell

Table 7.2 BDD vectors encoding the states in H^{T1}

State	BDD vector
r_1^0	$\langle RT1_2 : 0, RT1_1 : 0, RT1_0 : 0 \rangle$
t_1^0	$\langle RT1_2 : 0, RT1_1 : 0, RT1_0 : 1 \rangle$
t_1^1	$\langle RT1_2 : 0, RT1_1 : 1, RT1_0 : 0 \rangle$
t_1^2	$\langle RT1_2 : 0, RT1_1 : 1, RT1_0 : 1 \rangle$
t_1^3	$\langle RT1_2 : 1, RT1_1 : 0, RT1_0 : 0 \rangle$
t_1^4	$\langle RT1_2 : 1, RT1_1 : 0, RT1_0 : 1 \rangle$
t_1^5	$\langle RT1_2 : 1, RT1_1 : 1, RT1_0 : 0 \rangle$

Table 7.3 BDD vectors encoding the states in H^{C2} and H^{W2}

State	BDD vector
I_2	$\langle C2_1 : 0, C2_0 : 0 \rangle$
A_2	$\langle C2_1 : 0, C2_0 : 1 \rangle$
W_2	$\langle C2_1 : 1, C2_0 : 0 \rangle$
w_2^0	$\langle W2_1 : 0, W2_0 : 0 \rangle$
w_2^1	$\langle W2_1 : 0, W2_0 : 1 \rangle$
w_2^2	$\langle W2_1 : 1, W2_0 : 0 \rangle$

7.5.2 Conditionally-Preemptive Scheduling

Suppose that a matrix-based specification is denoted by

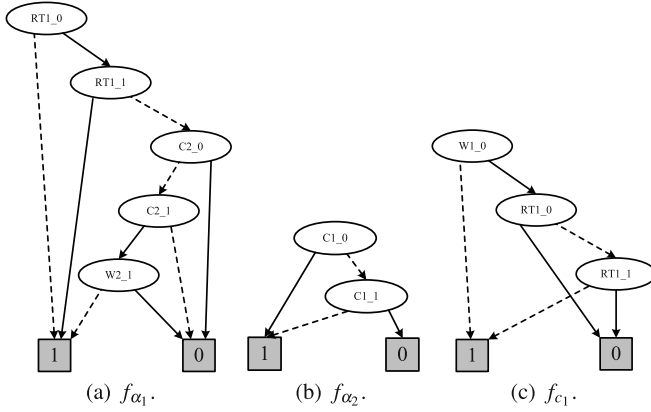


Fig. 7.15 SFBC controllers. (a) f_{α_1} . (b) f_{α_2} . (c) f_{c_1}

Table 7.4 Truth table of f_{α_1}

f_{α_1}	$RT1_0$	$RT1_1$	$RT1_2$	$C2_0$	$C2_1$	$W2_0$	$W2_1$
1	0	*	*	*	*	*	*
1	1	1	*	*	*	*	*
1	1	0	*	0	1	*	0
0	1	0	*	0	1	*	1
0	1	0	*	0	0	*	*
0	1	0	*	1	*	*	*

$$A = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix},$$

i.e., the execution of task τ_1 cannot be preempted by τ_2 . The corresponding STS specification is $(\{W_1\}, \alpha_2, \{W_1\}, c_2)$. By the *state feedback control* (SFBC) [8, 9], event c_2 can occur at any *basic-state-tree* whenever it is eligible to occur. The control functions of events α_1 , α_2 , and c_1 , denoted by f_{α_1} , f_{α_2} , and f_{c_1} , respectively, are shown in Fig. 7.15. A dashed (resp., solid) branch denotes that the variable is assigned 0 (resp., 1). According to Fig. 7.15a, the *truth table* for control function f_{α_1} is obtained, as shown in Table 7.4, where “*” denotes a variable that can be assigned 0 or 1.

The controllers display that:

- event α_1 can occur at state sets $\{A_1, t_1^2, t_1^2, w_2^2\}$ and $\{A_1, I_2, t_1^2, t_2^2\}$,
- event c_1 can occur at state sets $\{I_2, t_1^2, t_2^2, w_1^0\}$, $\{t_1^2, t_2^2, w_1^0, w_2^2\}$, and $\{A_2, t_1^3, t_2^0, w_1^1\}$,
- event α_2 can occur at state sets $\{A_1, A_2, t_1^0, t_2^0\}$, $\{I_1, A_2, r_1^0, t_2^0\}$, and $\{I_1, A_2, t_1^4, t_2^1\}$, and

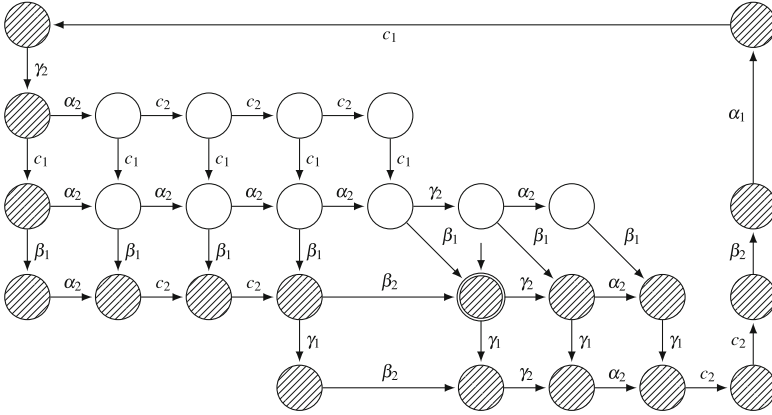


Fig. 7.16 EDF scheduling sequences

- event c_2 can occur at state sets $\{A_1, t_1^1, t_2^1, w_2^1\}$, $\{A_1, t_1^0, t_2^0, w_2^0\}$, $\{I_1, t_1^5, t_2^2, w_2^1\}$, and $\{I_1, t_1^4, t_2^1, w_2^0\}$.

The RTS under control can reach the following 24 sub-state-trees: $\{I_1, I_2, r_1^0, r_2^0\}$, $\{I_1, A_2, r_1^0, t_2^0\}$, $\{A_1, I_2, t_1^0, r_2^0\}$, $\{t_1^0, t_2^0, w_1^0, w_2^0\}$, $\{t_1^1, t_2^1, w_1^0, w_2^1\}$, $\{t_1^1, t_2^1, w_1^1, w_2^0\}$, $\{t_1^2, t_2^2, w_1^0, w_2^2\}$, $\{t_1^2, t_2^2, w_1^1, w_2^1\}$, $\{t_1^3, r_2^0, w_1^1, w_2^2\}$, $\{t_1^4, t_2^1, w_1^0, w_2^1\}$, $\{t_1^5, t_2^2, w_1^1, w_2^2\}$, $\{I_1, r_1^0, t_2^0, w_2^0\}$, $\{A_1, t_1^1, t_2^1, w_2^1\}$, $\{I_1, r_1^0, r_2^0, w_2^2\}$, $\{I_1, A_2, t_1^4, t_2^1\}$, $\{A_1, A_2, t_1^0, t_2^0\}$, $\{A_1, t_1^0, t_2^0, w_2^0\}$, $\{A_1, t_1^0, r_2^0, w_2^2\}$, $\{A_1, t_1^2, t_2^2, w_2^2\}$, $\{A_1, I_2, t_1^2, t_2^2\}$, $\{I_2, t_1^2, t_2^2, w_1^0\}$, $\{A_2, t_1^3, t_2^0, w_1^1\}$, $\{I_2, t_1^3, r_2^0, w_1^1\}$, and $\{A_2, t_1^4, t_2^1, w_2^1\}$.

7.5.3 Preemptive and Non-Preemptive Earliest-Deadline First Scheduling

For given *preemptive/non-preemptive EDF specifications*, the controllers for all the controllable events are calculated. All the *EDF scheduling sequences* are shown in Fig. 7.16, where the paths made by the states dashed by north east lines denote the non-preemptive EDF sequences. In other words, the diagram shown in Fig. 7.16 contains the three possible EDF sequences depicted in Fig. 7.17. Since in the time interval $[3, 6]$ the deadlines of τ_1 and τ_2 are equal to each other, τ_1 and τ_2 can be processed in any order. The sequence shown in Fig. 7.17a is the non-preemptive EDF sequence. The preemptive EDF specification for the RTS is given as follows.

Type 1: $\{\{A_2, w_1^0, t_1^0, t_2^0\}\}$;

Type 2: $\{(\{W_1, W_2, A_1, A_2, t_1^0, t_2^0\}, \alpha_1), (\{W_1, W_2, A_1, A_2, t_1^0, t_2^0\}, c_1), (\{W_1, W_2, A_1, A_2, t_1^0, t_2^1\}, \alpha_1), (\{W_1, W_2, A_1, A_2, t_1^0, t_2^1\}, c_1), (\{W_1, W_2, A_1, A_2, t_1^0, t_2^2\}, \alpha_1), (\{W_1, W_2, A_1, A_2, t_1^0, t_2^2\}, c_1), (\{W_1, W_2, A_1, A_2, t_1^1, t_2^0\}, \alpha_1), (\{W_1, W_2, A_1,$

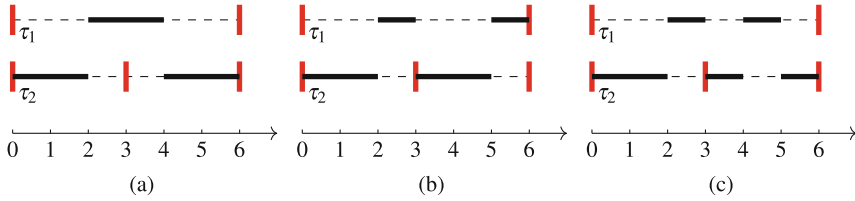


Fig. 7.17 EDF scheduling logics shown in Fig. 7.16. (a) Scheduler 1. (b) Scheduler 2. (c) Scheduler 3

$A_2, t_1^1, t_2^0, c_1), (\{W_1, W_2, A_1, A_2, t_1^1, t_2^1\}, \alpha_1), (\{W_1, W_2, A_1, A_2, t_1^1, t_2^1\}, c_1), (\{W_1, W_2, A_1, A_2, t_1^1, t_2^2\}, \alpha_1), (\{W_1, W_2, A_1, A_2, t_1^1, t_2^2\}, c_1), (\{W_1, W_2, A_1, A_2, t_1^2, t_2^0\}, \alpha_1), (\{W_1, W_2, A_1, A_2, t_1^2, t_2^0\}, c_1), (\{W_1, W_2, A_1, A_2, t_1^2, t_2^1\}, \alpha_1), (\{W_1, W_2, A_1, A_2, t_1^2, t_2^1\}, c_1), (\{W_1, W_2, A_1, A_2, t_1^2, t_2^2\}, \alpha_1), (\{W_1, W_2, A_1, A_2, t_1^2, t_2^2\}, c_1), (\{W_1, W_2, A_1, A_2, t_1^3, t_2^1\}, \alpha_1), (\{W_1, W_2, A_1, A_2, t_1^3, t_2^1\}, c_1), (\{W_1, W_2, A_1, A_2, t_1^3, t_2^2\}, \alpha_1), (\{W_1, W_2, A_1, A_2, t_1^3, t_2^2\}, c_1), (\{W_1, W_2, A_1, A_2, t_1^4, t_2^0\}, \alpha_2), (\{W_1, W_2, A_1, A_2, t_1^4, t_2^0\}, c_2), (\{W_1, W_2, A_1, A_2, t_1^4, t_2^2\}, \alpha_1), (\{W_1, W_2, A_1, A_2, t_1^4, t_2^2\}, c_1), (\{W_1, W_2, A_1, A_2, t_1^5, t_2^0\}, \alpha_2), (\{W_1, W_2, A_1, A_2, t_1^5, t_2^0\}, c_2), (\{W_1, W_2, A_1, A_2, t_1^5, t_2^1\}, \alpha_2), (\{W_1, W_2, A_1, A_2, t_1^5, t_2^1\}, c_2)\}.$

The non-preemptive EDF specification for the RTS is given as follows.

Type 1: $\{\{A_2, w_1^0, t_1^0, t_2^0\}, \{W_1, W_2\}\}$;

Type 2: $\{\{A_1, A_2, t_1^0, t_2^0\}, \alpha_1), (\{A_1, A_2, t_1^0, t_2^1\}, \alpha_1), (\{A_1, A_2, t_1^0, t_2^2\}, \alpha_1), (\{A_1, A_2, t_1^1, t_2^0\}, \alpha_1), (\{A_1, A_2, t_1^1, t_2^1\}, \alpha_1), (\{A_1, A_2, t_1^1, t_2^2\}, \alpha_1), (\{A_1, A_2, t_1^2, t_2^0\}, \alpha_1), (\{A_1, A_2, t_1^2, t_2^1\}, \alpha_1), (\{A_1, A_2, t_1^2, t_2^2\}, \alpha_1), (\{A_1, A_2, t_1^3, t_2^1\}, \alpha_1), (\{A_1, A_2, t_1^3, t_2^2\}, \alpha_1), (\{A_1, A_2, t_1^4, t_2^0\}, \alpha_2), (\{A_1, A_2, t_1^4, t_2^2\}, \alpha_1), (\{A_1, A_2, t_1^5, t_2^0\}, \alpha_2), (\{A_1, A_2, t_1^5, t_2^1\}, \alpha_2)\}.$

7.5.4 Non-Preemptive Earliest-Deadline First Scheduling Sequences

Suppose that we have four real-time tasks with the parameters listed in Table 7.5. By non-preemptive EDF scheduling, the following periodic execution sequence is found:

$$\begin{aligned} s_1c_1c_1\beta_1\alpha_2c_2c_2c_2\beta_2\alpha_3c_3c_3c_3c_3\beta_3\alpha_4c_4\gamma_1c_4c_4c_4\beta_4\alpha_1c_1\gamma_2c_1 \\ \beta_1\alpha_2c_2c_2c_2\beta_2\beta_2s_2c_1c_1\beta_1\alpha_3c_3c_3c_3c_3\beta_3lllls_3c_1c_1\beta_1\alpha_2c_2c_2c_2\beta_2 \\ \alpha_4c_4c_4c_4c_4s_4c_1c_1\beta_1\alpha_3c_3c_3c_3\gamma_2c_3\beta_3\alpha_2c_2c_2c_2\beta_2\gamma_1\alpha_1c_1c_1\beta_1llllllll \end{aligned}$$

with

Table 7.6 Tasks of a large RTS

Task	C_i	\mathbf{T}_i	Task	C_i	\mathbf{T}_i	Task	C_i	\mathbf{T}_i
τ_{11}	1	[40, 40]	τ_{21}	1	[40, 40]	τ_{31}	1	[20, 20]
τ_{12}	1	[30, 30]	τ_{22}	1	[40, 40]	τ_{32}	1	[25, 25]
τ_{13}	2	[40, 40]	τ_{23}	2	[40, 40]	τ_{33}	1	[20, 20]
τ_{14}	1	[50, 50]	τ_{24}	1	[40, 40]	τ_{34}	1	[25, 25]
τ_{15}	1	[30, 30]	τ_{25}	1	[50, 50]	τ_{35}	1	[25, 25]

By the supervisory control of STS, for S1, the controllers allow all the controllable events to occur whenever they are eligible. However, no safe execution sequences with respect to S2 are found.

7.7 Conclusion

STS are a complex DES framework with compact representation. As stated in Chap. 2, [8], and [9], the state explosion problem faced by the SCT community is to some extent managed in STS. The efficiency of the proposed method is due in part to BDD [2] and symbolic computation. In the case of large-scale DES the required computer memory is more likely to be acceptable than the case that an explicit state transition enumeration is employed.

This chapter reports on a unified STS-based framework to model and schedule RTS by addressing conditionally-preemptive and dynamic priority scheduling specifications. A formal constructive method is presented to model an RTS that processes multi-period and sporadic tasks, in which a multi-period task is assigned with a set of possible periods between a minimum period and a maximum period. Based on this framework, the PFCP specifications proposed in [16] are converted to STS specifications in a compact representation that often manages the state explosion problem for practical purposes. As a classical dynamic priority scheduling, partially preemptive or non-preemptive earliest deadline first (EDF) scheduling is also addressed in this chapter. Moreover, there is no technology barrier for designing specifications for LLF scheduling. The PFCP scheduling presented in this chapter is applied to a real-world RTS example.

The dynamic specifications presented in Sect. 7.4 are all state-based, which avoids using memories (specification holons) to record the system behavior. Thus the approach provided in this chapter avoids building a large group of memories, which is normally very laborious for the classical supervisory control of DES.

If other SCT synthesis tools, such as TCT⁵ [19], libFAUDES⁶ [11], Desuma⁷ [13], and Supremica⁸ [1], are able to handle the state-based specifications presented in Sects. 7.3 and 7.4, then they can also be used to find the safe execution sequences of the RTS with respect to dynamic specifications.

References

1. Akesson, K., Fabian, M., Flordal, H., Malik, R.: Supremica-an integrated environment for verification, synthesis and simulation of discrete event systems. In: International Workshop on Discrete Event Systems, pp. 384–385. IEEE (2006)
2. Andersen, H.R.: An Introduction to Binary Decision Diagrams. Lecture Notes, (available online), IT University of Copenhagen (1997). http://web.archive.org/web/20140222052815/http://configit.com/configit_wordpress/wp-content/uploads/2013/07/bdd-eap.pdf
3. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.* **100**(8), 677–691 (1986)
4. Bryant, R.E.: Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.* **24**(3), 293–318 (1992)
5. Chao, W., Gan, Y., Wang, Z., Wonham, W.M.: Modular supervisory control and coordination of state tree structures. *Int. J. Control* **86**(1), 9–21 (2013)
6. Chen, P.C.Y., Wonham, W.M.: Real-time supervisory control of a processor for non-preemptive execution of periodic tasks. *Real-Time Syst.* **23**, 183–208 (2002)
7. Janarthanan, V., Gohari, P., Saffar, A.: Formalizing real-time scheduling using priority-based supervisory control of discrete-event systems. *IEEE Trans. Autom. Control* **51**(6), 1053–1058 (2006)
8. Ma, C., Wonham, W.M.: Nonblocking Supervisory Control of State Tree Structures, vol. 317. Springer-Verlag, Berlin (2005)
9. Ma, C., Wonham, W.M.: Nonblocking supervisory control of state tree structures. *IEEE Trans. Autom. Control* **51**(5), 782–793 (2006)
10. Mok, A.K.: Fundamental design problems of distributed systems for the hard-real-time environment. Ph.D. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts (1983)
11. Moor, T., Schmidt, K., Perk, S.: libFAUDES—An open source C++ library for discrete event systems. In: 2008 9th International Workshop on Discrete Event Systems, pp. 125–130. IEEE (2008)
12. Park, S.J., Cho, K.H.: Real-time preemptive scheduling of sporadic tasks based on supervisory control of discrete event systems. *Inf. Sci.* **178**(17), 3393–3401 (2008)
13. Ricker, L., Lafortune, S., Genc, S.: Desuma: A tool integrating giddes and umdes. In: 8th International Workshop on Discrete Event Systems, pp. 392–393. IEEE (2006)
14. Sha, L., Abdelzaher, T., Cervin, A., Baker, T., Burns, A., Buttazzo, G., Caccamo, M., Lehoczky, J., Mok, A.K.: Real time scheduling theory: a historical perspective. *Real-Time Syst.* **28**(2), 101–155 (2004)
15. Wang, X., Li, Z., Wonham, W.M.: Dynamic multiple-period reconfiguration of real-time scheduling based on timed DES supervisory control. *IEEE Trans. Ind. Inf.* **12**(1), 101–111 (2016)

⁵ The software tool TCT is available at <http://www.control.utoronto.ca/DES>.

⁶ The software libFAUDES incl. luafaudes is available at <https://www.rt.tf.fau.de/FGdes>.

⁷ The software Desuma is available at <https://wiki.eecs.umich.edu/desuma/index.php/DESUMA>.

⁸ The software Supremica is available at <https://supremica.org/>.

16. Wang, X., Li, Z., Wonham, W.M.: Optimal priority-free conditionally-preemptive real-time scheduling of periodic tasks based on DES supervisory control. *IEEE Trans. Syst. Man Cybern. Syst.* **47**(7), 1082–1098 (2017)
17. Wang, X., Li, Z., Wonham, W.M.: Priority-free conditionally-preemptive scheduling of modular sporadic real-time systems. *Automatica* **89**, 392–397 (2018)
18. Wang, X., Li, Z., Wonham, W.M.: Real-time scheduling based on nonblocking supervisory control of state-tree structures. *IEEE Trans. Autom. Control* **66**(9), 4230–4237 (2021)
19. Wonham, W.M., Cai, K.: *Supervisory Control of Discrete-Event Systems*. Monograph Series Communications and Control Engineering. Springer, Berlin (2018)

Chapter 8

Conclusion and Future Work



8.1 Conclusion

As an interdisciplinary approach, SCT-based real-time scheduling and reconfiguration are a newly-identified research topic. This monograph provides several formal real-time systems (RTS) construction approaches based on discrete-event systems (DES) [18, 25], timed DES (TDES) [2], and state-tree structures (STS) [12, 13]. Based on TDES, DES, and STS, this monograph reports unified supervisory control theory (SCT)-based frameworks to build and dynamically schedule/reconfigure RTS. The presented frameworks can be utilized to model an RTS that processes multi-period and sporadic tasks, in which a multi-period task is assigned with a set of possible periods between a *minimum* and a *maximum period*. The proposed *modular models* are taken to be generic entities, which are utilized to model a problem domain such as “hard real-time manufacturing and reconfigurations” and manage its manufacturing production process. A processor could be a robot or an assembly-line worker, and a task could be an industrial operation performed by manufacturing lines.

8.1.1 RTS Modelling Methods

In this monograph, an RTS is denoted by \mathbb{S} . We assume that a set of n RTS tasks processed by a *uni-processor* RTS is represented by a task set $\mathbb{S} = \{\tau_1, \tau_2, \dots, \tau_i, \dots, \tau_n\}$ with $i \in \mathbf{n} := \{1, 2, \dots, n\}$. The main differences among the three SCT modelling frameworks are stated in Table 8.1, in which, for the execution of a real-time task τ_i , both events t and c_i represent that one time unit is utilized to process a real-time task τ_i . Event t is a global tick event and c_i is a local event, both representing task τ_i under execution. As stated in Table 8.1, in the TDES (resp., DES or STS) model, the execution of different tasks is considered as

Table 8.1 SCT-based RTS models

SCT model	τ_i in process	Idle operation	Resource sharing	Resource preemption	Preemption	Hierarchical modelling
TDES	t	t	Y	Y	N	N
DES	c_i	l	N	Y	Y	N
STS	c_i	l	N	Y	Y	Y

the same (resp., different) events. Hence, in the *synchronous product*, they will occur simultaneously (resp., separately). The *idle operation* of processor is represented by t and l in TDES and DES (and also STS), respectively. Moreover, “Y” and “N” in Table 8.1 represent “yes” and “no”, respectively.

For a periodic RTS task τ_i , any of its task execution models can be considered as a special case of a general reconfigurable task model described by

$$\tau_i = (R_i, \mathbf{C}_i, D_i, \mathbf{T}_i)$$

with

- a *release time* R_i ,
- an *execution time interval* \mathbf{C}_i ,
- a *deadline* D_i , and
- a *multi-period* \mathbf{T}_i ,

where R_i and D_i are non-negative integers. As stated in Chap. 4, a multi-period

$$\mathbf{T}_i = [T_i^l, T_i^u] \in \mathbb{N} \times \mathbb{N},$$

is specified by a non-empty interval where the number of time units that elapses between any two successive releases lies within \mathbf{T}_i . Hence a multi-period has a *lower bound* (i.e., shortest one) represented by T_i^l and an *upper bound* (i.e., longest one) represented by T_i^u . In an execution period, only an exact execution time C (resp., period T) in \mathbf{C}_i (resp., \mathbf{T}_i) of task τ_i is actually taken. Only a period T in \mathbf{T}_i of task τ_i is selected. The lower and upper bounds of the execution interval \mathbf{C}_i are the best-case execution time (BCET) and the worst-case execution time (WCET) of task τ_i , denoted by C_i^l and C_i^u , respectively, i.e.,

$$\mathbf{C}_i = [C_i^l, C_i^u].$$

For simplification, we write C_i (resp., T_i , in accordance with Chap. 5) instead of \mathbf{C}_i (resp., \mathbf{T}_i) in the case of $C_i^l = C_i^u$ (resp., $T_i^l = T_i^u$).

A general TDES model is proposed to represent periodic real-time tasks. A task is represented by a TDES

$$\mathbf{G}_i = (Q_i, \Sigma_i, \delta_i, q_{0,i}, Q_{m,i})$$

where Σ_i consists of

- t : the *tick* event,
- γ_i : the release event of τ_i ,
- α_i : the execution of τ_i is *started*, and
- β_i : the execution of τ_i is *finished*.

According to supervisory control of TDES, a monolithic RTS execution model (as the plant) and a user-defined specification are required. However, the proposed TDES-based monolithic RTS execution model may be conservative. This discussion contains three parts:

- **Task model:** Generally, in quite limited cases, resources are available to execute several RTS tasks concurrently (e.g., Fig. 4.7). The reason is that this assumption violates a basic rule for RTS in [11]: all the tasks are independent.
- **Synchronous product:** The timing parameters of RTS tasks are represented by time bounds of events representing release/arrival, starting, and finishing. Hence, the execution of a task is represented by the global *tick* event t . Clearly, in the synchronous product, they will occur simultaneously. As discussed above, this case is very rare in RTS scheduling.
- **Preemption is impossible:** Since the execution of any task is represented by the global *tick* event t , task execution preemption (event t for a task's execution preempts the occurrence of another task) is impossible.

For instance, as stated in Chaps. 5, 6, and 7, based on DES and STS, suppose that we have two tasks τ_1 and τ_2 and their executions are represented by c_1 in Σ_1 and c_2 in Σ_2 , respectively. Moreover, assume that we have two substrings $s_1 = c_1c_1$ and $s_2 = c_2$ under the condition that there exist

$$w_1, v_1 \in \Sigma_1^*, w_1s_1v_1 \in L(\mathbf{G}_1)$$

and

$$w_2, v_2 \in \Sigma_2^*, w_2s_2v_2 \in L(\mathbf{G}_2).$$

Intuitively, the execution of s_1 and s_2 within a uni-processor needs to take three time units. Let $L_1 = \{s_1\} \subset \Sigma_1'^*$ and $L_2 = \{s_2\} \subset \Sigma_2'^*$ with $\Sigma_1' = \{c_1\} \subset \Sigma_1$ and $\Sigma_2' = \{c_2\} \subset \Sigma_2$. Their synchronous product is

$$L_1 || L_2 = \{c_2c_1c_1, c_1c_2c_1, c_1c_1c_2\};$$

it represents all the possible executions correctly. However, according to the TDES model presented in Chap. 4, we have

- $s_1 = tt$,
- $s_2 = t$,
- $L_1 = \{s_1\} \subset \Sigma_1'^*$,

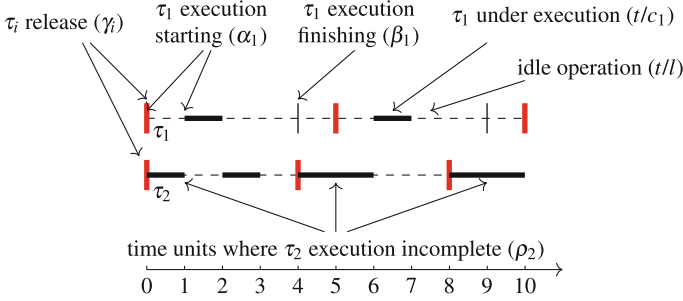


Fig. 8.1 Event assignments in TDES, DES, and STS

- $L_2 = \{s_2\} \subset \Sigma_2^*$,
- $\Sigma_1' = \{t\} \subset \Sigma_1$, and
- $\Sigma_2' = \{t\} \subset \Sigma_2$.

The *synchronous product* of L_1 and L_2 is $L_1 || L_2 = \emptyset$. Evidently, the TDES model cannot actually represent general resource preemption real-time scheduling.

All the SCT frameworks discussed above model the RTS behaviors related to task execution properly. As illustrated in Fig. 8.1, (built on Fig. 1.1a), for any real-time task τ_i with $i \in \mathbb{N}$, (summarized from [21–24]), six behaviors are defined below.

- arrival of task τ_i : in TDES, DES, and STS, represented by event γ_i ,
- execution of τ_i starting: in TDES, DES, and STS, represented by event α_i ,
- execution of τ_i : in TDES, represented by global clock t ; in DES and STS, by event c_i ,
- execution of τ_i preempted by another task τ_j : in DES and STS, represented by event c_j ; no such assumption in TDES,
- execution of τ_i completing: in TDES, DES, and STS, represented by event β_i , and
- execution of τ_i is not *completed* (considering the exact execution time): in DES, represented by event ρ_i .

Assume that in the *Gantt chart* depicted in Fig. 8.1, we have $\tau_1 = (0, [1, 1], 4, [5, 5])$ and $\tau_2 = (0, [1, 2], 4, [4, 4])$, in which τ_2 is associated with a BCET and a WCET being equal to 1 and 2 time units, respectively. Consequently, event ρ_2 occurs at time $t = 1$, $t = 5$, and $t = 9$ since the execution is not completed at the BCET. Clearly, it is trivial to extend it into an STS model, which is ignored in Chap. 7.

Suppose that n tasks are running in an RTS. Its global behavior is represented by a DES/STS modelling framework denoted by \mathbf{G} with a global event set denoted by Σ . The alphabet Σ can be partitioned into *controllable* events and *uncontrollable* events. Formally,

$$\Sigma = \Sigma_{con} \dot{\cup} \Sigma_{unc},$$

with

- $\Sigma_{con} = \{\alpha_i, c_i | i \in \mathbf{n}\}$: the controllable event subset, and
- $\Sigma_{unc} = \{\beta_i, \rho_i, \gamma_i, l | i \in \mathbf{n}\}$: the uncontrollable event subset.

Moreover, Σ is also partitioned into

- $\Sigma_o = \{\gamma_i, \alpha_i, \beta_i | i \in \mathbf{n}\}$: the *operation event set*, and
- $\Sigma_e = \{c_i, l | i \in \mathbf{n}\}$: the *execution event set*.

The occurrence of any event σ in Σ leads the system from one state q to another state q' . Formally, write $\delta(q, \sigma) = q'$. Let $\sigma = c_i$. The execution of a task τ_i during one processor time unit occurs in state q' . Otherwise, letting $\sigma = l$, the processor is in an idle operation for one processor time unit when it stays in state q' .

In DES, the semantics of all the events in Σ_e is similar to the *tick* event defined in TDES [2], i.e., the occurrence of an event c_i or l in DES (or an event t in TDES) represents that a processor time unit is utilized to execute a real-time task or in an idle operation. However, since the events in Σ_e are not represented by a unique global *tick* event t , the synchronous product of several RTS task models over Σ_e will not result in the concurrent execution of real-time tasks. As a consequence, in comparison with [3, 10, 15], and [20], the DES/STS-based RTS model is more general.

Based on the three frameworks and the modelling methodology discussed above, in the literature, RTS tasks are modelled in three different ways:

- RTS execution models (describing WCETs or exact execution times) as plants and other parameter as specifications [5–10, 16, 17, 20, 23],
- the monolithic task behavior as a plant [20, 21] (as stated in Chaps. 4 and 5), and
- individual parameters as modular plants [22–24] (as stated in Chaps. 6 and 7).

Essentially, the cores of all the modelling approaches discussed above are identical: the real-time tasks' behavior is represented by formal languages that are generated by TDES, DES, or holons in STS. Then, SCT is utilized to find out the safe execution sequences. For the scheduling of tasks τ_1 and τ_2 shown in Fig. 8.3 (identical with Fig. 1.1b), a TDES sub-diagram and a DES sub-diagram representing the processor behavior are illustrated in Figs. 8.4 and 8.2, respectively. The states in Figs. 8.4 and 8.2 are marked with the corresponding time units shown in Fig. 8.3. According to Sect. 4.2.2, the states in Figs. 8.4 and 8.2 filled with gray represent that a task is under execution. Notice that state x in Fig. 8.2 is special:

- if the system enters state x by event c_2 , task τ_2 is under execution, or
- if the system enters state x by event α_1 , the execution of task τ_1 starts in the next state.

The latter shows that, after task τ_1 arrives at time $t = 1$, the execution of task τ_1 satisfies one of the following two cases:

- the execution of τ_1 is started at $t = 1$: the processor considers that the execution of task τ_1 is started at $t = 1$, but it is preempted in time interval $[1, 2)$, or

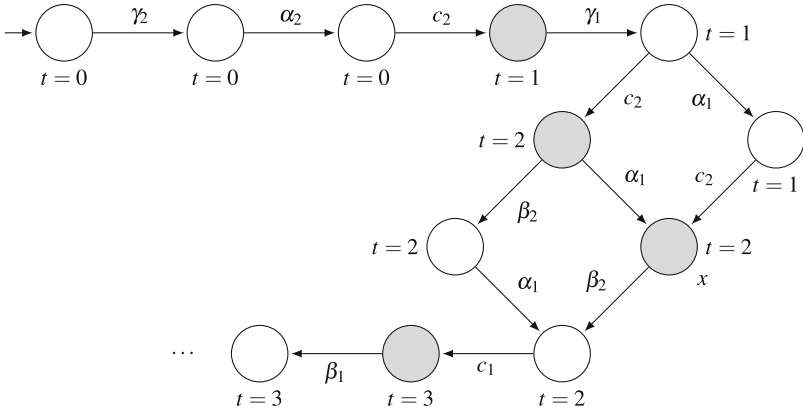
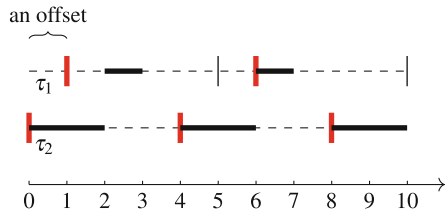


Fig. 8.2 A DES sub-diagram

Fig. 8.3 Real-time scheduling of τ_1 with an offset and τ_2



- the execution of τ_1 is started at $t = 2$: the execution of task τ_1 is delayed to $t = 2$. At the same time, it is under execution immediately.

Actually, in both DES modelling and real-time scheduling, it is not so necessary to consider the first case. For simplicity, in this monograph, we do not distinguish such cases, i.e., we directly fill any state with an entry c_i with gray.

Remarks

1. Notice that the real-time scheduling shown Fig. 8.1 can be represented by DES. However, TDES is not able to describe such a preemptive real-time scheduling sequence.
2. In this monograph, all the DES models can be converted into STS holons. □

In real world, the execution time of a job always varies over time. The exact execution time of a task lies between its BCET and WCET. By addressing the exact execution time of real-time tasks, Chap. 7 presents a modular modelling framework to describe the parameters of real-time tasks, conforming to the pertinent concepts and techniques of DES. For a periodic task τ_i , between its BCET and WCET, two uncontrollable events β_i and ρ_i in the corresponding DES execution model represent that the execution is completed or not. DES will “check” the execution process until event β_i occurs. A sub-diagram is provided in Fig. 8.5. It shows that, at state 1, if the execution of task τ_i is completed, then event β_i occurs. Otherwise, event ρ_i

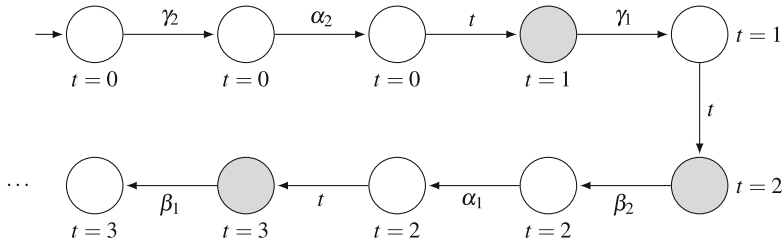


Fig. 8.4 A TDES sub-diagram

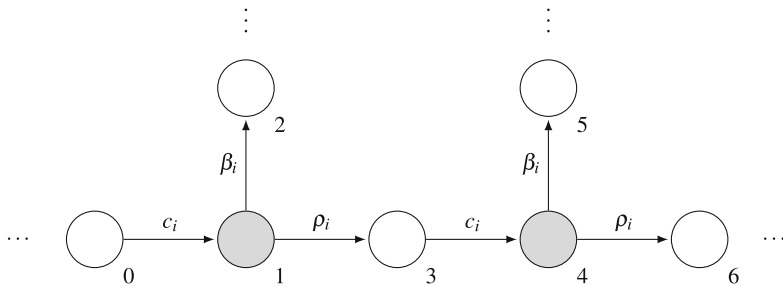


Fig. 8.5 A sub-diagram for exact execution time

occurs, which leads the system from state 0 to 1 for the next time unit execution and checking.

Users are suggested to propose scheduling or reconfiguration requirements according to their will. These scheduling/reconfiguration requirements, either priority-based or not, from the perspective of either processor or individual task, are converted to formal SCT specifications offline. With these specifications assigned, all the safe execution sequences of an RTS can be found in its optimal supervisor.

8.1.2 An Overview of Specifications Describing RTS Scheduling Requirements

In this monograph, we distinguish conditionally-preemptive and dynamic real-time scheduling policies. As stated in Chap. 5, conditionally-preemptive scheduling policies are assigned from the perspective of both the processor and individual tasks. The preemption relations do not change while the tasks are under execution. In the literature, dynamic scheduling policies are more feasible. In [4], the author shows that, among all preemptive scheduling algorithms, the famous *dynamic priority* scheduling earliest deadline first (EDF) is optimal. If there exists a feasible scheduling for a task set, then the scheduling produced by EDF is also feasible.

On the processor level, conditionally-preemptive specifications are represented by a preemption matrix. By defining the preemption relation among any two tasks running in a processor, a preemption matrix can be utilized to describe all the possible fixed-priority (FP) preemption relations and other user-specified preemption relations. Based on this matrix, the corresponding DES specifications are designed accordingly. On the task level, the task preemption relations are depicted by DES specifications directly. Clearly, the presented two general conditional-preemption specifications are utilized to customize scheduling and preemption requirements conditionally.

The dynamic priority specifications presented in Chap. 7 are state-based, which avoids using memories (specification holons) to record the system behavior. Thus the provided approach avoids building a large group of memories, which is normally very laborious for the classical supervisory control of DES.

To the best of our knowledge, by modelling RTS in a hierarchical (vertical) and concurrent (horizontal) structure, Chap. 7 is an attempt to provide an SCT-based framework such that an RTS can be scheduled by addressing both conditionally-preemptive and dynamic scheduling requirements. As seen, the presented scheduling framework can find the optimal behavior (all the safe execution sequences) of an RTS by the supervisory control of STS; a few quantitatively optimal schedule sequences are selected, from perhaps a large number of safe execution sequences, which rank at the top according to some specified optimality criteria.

If other SCT synthesis tools, such as TCT¹ [25], libFAUDES² [14], Desuma³ [19], and Supremica⁴ [1], are able to handle the state-based specifications presented in Sects. 7.3 and 7.4, then they can also be used to find the safe execution sequences of the RTS with respect to dynamic specifications.

From the perspective of SCT, an approach that can speed up the supervisor synthesis reduces the number of states in the plant and specification. For TDES and DES, the presented synthesis speeding up approach can be applied to Chaps. 5 and 6. This monograph divides the calculations into three steps. Each step considers different specifications as follows.

- Step 1: from the perspective of processors, matrix-based conditional-preemption specifications are taken into account.
- Step 2: from the perspective of individual tasks, WCET-based conditional-preemption specifications are considered.
- Step 3: other user defined specifications are touched upon.

According to [12] and [13], the STS framework (rooted in binary decision diagrams) is well-developed to manage the state explosion problem. Hence, the “speeding up”

¹ The software tool TCT is available at <http://www.control.utoronto.ca/DES>.

² The software libFAUDES incl. luafaudes is available at <https://www.rt.tf.fau.de/FGdes>.

³ The software Desuma is available at <https://wiki.eecs.umich.edu/desuma/index.php/DESUMA>.

⁴ The software Supremica is available at <https://supremica.org/>.

approach is not so necessary for STS-based real-time scheduling and reconfiguration.

8.2 Future Work

Future work includes two parts: supervisory control theory (SCT) and real-time scheduling.

For SCT, a possible topic is *hierarchical (nested) supervisory control* of STS. With the state explosion problem managed, STS are a powerful framework to model complex dynamic systems in a compact and natural way. In an STS, such a system is hierarchically abstracted to be superstates with *layered internal structures*. In our future work, we will focus on hierarchical supervisory control of STS. The main idea is: we decompose an STS into a set of *STS nests* (the largest flat fragments) in the hierarchical layers automatically; thereafter, at any fixpoint, an STS nest tracks the system dynamics partially. Finally, given an STS framework with multiple STS nests, without tracking its *global dynamics* to synthesize the optimal behavior, a nested optimal nonblocking supervisor is obtained. It is interesting to investigate the modelling and supervisory control of timed STS by building on their nested structure.

For real-time scheduling, SCT-based multi-processor scheduling/reconfiguration policies and their *industrial implementations* are worth further exploration. Possible research topics include: soft scheduling and reconfiguration, low-power scheduling and reconfiguration, resource constrained scheduling and reconfiguration, resource sharing scheduling and reconfiguration, *job skipping reconfiguration*, *resource allocation* during scheduling and reconfiguration, *feedback scheduling* and reconfiguration, *multi-class resource scheduling* and reconfiguration, and *distributed scheduling* and reconfiguration.

References

1. Akesson, K., Fabian, M., Flordal, H., Malik, R.: Supremica—an integrated environment for verification, synthesis and simulation of discrete event systems. In: International Workshop on Discrete Event Systems, pp. 384–385. IEEE (2006)
2. Brandin, B.A., Wonham, W.M.: Supervisory control of timed discrete-event systems. IEEE Trans. Autom. Control **39**(2), 329–342 (1994)
3. Chen, P.C.Y., Wonham, W.M.: Real-time supervisory control of a processor for non-preemptive execution of periodic tasks. Real-Time Syst. **23**, 183–208 (2002)
4. Dertouzos, M.L.: Control robotics: the procedural control of physical processes. In: IF IP Congress, pp. 807–813 (1974)
5. Devaraj, R., Sarkar, A., Biswas, S.: Fault-tolerant preemptive aperiodic RT scheduling by supervisory control of TDES on multiprocessors. ACM Trans. Embed. Comput. Syst. **16**(3), 1–25 (2017)

6. Devaraj, R., Sarkar, A., Biswas, S.: Fault-tolerant scheduling of non-preemptive periodic tasks using SCT of timed DES on uniprocessor systems. *IFAC-PapersOnLine* **50**(1), 9315–9320 (2017)
7. Devaraj, R., Sarkar, A., Biswas, S.: Real-time scheduling of non-preemptive sporadic tasks on uniprocessor systems using supervisory control of timed DES. In: *American Control Conference*, pp. 3212–3217. IEEE (2017)
8. Devaraj, R., Sarkar, A., Biswas, S.: Supervisory control approach and its symbolic computation for power-aware RT scheduling. *IEEE Trans. Ind. Inf.* **15**(2), 787–799 (2019)
9. Janarthanan, V., Gohari, P.: Multiprocessor scheduling in supervisory control of discrete-event systems framework. *Control Intell. Syst.* **35**(4), 360 (2007)
10. Janarthanan, V., Gohari, P., Saffar, A.: Formalizing real-time scheduling using priority-based supervisory control of discrete-event systems. *IEEE Trans. Autom. Control* **51**(6), 1053–1058 (2006)
11. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM* **20**(1), 46–61 (1973)
12. Ma, C., Wonham, W.M.: *Nonblocking Supervisory Control of State Tree Structures*, vol. 317. Springer, Berlin (2005)
13. Ma, C., Wonham, W.M.: Nonblocking supervisory control of state tree structures. *IEEE Trans. Autom. Control* **51**(5), 782–793 (2006)
14. Moor, T., Schmidt, K., Perk, S.: libFAUDES—An open source C++ library for discrete event systems. In: *2008 9th International Workshop on Discrete Event Systems*, pp. 125–130. IEEE (2008)
15. Park, S.J., Cho, K.H.: Real-time preemptive scheduling of sporadic tasks based on supervisory control of discrete event systems. *Inf. Sci.* **178**, 3393–3401 (2008)
16. Park, S.J., Cho, K.H.: Supervisory control for fault-tolerant scheduling of real-time multiprocessor systems with aperiodic tasks. *Int. J. Control* **82**(2), 217–227 (2009)
17. Park, S.J., Yang, J.M.: Supervisory control for real-time scheduling of periodic and sporadic tasks with resource constraints. *Automatica* **45**(11), 2597–2604 (2009)
18. Ramadge, P.J., Wonham, W.M.: Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.* **25**(1), 206–230 (1987)
19. Ricker, L., Lafortune, S., Genc, S.: Desuma: A tool integrating giddes and umdes. In: *8th International Workshop on Discrete Event Systems*, pp. 392–393. IEEE (2006)
20. Wang, X., Li, Z., Wonham, W.M.: Dynamic multiple-period reconfiguration of real-time scheduling based on timed DES supervisory control. *IEEE Trans. Ind. Inf.* **12**(1), 101–111 (2016)
21. Wang, X., Li, Z., Wonham, W.M.: Optimal priority-free conditionally-preemptive real-time scheduling of periodic tasks based on DES supervisory control. *IEEE Trans. Syst. Man Cybern. Syst.* **47**(7), 1082–1098 (2017)
22. Wang, X., Li, Z., Wonham, W.M.: Priority-free conditionally-preemptive scheduling of modular sporadic real-time systems. *Automatica* **89**, 392–397 (2018)
23. Wang, X., Li, Z., Moor, T.: SCT-based priority-free conditionally-preemptive scheduling of modular real-time systems with exact task execution time. *Discrete Event Dyn. Syst. Theory Appl.* **29**, 501–520 (2019)
24. Wang, X., Li, Z., Wonham, W.M.: Real-time scheduling based on nonblocking supervisory control of state-tree structures. *IEEE Trans. Autom. Control* **66**(9), 4230–4237 (2021)
25. Wonham, W.M., Cai, K.: *Supervisory Control of Discrete-Event Systems*. Monograph Series Communications and Control Engineering. Springer (2018)

Glossary

$[P]$	Predicate transformer to compute $\sup^{\mathcal{C}^2}(P)$
$\mathcal{B}(ST)$	Set of all basic-state-trees of ST
$\mathcal{C}^2(P)$	Family of weakly controllable and coreachable subpredicates of P
$\mathcal{C}(E)$	Family of controllable sublanguages of E
\mathcal{E}	Expansion function of a superstate
\mathcal{E}^*	Reflexive and transitive closure of \mathcal{E}
$\mathcal{E}^+(x)$	Unfolding of $\mathcal{E}(x)$
\mathcal{H}	Family of holons
\mathbb{S}	Real-time system
$\mathcal{ST}(ST)$	Set of all sub-state-trees of ST
\mathcal{T}	Type function of a given state
$\mathcal{V}(T)$	Key leaf state set of state-tree T
W_i	Worst-case response time of τ_i
f_σ	Control function for event σ
l_σ	Lower time bound for event σ
q_0	Initial state of generator \mathbf{G}
$\sup^{\mathcal{C}}(E)$	(unique) Supremal element within $\mathcal{C}(E)$
$\sup^{\mathcal{C}^2} \mathcal{P}(P)$	(unique) Supremal element within $\mathcal{C}^2 \mathcal{P}(P)$
u_σ	Upper time bound for event σ
C_i	Worst-case execution time of τ_i
\mathbf{C}_i	Exact execution time interval of τ_i
D_i	Deadline of τ_i
$CR(\mathbf{G}, P)$	Coreachability predicate of P
E	Specification language
$Elig_{\mathbf{G}}$	Largest eligible state-tree
H^{C_i}	Higher level holon describing the execution time of τ_i
H^{D_i}	Holon describing deadline of τ_i
H^{W_i}	Lower level holon describing the execution time of τ_i
H^{T_i}	Holon describing the release time and periods of τ_i

$J_{i,j}$	j -th job of real-time task τ_i
$L(\mathbf{G})$	Closed behavior of \mathbf{G}
$L_m(\mathbf{G})$	Marked behavior of \mathbf{G}
$\overline{L_m(\mathbf{G})}$	(prefix) Closure of $L_m(\mathbf{G})$
$M_\sigma(P)$	Weakest liberal precondition of P
$Next_{\mathbf{G}}$	Largest next state-tree
P	Predicate
$Pred(\mathbf{ST})$	Set of all predicates on $\mathcal{B}(\mathbf{ST})$
$Pwr(\Sigma)$	Power set of Σ
Q	State set of generator \mathbf{G}
$R(\mathbf{G}, P)$	Reachability predicate of P
R_i	Release time of τ_i
T_i	Period of τ_i
T_σ	Timer interval for event σ
U	Processor utilization
V/\mathbf{G}	\mathbf{G} under supervision of V
X_0	Initial state set
X_m	Terminal state set
X_E	External state set
X_I	Internal state set
$X_{\mathcal{A}}(x)$	State aggregation bonded with superstate x
\mathbf{A}	Preemption matrix
\mathbf{G}	Generator or state-tree structure
\mathbf{ST}	State-tree
\mathbf{ST}_0	Initial state-tree
\mathbf{ST}_m	Marker state-tree set
\mathbf{T}_i	Multi-period of τ_i
δ	Transition function
δ_{BI}	Incoming boundary transitions
δ_{BO}	Outgoing boundary transitions
ϵ	Empty string
τ	Real-time task
Γ	Global backward transition function
Δ	Global forward transition function
Σ	Event set of generator \mathbf{G}
Σ^*	Set of strings over Σ
Σ_{act}	Activity event set of a timed discrete-event system
Σ_{con}	Controllable event set
Σ_e	Execution event set of generator \mathbf{G}
Σ_o	Operation event set of generator \mathbf{G}
Σ_{rem}	Remote event set
Σ_{spe}	Prospective event set
Σ_{unc}	Uncontrollable event set
Σ_B	Boundary event set
Σ_I	Internal event set
Φ	Set of control patterns

Index

A

Absolute clock time, 55
Absolute deadline, 57
Absolute release time, 57
Abstraction, 31
Active event, 71
Active task, 60
Activity-loop-free, 27
Activity set, 24
Activity transition graph, 24
Ad hoc, 71
Adjacent higher level, 34
Agent, 50
Aggregation, 31
Alphabet, 17
Ancestor, 39
AND-adjacent, 39
AND-component, 32
AND superstate, 32
Aperiodic task, 5
Arbitrary union, 22
Arrival, 74
Arrival time, 134
Ascending order, 100
Asynchronous real-time system, 60
Automaton, 2

B

Backward transition, 43
Basic-state-tree, 40
Best-case execution time, 55
Binary decision diagram, 164
Binary decision diagram node, 179

Binary decision diagram variable, 51
Binary value, 52
Boolean function, 3
Bottom element, 47
Boundary consistency, 43
Boundary event set, 34
Boundary transition structure, 34
Brandin-Wonham framework, 24
Busy time, 98

C

Cartesian product, 25
Catenation, 17
Characteristic function, 47
Child-state-tree, 39
Closed behavior, 18
Cold start, 67
Compact representation, 3
Complete lattice, 47
Complex discrete event system, 163
Computational complexity, 52
Concurrent (horizontal) structure, 163
Concurrent transition structure, 163
Control function, 49
Controllability, 23
Controllable event, 2
Controllable event subset, 102
Controllable subpredicate, 49
Controller, 2
Control logic, 3
Control map, 22
Control pattern, 21
Coreachability predicate, 48

Correct functioning, 95
 Current status, 50
 Cyclic executive, 71

D

Deadline, 4
 Deadline monotonic scheduling, 6
 Decision maker, 50
 Descendant, 39
 Descending order, 100
 Directed acyclic graph, 164
 Discrete-event system, 1
 Discrete-event system diagram, 141
 Discrete-event system synthesis tool, 21
 Disjoint union, 32
 Distributed scheduling, 197
 Double circle, 18
 Dynamic reconfiguration, 72
 Dynamic system, 2

E

Earliest deadline first, 60
 Earliest deadline first scheduling sequence, 182
 Elastic period, 7
 Eligible event, 26
 Empty string, 17
 Enabled event, 26
 Encoding process, 179
 Entering arrow, 18
 Entrance, 26
 Event occurrence, 126
 Event set, 2
 Exact execution time, 55
 Exclusive-or, 32
 Execution completion, 71
 Execution process, 138
 Execution starting, 71
 Execution time, 55
 Exit, 43
 Exosystem, 38
 Expansion function, 32
 External state set, 34
 External structure, 34

F

False, 164
 Feasibility analysis, 6
 Feedback scheduling, 197
 Finite event set, 17
 Finite sequence, 17
 Finite state set, 17

First-release time, 55
 Fixed-priority scheduling, 58
 Fixed period, 91
 Fixed priority, 5
 Forcible event, 29
 Forcible event set, 29
 Formal language, 2
 Formal specification, 133
 Forward transition, 43
 Free time, 98

G

Gantt chart, 58
 Garbage collection, 5
 Generator, 17
 Global clock, 25
 Global dynamic, 197
 Global forward transition function, 42

H

Hard deadline, 4
 Hierarchical finite state machine, 31
 Hierarchical (vertical) structure, 163
 Hierarchical (nested) supervisory control, 197
 Hierarchical transition relation, 38
 Hierarchical transition structure, 163
 Higher level, 166
 Highest priority, 58
 Highest processor utilization, 72
 Holon, 33
 Horizontal transition relation, 38
 Hyper-period, 8

I

Idle, 74
 Idle operation, 5
 Incoming arrow, 35
 Incoming boundary transition, 34
 Independent task, 53
 Industrial implementation, 197
 Ineligible event, 27
 Infinite stream, 136
 Infinite string, 3
 Initial state, 17
 Initial state-tree, 42
 Initial state set, 34
 Internal event set, 34
 Internal state set, 34
 Internal structure, 34
 Internal transition structure, 34
 Inverse image function, 19
 Irregular arrival time, 131

J

Job, 55
 Job skipping reconfiguration, 6

K

Key leaf state set, 42

L

Language, 18
 Largest eligible state-tree, 43
 Largest next state-tree, 43
 Layer, 3
 Layered internal structure, 197
 Leaf state, 40
 Leaf state set, 42
 Least common multiple, 8
 Least laxity first, 60
 Length of string, 18
 Local coupling, 43
 Local transition, 33
 Lower bound, 73
 Lower-level holon, 166
 Lower time bound, 25
 Lowest priority, 5

M

Manufacturing cell, 123
 Marked behavior, 18
 Marked language, 126
 Marker state, 17
 Marker state-tree set, 42
 Marker state set, 25
 Marking nonblocking supervisory control, 22
 Maximally permissive predicate, 164
 Maximum period, 189
 Minimally restricted controller, 112
 Minimum period, 189
 Model checker, 7
 Modular model, 189
 Motor network, 85
 Multi-class resource, 197
 Multi-period periodic task, 56
 Multi-period task, 134
 Mutual exclusion, 173

N

Natural projection, 19
 n -bit binary vector, 179
 Nearest common ancestor, 39
 Nonblocking specification, 106

Nonblocking supervisory control, 22
 Non-empty state set, 173
 Non-negative integer, 134
 Non-preemptive earliest deadline first scheduling, 183
 Non-preemptive earliest deadline first specification, 182
 Non-preemptive scheduling, 5
 Non-preemptive scheduling specification, 71
 Non-reconfigurable task, 74
 Non-repetitive task, 57
 Non-schedulable real-time system, 71

O

Offline, 67
 Offset, 4
 Optimal behavior, 49
 OR-component, 32
 OR superstate, 32
 Outgoing arrow, 35
 Outgoing boundary transition, 34
 Overload, 67

P

Parameter, 132
 Partially non-preemptive earliest deadline first specification, 178
 Partially non-preemptive specification, 164
 Partially preemptive earliest deadline first specification, 177
 Partial state transition function, 17
 (Partial) transition function, 102
 Pending event, 27
 Period, 56
 Periodic task, 56
 Plant, 17
 Power set, 19
 Predicate, 47
 Preemption matrix, 98
 Preemption policy, 99
 Preemption relation, 151
 Preemption time, 98
 Preemptive earliest deadline first scheduling, 95
 Preemptive earliest deadline first specification, 182
 Preemptive release map, 114
 Preemptive scheduling, 5
 Preemptive scheduling map, 114
 Prefix, 18
 (Prefix) closed language, 126
 Prefix closure, 18

Priority-free conditionally-preemptive real-time scheduling, 106
 Priority-free conditionally-preemptive specification, 106
 Priority-free scheduling, 98
 Processing period, 170
 Processing time, 55
 Processor behavior, 101
 Processor time unit, 96
 Processor utilization, 57
 Processor utilization interval, 72
 Prohibitible event set, 29
 Prospective event set, 25

R

Ramdge-Wonham framework, 17
 Random disturbance, 6
 Rate-monotonic scheduling, 6
 Reachability predicate, 48
 Real-time reconfiguration, 67
 Real-time scheduling, 3
 Real-time system, 1
 Real-time task, 3
 Real-time task execution, 126
 Reconfigurable real-time system, 85
 Reconfiguration scenario, 67
 Reflexive and transitive closure, 38
 Regular arrival time, 131
 Regular language, 126
 Relative deadline, 56
 Remote event set, 25
 Resource, 131
 Resource-sharing task, 53
 Resource allocation, 197
 Response time, 95
 Root state, 38
 Running time, 98

S

Safe execution sequence, 71
 Safe release sequence, 114
 Schedulability, 2
 Scheduling algorithm, 5
 Scheduling map, 89
 Scheduling policy, 99
 Selfloop, 20
 Shortest period, 72
 Simple state, 32
 Smallest laxity, 64
 Soft deadline, 4
 Source state, 34
 Specification language, 22

Sporadic task, 56
 State aggregation, 41
 State-based specification, 173
 Statechart, 31
 State explosion problem, 163
 State feedback control, 181
 State set, 102
 State space, 38
 State-transition structure, 2
 State-tree, 38
 State-tree structure, 31
 State-tree structure nest, 197
 State-tree structure specification, 164
 Storage space, 164
 Stronger, 47
 Structured state set, 38
 Structured state space, 163
 Sublanguage, 22
 Sub-state-tree, 40
 Superstate, 31
 Supervisor, 2
 Supervisor synthesis, 52
 Supervisory control, 21
 Supervisory control of state-tree structure, 163
 Supervisory control theory, 1
 Supremal element, 22
 Synchronized specification, 101
 Synchronous product, 19
 Synchronous real-time system, 60
 Synchronous task set, 4
 System behavior, 2
 System start-up, 4

T

Target state, 34
 Task arrival, 71
 Task behavior, 132
 Task-centered conditional-preemption specification, 107
 Task execution model, 138
 Task release, 71
 TCT, 21
 TCT operation, 108
 Temporal characteristic, 71
 Terminal nodes, 164
 Terminal state, 166
 Terminal state set, 34
 Tick down, 25
 Tick event, 25
 Time bounds, 24
 Timed discrete-event system, 2
 Timed discrete-event system synthesis tool, 28
 Timed transition graph, 29

Time difference, 168
Time interval, 58
Timer, 25
Timer interval, 25
Timing requirement, 71
Top-down state-based modelling framework, 163
Top element, 47
Traditional periodic task, 134
Transition graph, 18
Transition structure, 3
True, 164
Truth-value, 47
Truth table, 52
TTCT, 28
Type function, 32

U
Unboundedly postponed execution, 5
Uncontrollable event, 2

Uncontrollable event subset, 102
Uncontrollable path, 49
Unfolding, 38
Uni-processor real-time system, 138
Upper bound, 73
Upper time bound, 25

V
Vector, 52

W
Weakest liberal precondition, 48, 49
Weakest predicate, 48
Weakly controllable and coreachable behavior, 49
Well-formed, 40
Work, 74
Worst-case execution time, 55
Worst-case response time, 96