DATA ABSTRACTION METHOD FOR MODEL CHECKING OF REAL-TIME SYSTEMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MUSTAFA DURSUN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

FEBRUARY 2015

Approval of the thesis:

DATA ABSTRACTION METHOD FOR MODEL CHECKING OF REAL-TIME SYSTEMS

submitted by **MUSTAFA DURSUN** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Gülbin Dural Ünver
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Gönül Turhan Sayan
Head of Department, **Electrical and Electronics Engineering** _____

Prof. Dr. Semih Bilgen
Supervisor, **Electrical and Electronics Eng. Dept., METU** _____

Examining Committee Members:

Assoc. Prof. Dr. Cüneyt Bazlamaçcı
Electrical and Electronics Engineering Department, METU _____

Prof. Dr. Semih Bilgen
Electrical and Electronics Engineering Department, METU _____

Prof. Dr. Ali Doğru
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Umut Orguner
Electrical and Electronics Engineering Department, METU _____

Assist. Prof. Dr. Ö. Özgür Tanrıöver
Computer Engineering Department, Ankara University _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: MUSTAFA DURSUN

Signature :

ABSTRACT

DATA ABSTRACTION METHOD FOR MODEL CHECKING OF REAL-TIME SYSTEMS

Dursun, Mustafa
Ph.D., Department of Electrical and Electronics Engineering
Supervisor : Prof. Dr. Semih Bilgen

February 2015, 56 pages

Model checking consists of automatic techniques for verifying whether a specified formal property holds for a specific state in a given finite-state model of a system. A major limitation of model checking arises in modeling infinite state systems. This limitation is the main obstacle for model checking of real time systems, due to the need for verifying real time constraints and the necessity of considering infinite data domains. Timed automata models are used to successfully cater for temporal behavior in modeling real time constraints. Abstracting infinite data sets with finite representations is mandatory for feasibility of model checking. In this study, we present an abstraction method for data which is collectively produced by a set of concurrent, asynchronous and periodic tasks in a real time system. The proposed method maps the infinite data domain to a finite one by taking temporal dependencies into account, and models the data with a finite state automaton. Proof of concept is provided with a case study that implements the proposed technique on a multi-sensor data aggregation problem.

Keywords: Formal Verification, Model checking, Data Abstraction, Timed Automata

ÖZ

GERÇEK ZAMANLI SİSTEMLERİN MODEL DENETİMİ İÇİN VERİ SOYUTLAMA YÖNTEMİ

Dursun, Mustafa
Doktora, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Semih Bilgen

Şubat 2015 , 56 sayfa

Model irdeleme, sonlu durum sistemlerinin doğrulanmasında, sistem modeli ve biçimsel özellikler kullanılarak belli bir durumda belli bir özelliğin geçerli olup olmadığının sistematik biçimde irdelenmesini sağlayan otomatik teknikleri içerir. Model irdelemenin başlıca sınırlaması sonsuz durumlu sistemlerin modellenmesinde ortaya çıkmaktadır. Bu sınırlama, gerçek zaman kısıtlamalarının doğrulanması ve sonsuz veri tanım kümelerinin göz önünde bulundurulması zorunluluğundan dolayı, model irdelemenin gerçek zamanlı sistemlerde uygulanabilmesi için esas engeldir. Zamanlı otomatlar zamansal davranış üzerindeki gerçek zaman kısıtlarının modellenmesinde başarıyla kullanılmaktadır. Sonsuz veri kümelerinin soyutlanarak sonlu betimlenmesi model irdelemenin uygulanabilirliği bakımından zorunludur. Bu çalışmada, eşzamanlı, asenkron ve periyodik bir görev kümesi tarafından müşterek olarak üretilen veri için bir soyutlama yöntemi sunulmaktadır. Önerilen yöntem ile, üretilen verilerin zamansal bağımlılıkları dikkate alınarak sonsuz bir veri tanım kümesi sonlu bir veri tanım kümesi ile eşlenmekte, ve veri bir sonlu durum otomatı ile modellenmektedir. Önerilen yöntemin uygulanabilirliği, çok algılayıcı bir sistemde veri bütünleştirme problemi üzerinde gösterilmiştir.

Anahtar Kelimeler: Biçimsel Doğrulama, Model İrdeleme, Veri Soyutlama, Zamanlı

To my wife Eda, and my son Kemal

ACKNOWLEDGMENTS

I would like to thank my supervisor Professor Semih Bilgen for his constant support and guidance. It was a great honor to work with him for the last twelve years and our cooperation influenced my academical and world view highly. I also would like to thank Professor Lui Sha for his support and guidance on my stay at University of Illinois at Urbana Champaign. While away from my home, he not only supported me on my research but also provided that I feel welcome. He also motivated and influenced me highly in scientific context.

A lot of people influenced and supported this work scientifically and their contribution were most valuable for me. Members of my thesis committee Professor Ali Doğru and Assoc. Prof. Cüneyt Bazlamçcı always gave valuable feedback for the progress of this work, and were not hesitant to warn me of the shortcomings or risks of my work. Assoc. Prof. Umut Orguner also provided valuable feedback for the case study of this research.

I would like to thank my superiors at ASELSAN for their support and tolerating me time for the Phd. studies

My family also provided invaluable support for this work. I am grateful to my mother, Hatice Çifter, and especially my wife,Eda Altuntaş, she always make me feel loved and cared.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURES

LIST OF ALGORITHMS

ALGORITHMS

LIST OF ABBREVIATIONS

ACTL	Action Computation Tree Logic
CTL	Computation Tree Logic
IMF	Information Matrix Fusion
LTL	Linear Temporal Logic
MSDF	Multi Sensor Data Fusion
OOST	Out of Sequence Track
TA	Timed Automata
TCTL	Timed Computation Tree Logic

CHAPTER 1

INTRODUCTION

Formal verification aims to prove or disprove the correctness of the design of a system described in a mathematical formalism according to a property or a set of properties. Formal verification with model checking techniques are based on models describing the system behavior in a mathematically precise and unambiguous manner, and formalizing the property to be checked [1]. Given a finite-state model of a system and a formal property, model checking aims to systematically determine whether this property holds for (a given state in) that model [1]. Prerequisites of model checking are formal specification of the system behavior with a finite state model, and formal specification of the properties to be checked with a property specification language. A model checking technique explores the mathematical model of the system exhaustively, and checks all explored states of the system model in order to prove that a given property is satisfied. The applicability of model checking is limited by the requirement that system behavior be modeled by a finite number of states.

In a real-time system the correctness of system behavior depends not only on the correctness of the output, but also the time at which the output is produced [3]. A task is the primary unit of computation in a real-time system. A task performs computation on inputs to produce computational results conveyed to the physical world as system outputs or to other tasks as inputs. The computation time of a task is finite and response time is the elapsed time from the task release time to the finishing time of computation [4]. Computational results of a task depend on the temporal constraints on the task such as the response time and specified deadlines. Therefore, both data and temporal characteristics have to be specified in the formal model of a real time

system to apply model checking. Real-time constraints and multivalued data variables over an infinite domain [7] necessitate the use of finite abstract system models for model checking, which obstructs the applicability of model checking for real time systems. Therefore, specification of time and data in the system model necessitates abstraction to construct finite abstract system model.

Modeling the timeliness of a system is possible with timed modeling formalisms, which are augmented with clocks to express timeliness. The problem of infiniteness caused by specification of time has two aspects. Firstly, specification of time has to be finite and progress of time must be restricted by some conditions [14]. Since real-time systems have bounded response time, period, computation time and deadline, it is possible to restrict progress of time in system model with bounded and finite clock variable values. Secondly, even if the progress of time is restricted in the system model, exploring global states of system by taking timeliness into account may cause infiniteness. Timed automata [9], a well-accepted model for representing and analyzing real-time systems, presents a solution to this problem by using region graph construction [10]. Region graph construction abstracts behaviors of timed automata by constructing a finite automaton, region automaton.

The problem of infiniteness caused by specification of data over infinite domain in the system model requires data abstraction [11]. Data abstraction maps the concrete data values to abstract data values over a finite domain according to an abstraction mapping. In real time systems data values depend on the real time constraints of the system, therefore data has to be abstracted according to the temporal dependencies. These dependencies are complex when concurrency and asynchrony exist

Real time systems are usually highly concurrent and asynchronous, and some computational results are collectively produced by a set of concurrent, asynchronous and periodic tasks, which process input data from different sources via a common operation. Due to concurrency and asynchrony, out of order execution of tasks is possible. If collective execution of the task set is not order independent, out of order execution of tasks lead to the out of sequence processing of input data, hence task functionality may be severely degraded. For reliable model checking, the finite state system model must correctly represent such temporal characteristics of concurrent and asyn-

chronous tasks over infinite data domains.

1.1 The Objective of the Study, Research Questions and Scope of the Study

The objective of this study is to propose a data abstraction method for infinite domain data collectively generated by a set of concurrent, asynchronous and periodic tasks in a real time system. Hence, the two fundamental research questions to be addressed are:

1. Determination of an abstract set of data values;
2. Constructing a mapping between concrete data values and the determined abstract data set, preserving model characteristics for model checking.

The proposed method abstracts data from an infinite concrete domain according to a categorization of "usefulness". Hence the abstract data domain consists of two values; acceptable and erroneous. While acceptable data value represents no degradation in usefulness of data generated by a set of multiple concurrent producers, that is, computational results of a concurrent set of tasks, erroneous data value represents degraded usefulness of the produced data. To determine the usefulness of the computational result of a task set, characteristics of the common operations that the set of tasks set perform to generate the computational result is important. If the common operation of the task set is not order independent, than the process sequences of the input data in the task set affect usefulness of the computational result. Concurrency and asynchrony of the task set lead to the possibility of out of sequence data processing, which causes degradation in the usefulness of the computational result. On the other hand, having mechanisms to provide resiliency against out of sequence data processing to eliminate the degradation in the usefulness of computational result is a requirement of real time systems [6]. In this study, we restrict the scope of the abstraction method to systems that have mechanisms to eliminate degradation in the usefulness of computational results when in sequence input data is processed in correct sequence. In this respect, abstraction method maps concrete data domain to abstract data domain according to the execution order of tasks and thereby input data process sequences.

While out of sequence process of input data causes erroneous data value, acceptable data value is acquired by in sequence process of input data.

Using execution order of tasks in the abstraction mapping enables removing absolute time and input data values from the system model. On the other hand, this mapping requires specification of execution orders of tasks, thereby input data process sequences, according to the abstract data values in the system model. Therefore, a relation between abstract data values and input data process sequence has to be defined according to the common operation characteristic that is performed by the task set. Since the computational result is produced by periodic real time tasks and common operation has mechanisms to eliminate degradation in usefulness, it is possible to specify execution orders of tasks with finite states and transitions according to the abstraction mapping. Data abstraction method abstracts the computational result data with finite state automaton whose states represents the usefulness of computational result and input data process sequences according to the usefulness of computational result.

1.2 Organization of the Document

The remainder of this dissertation is organized as follows: Chapter 2 briefly reviews the relevant literature. Chapter 3 describes the reference system model and sequentialization of concurrency in the system model into the input data process sequences in the first section. The proposed data abstraction mapping and algorithm is introduced in the section of Chapter 3. That chapter also presents a rigorous analysis of certain properties of and relations between the constituents of the proposed method. Chapter 4 is devoted to a case study that illustrates implementation of the proposed abstraction technique in the context of a multi-sensor data aggregation problem. Finally, Chapter 5 concludes the dissertation and summarizes the contributions and limitations of this research. Possibilities for further study are also provided in this chapter.

After the bibliography, three appendices which consist of (A) data abstraction algorithm procedures are presented.

CHAPTER 2

LITERATURE SURVEY

2.1 Real-Time Systems

A real-time system is a computer system whose correctness depends not only on the output, but also the time at which the output is produced [3]. Real-time systems are inherently concurrent because they have to model the parallelism that exists in the real-world objects that they are monitoring and controlling [5]. Many real-time systems are reactive to control factories, plants, transportation systems, cars, and a wide variety of everyday objects. To achieve this, a real-time system collects data and produces results about the state of controlled object, then generates actuation commands according to the produced results to control the state of controlled object. The state of controlled object is defined by state variables.

A real-time system is usually modeled as a set of concurrent tasks [4]. Each task represents a computation that needs to be performed according to a set of resources and timing constraints [4]. A task is described by timing parameters. Release time is the time instant at which a task becomes ready for execution [3]. Computation time is the time necessary to complete task's computation without any interruption. Completion or finishing time is the time instant at which the task's computation is completed. Response time is the length of time from release time to the completion time. Deadline is the time instant by which computation of a task is required to be completed. The tasks in a real time system have different types such as periodic, sporadic and aperiodic. The tasks which are executed repeatedly at regular intervals can be modeled as periodic tasks. On the other hand, tasks responding external events can be modeled

as sporadic or aperiodic. A periodic task is characterized as a sequence of jobs. A job is a task instance scheduled and executed in each period on different data.

In many real-time systems, tasks communicate via shared data. Producer and consumer tasks are synchronized if they are constrained to execute in a particular order. On the other hand, if there is no constraint about the execution orders of tasks, each producer task places the data generated by it in a shared address space to be used by the consumer task at any time [3].

The behavior and outputs of a real-time system depends on the accuracy of the computed state variable values of controlled objects. The increasing jitter in release times and variation in execution times of the tasks computing state variables defers the computation of state variable values, and causes usage of old values of state variable by consumer tasks. Moreover, the asynchrony and concurrency of consumer tasks may result in out of order executions, which causes update of state variables with older information. In these situation, the accuracy of state variables degrades. For hard real time systems, the degradation in the accuracy of state variables causes failures. In contrast, in soft real time systems the degradation in the accuracy can be tolerated according to the usefulness of state variables. However, this situation requires taking the usefulness of state variables into account for the system behavior.

2.2 Formal Verification and Model Checking

Formal verification consists in proving or disproving the correctness of the design of a system described in a mathematical formalism [1]. Model checking [2] is a collection of automatic techniques for verifying finite-state concurrent systems that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for (a given state in) that model [1]. Formal verification with model checking techniques are based on models describing the system behavior in a mathematically precise and unambiguous manner, and formalizing the property to be checked [1]. A model checking technique explores the mathematical model of the system exhaustively, and checks all explored states of the system model in order to prove that it satisfies a given property.

There are two prerequisites of model checking: Formal specification of the system behavior with a formal model, and formal specification of the properties to be checked with a property specification language. Formal modeling of system behavior is a challenging task and decisive about the success of formal verification. The system models are mostly expressed using finite-state automata, consisting of a finite set of states and a set of transitions [1]. Some of the most important modeling formalisms are guarded command language [16], and process algebra [17, 18]. In addition to the modeling of the system behavior, the properties that are to be verified has to be specified in a formal logic. Temporal logic is a formal logic which has a a high degree of expressiveness for reactive and concurrent systems [27]. The main operators of temporal logic are eventually and always. Temporal logic enables to specify reachability, liveliness, safety, and fairness properties. Temporal logic has two main classifications: Linear temporal logic (LTL) [28], Computation Tree Logic (CTL) [46] and CTL* [47], which subsumes both LTL and CTL. LTL is based on a linear-time perspective; a computation is viewed as a linear sequence with only one possible state at each moment of time. On the other hand, CTL is based on branching-time perspective; a computation is viewed as a tree sequence with many possible states at each moment of time. Basic temporal modalities of temporal logic include eventually, $\Diamond$, and always, $\Box$ operators. LTL adds two basic temporal operators next, O, and until, U. CTL includes additional temporal operators to allow the expression of branching; existential quantifier, $\exists$, and a universal quantifier, $\forall$.

Model checking technique is basically an algorithmic approach in which the properties of system are checked if they are satisfied in all states of the system model. Different property specification languages and modeling formalisms requires different model checking algorithms. The main and common obstacle of model checking is state explosion problem. The number of states of a system model can be enormous [15] because of concurrency and asynchrony. For a system consisting of multiple processes with finite states without any synchronization, the number of global states is the product of state numbers of all processes. State explosion problem is an important issue in model checking of asynchronous and concurrent systems. Moreover, software and hardware systems are data-dependent, and contains multivalued variables. Specification of multivalued variables in a system model increases the number

of global states exponentially if the variables are bounded and causes state explosion problem. Finally, specification of time for the verification of timed systems are more complex than verification of untimed systems, resulting state explosion problem. To avoid the state explosion problem several methods are developed for model checking algorithms. Symbolic model checking with binary decision diagrams (BDD) [29] is a method to fight with state explosion problem by representing the system states with binary decision diagram instead of a transition system. Another method is the partial order reduction, which tackle with the state space explosion in concurrent systems with asynchronous components. The concurrency and asynchrony makes different execution orderings of transitions possible. On the other hand, if transitions are independent different orderings does not affect the outcome. Partial order reduction method observes the independence of transitions and ignores the transitions which gives the same outcome. Counterexample-guided abstraction refinement [30] is another method for tackling the state explosion problem. Abstraction is another method of reducing the size of state space of the system model to tackle the state explosion problem. Abstraction takes a model and replaces the states of system model with abstract states according to an abstraction mapping and constructs an abstract model. Existential abstraction [31], computes an upper approximation of the original model. When a specification in the temporal logic is true in the abstract model, it will also be true in the concrete design. However, if the specification is false in the abstract model, the counterexample may be the result of some behavior in the approximation which is not present in the original model. When this happens, it is necessary to refine the abstraction so that the behavior which caused the erroneous counterexample is eliminated. The main contribution of [9] is an efficient automatic refinement technique which uses information obtained from erroneous counterexamples. The refinement algorithm keeps the size of the abstract state space small due to the use of abstraction functions which distinguish many degrees of abstraction for each program variable. Bounded model checking [32] is another method, which exploits fast Boolean satisfiability (SAT) solvers to search for counterexamples of bounded length [15].

2.3 Model Checking of Real-Time Systems

For model checking of real time systems, the timed modeling formalisms are necessary to formalize the timeliness of a system, and they are augmented with clocks to express timeliness. Some of the most important modeling formalisms for timed systems are timed process algebra [19], time petri nets (TPN) [21], and timed (TA) [8, 9]. TPN is very useful for modelling of a wide range of real-time systems including work-flow processes, scheduling problems and others [22]. In TPN, each transition is associated with a clock that records the time lapse since it was last enabled. On the other hand, unbounded TPN are too expressive and are hence unsuitable for automatic verification, which means that most of the verification approaches are limited to bounded nets [22]. Timed automata (TA) [8, 9] are a well-accepted model for representing and analyzing real-time systems. TA are as expressive as bounded Petri net based models. TA forms an extension of finite automata with dense time clocks and enables one to specify real-time systems. A combination of an easily understandable syntax and semantics together with the support for C-like constructs and data structures makes TA a widely applicable and successful approach to modeling and verification of time dependent systems [22].

Timed automata are now widely used to model real-time systems. To specify properties of timed automata, the most used branching-time temporal logic is TCTL, Timed Computation Tree Logic (TCTL) [9, 12]. TCTL is a real-time variant of CTL aimed to express properties of timed automata. The main difficulty of the TCTL model-checking problem is that a transition system with uncountably many states has to be analyzed [1]. To solve this problem, region graph construction [10] is widely used. Region graph construction constructs the region automaton, which is finite and suitable to apply ordinary model checking techniques. On the other hand, the number of states of the region automaton can increase exponentially. Model checking of timed automata is supported by tools like Kronos [23], UPPAAL [24], RED [25], and CMC [26]. These tools support timed automata, TCTL, and construction of region automata. They treat clock variables differently from discrete state variables and use specialized data structures to represent clock regions in order to suppress exponential increase in the number of states of the region automaton [13]. On the other hand,

some of these tools, such as KRONOS, does not allow variables, which makes tool unsuitable for data operations.

CHAPTER 3

DATA ABSTRACTION METHOD

3.1 Reference System Model

In this study we consider real-time systems that consist of concurrent periodic real-time tasks.In this study we consider real-time systems that consist of concurrent periodic real-time tasks. A task, τ_i is a tuple $\langle I_i, T_i, \phi_i, P_i, C_i \rangle$, where I is a task identifier, T is a bounded procedure, P is the period, ϕ is the release offset of the task with respect to the minimum release time of the task set, and C is the worst case execution time of the task. A task periodically executes T, and each periodic execution of a task is called a job (J_i^j), where J_i^j denotes execution of τ_i in the j^{th} period. A task is a sequence of jobs, $\tau_i = \{J_i^j\}_{j=0}^{\infty}$.

A task set $\mathcal{T}$ is the set of N periodic real-time tasks, which are related through a collectively performed system function to produce a computational result in its procedure T. A task set is a concurrent composition of the tasks and jobs belonging to tasks. Therefore, it is possible to express a task set as a composition of N tasks:

$$\mathcal{T} = \{J_1^j\}_{j=0}^{\infty} || \{J_2^j\}_{j=0}^{\infty} || ... || \{J_N^j\}_{j=0}^{\infty} \tag{3.1}$$

The execution orders of jobs in a task set depend on the period and release offset of the tasks. To involve the temporal properties, we define the task set as a sequence of jobs by using unit impulse as described in expressions 3.2, where k represents the involvement of temporal properties. A member of the sequence in the n^{th} index or

order is represented as $\mathcal{T}_n(k)$.

$$\begin{aligned}\mathcal{T}(k) =& J_1^0\delta(k-\phi_1) + J_1^1\delta(k-P_1-\phi_1) + J_1^2\delta(k-2P_1-\phi_1) + ... \\ &+ J_2^0\delta(k-\phi_2) + J_2^1\delta(k-P_2-\phi_2) + J_2^2\delta(k-2P_2-\phi_2) + ... \\ &... \\ &+ J_N^0\delta(k-\phi_N) + J_N^1\delta(k-P_N-\phi_N) + J_N^2\delta(k-2P_N-\phi_N) + ...\end{aligned} \tag{3.2}$$

$$\mathcal{T}(k) = \sum_{i=1}^{N}\sum_{j=0}^{\infty} J_i^j\delta(k-jP_i-\phi_i) \tag{3.3}$$

In many systems jobs exhibit release time jitter over some range [3]. Release time jitter may cause different execution orders of jobs in a task set. In the task set, a job (J_i^j) of a task (τ_i) becomes enabled at time $jP_i + \phi_i$. On the other hand, release time jitter may delay the execution of J_i^j until the next period of the task, i.e. release time jitter ($rj_{i,j}$) of each job of τ_i has a range $[0, P_i - C_i]$, leading to multiple execution sequences of jobs. Expression 3.4 describes the task set as a sequence of jobs when release time of jobs is jittered, where $rj_{i,j}^{'}$ represents an exact value for release time jitter of J_i^j.

$$\mathcal{T}^{'}(k) = \sum_{i=1}^{N}\sum_{j=0}^{\infty} J_i^j\delta(k-jP_i-\phi_i-rj_{i,j}^{'}) \tag{3.4}$$

Figure 3.1 and Figure 3.2 show the execution order of the jobs in a task set, $\mathcal{T}_1$, consisting of two tasks, τ_1 and τ_2 in a duration $[t_0, t_0 + \phi_1 + 3P_1)$. The periods of tasks are equal and ϕ_1 is smaller than ϕ_2. Figure 3.1 shows the execution of jobs when the release time is not jittered in the task set. The execution sequence of jobs is $J_1^0, J_2^0, J_1^1, J_2^1, J_1^2, J_2^2$. On the other hand, Figure 3.2 shows the execution orders of the jobs when the release time is jittered in the task set. The release time jitter of all jobs is zero except from the release time jitter of second job of τ_1, $rj_{1,1}$ is between $\phi_2 - \phi_1$ and $P_1 - C_1$. The release time jitter of $j_{1,1}$ causes a different execution sequence of jobs; $J_1^0, j_2^0, J_2^1, J_1^1, J_1^2, J_2^2$. With all possible release time jitter values, there exist eight different execution sequences of jobs in the duration $[t_0, t_0 + \phi_1 + 3P_1)$.

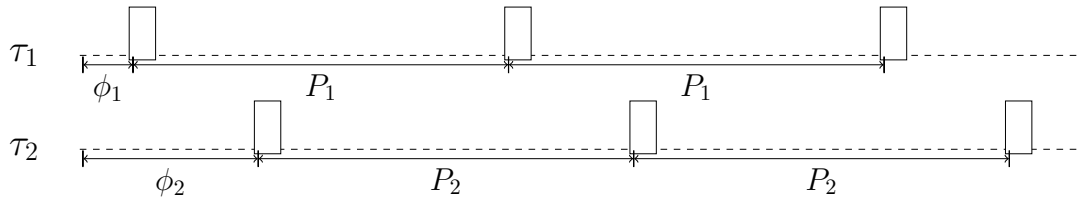


Figure 3.1: Example of two periodic tasks without any release time jitter

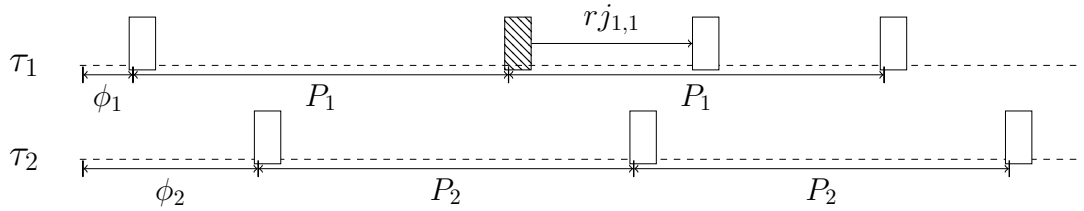


Figure 3.2: Example of two periodic tasks with release time jitter

Each task in the task set periodically updates the computational result data by processing its input data according to T. In each period of a task a new input data value is validated and the computational result data is updated with a new input data value. To distinguish the input data values with respect to the jobs in the task set, we define the input data as input data entity, which is a tuple $\langle I, v, t\rangle$, where I is the task ID, v is data value and t is the timestamp. The input data entity of τ_i generated in the j$^{\text{th}}$ period is described by

$$l_i^j = \langle I_i, v_i^j, jP_i + \phi_i\rangle \tag{3.5}$$

where v_i^j is the input data value validated at time $\phi_i + jP_i$.

The input data domain of a task i, $\mathbb{L}^{\tau_i}$, is the set of all input data entities, which are processed to update computational result data in the context of the task, τ_i. The input data domain of a task set is thus the union of input data domains of each task in the task set:

$$\mathbb{L} = \mathbb{L}^1 \cup \mathbb{L}^2 \cup ... \cup \mathbb{L}^N \tag{3.6}$$

Each input data entity (l_i^j) is processed in the procedure T_i^j in job J_i^j. Input data entities are processed according to the execution sequence of jobs in the task set. Therefore, different execution orders of jobs caused by release time jitter causes different input data process sequences. Expression 3.7 describes an input data process

sequence of the task set in the time interval $[0, \infty)$ for specific release time jitters.

$$\varsigma^{\mathcal{T}}(k) = \sum_{i=1}^{N} \sum_{j=0}^{\infty} l_i^j \delta(k - jP_i - \phi_i - rj_{i,j}^{'}) \tag{3.7}$$

An input data process sequence in a time interval, $\Delta T = [T_1, T_2]$ as:

$$\varsigma^{\Delta T}(k) = \sum_{i=1}^{N} \sum_{j=\lfloor T_1/P_i \rfloor}^{\lfloor T_2/P_i \rfloor} l_i^j \delta(k - jP_i - \phi_i - rj_{i,j}^{'}) \tag{3.8}$$

The ordered input data process sequence is the list of input data entities in domain $\mathbb{L}$ with an order according to the timestamp of each input data entity. To obtain ordered input data process sequence, the release time jitter should be omitted:

$$\gamma^{\mathcal{T}}(k) = \sum_{i=1}^{N} \sum_{j=0}^{\infty} l_i^j \delta(k - jP_i - \phi_i) \tag{3.9}$$

An ordered input data process sequence in a time interval, $\Delta T = [T_1, T_2]$ is expressed as:

$$\gamma^{\Delta T}(k) = \sum_{i=1}^{N} \sum_{j=\lfloor T_1/P_i \rfloor}^{\lfloor T_2/P_i \rfloor} l_i^j \delta(k - jP_i - \phi_i) \tag{3.10}$$

Hyperperiod (H) of a task set is the least common multiple of the periods of tasks in the task set. *Hyperperiod Interval (ΔH_i)* is the time interval described as:

$$\Delta H_h = [\phi_{min} + (h-1) \times H, \phi_{min} + h \times H) \tag{3.11}$$

where ϕ_{min} is the minimum task offset, $min(\{\phi_i\}_{i=1}^{N})$. It is possible to describe ordered input data process sequences by using hyperperiod intervals. Equation 3.12 describes an ordered input data process sequence in the h[th] hyperperiod interval.

$$\gamma^{\Delta H_h}(k) = \sum_{i=1}^{N} \sum_{j=\lfloor (\phi_{min}+(h-1)\times H)/P_i \rfloor}^{\lfloor (\phi_{min}+h\times H)/P_i \rfloor - 1} l_i^j \delta(k - jP_i - \phi_i) \tag{3.12}$$

where $h > 0$. The ordered input data process sequence in successive hyperperiods from the first hyperperiod to the M[th] hyperperiod:

$$\gamma^{\Delta H_1^M}(k) = \sum_{i=1}^{N} \sum_{j=\lfloor \phi_{min}/P_i \rfloor}^{\lfloor (\phi_{min}+M\times H)/P_i \rfloor - 1} l_i^j \delta(k - jP_i - \phi_i) \tag{3.13}$$

Definition 1. An input data process sequence domain, $L_{\varsigma(k)}(k)$, is the set of input data entities whose elements are the member of input data process sequence $\varsigma(k)$.

Input data process sequences in domain $\mathbb{L}$ have various relations that reflect the relation between input data process sequence domains and orders of the input data entities in the input data process sequences. Definition 2 defines the equality relation.

Definition 2. Two input data process sequences, $\varsigma^1(k)$ and $\varsigma^2(k)$, are equal if $L_{\varsigma_n^1(k)}(k) = L_{\varsigma_n^2(k)}(k)$ and the process order of each input data entities are same in both sequences, i.e. $\varsigma_n^1(k) = \varsigma_n^2(k)$.

Definition 3. Two input data process sequences, $\varsigma^1(k)$ and $\varsigma^2(k)$, are similar ($\sim$) if $L_{\varsigma^1(k)}(k) = L_{\varsigma^2(k)}(k)$ and the process order of input data entities are different.

Another relation is k-shift congruency between input data entities and input data process sequences which implies that two input data entities are processed in the context of the same task in different periods.

Definition 4. Let l_i^j and l_i^m be two input data entities, H be hyperperiod of task set and suppose that $M \in \mathbb{N}$. If m is congruent to j mod MH/P_i, i.e. $m - j = MH/P_i$, then l_i^m is congruent to l_i^j shift MH, $l_i^m \equiv l_i^j$ shift MH.

K-shift congruency between two input data process sequences describes that the process orders of input data entities in the sequences are same with respect to the task context to which they belong by abstracting the absolute time. Definition 5 defines the k-shift congruence relation between two input data process sequences.

Definition 5. Let $\varsigma^1(k)$ and $\varsigma^2(k)$ be two input data process sequences described by expressions

$$\varsigma^1(k) = \sum_{i=1}^{N} \sum_{j=\lfloor T_1/P_i \rfloor}^{\lfloor T_2/P_i \rfloor} l_i^j \delta(k - jP_i - \phi_i - rj_{i,j}^{'}) \tag{3.14}$$

$$\varsigma^2(k) = \sum_{i=1}^{N} \sum_{j=\lfloor (T_1+MH)/P_i \rfloor}^{\lfloor (T_2+MH)/P_i \rfloor} l_i^j \delta(k - jP_i - \phi_i - rj_{i,j}^{'}) \tag{3.15}$$

where $|\varsigma^1(k)| = |\varsigma^2(k)|$ and H is the hyperperiod. Suppose that $l_n^1(k)$ is l_i^j. If for each n$^{\text{th}}$ member of $l_n^1(k) = l_i^j$, $l_n^2(k)$ is equal to l_i^{j+MH/P_i}, then $\varsigma^2(k)$ is congruent to $\varsigma^1(k)$ shift MH, $\varsigma^2(k) \equiv \varsigma^1(k)$ shift MH or $\varsigma^2(k) \equiv \varsigma^1(k - MH)$.

Proposition 1. If $\varsigma_n^1(k)$ and $\varsigma_n^2(k)$ are two input data process sequences that $\varsigma^2(k) \equiv \varsigma^1(k)$ shift MH, then $L_{\varsigma^2}(k)$ is congruent to $L_{\varsigma^1}(k)$ shift MH, $L_{\varsigma^2}(k) \equiv L_{\varsigma^1}(k)$ shift MH.

Proof. Let l_i^j be a member of $\varsigma_n^1(k)$. For each member,l_i^j, of $\varsigma_n^1(k)$, l_i^{j+nH/P_i} is a member of $\varsigma_n^2(k)$. Therefore for each element of $\varsigma_n^1(k)$, there exist a k-shift congruent element of $\varsigma_n^2(k)$.

Definition 6. $\varsigma_n^1(k)$, $\varsigma_n^2(k)$ and $\varsigma_n^3(k)$ are three input data process sequences. If $\varsigma_n^2(k)$ is congruent to $\varsigma_n^1(k)$ shift MH ($\varsigma^2(k) \equiv \varsigma^1(k-MH)$) and $\varsigma_n^3(k)$ is similar with $\varsigma_n^2(k)$, $\varsigma^3(k) \sim \varsigma^2(k)$, than $\varsigma_n^3(k)$ similar congruent to $\varsigma_n^1(k)$ shift MH, $\varsigma^3(k) \cong \varsigma^1(k-MH)$.

Due to jitter, as expressed in Equation 3.7, input data entities can be processed in different orders. The main reason of multiple input data process sequences is interleaving between input data entities because of release time jitter of tasks. An input data entity, processed in a task context, may interleave with other input data entities, which are processed in other tasks' context. Definition 7 defines the interleaving set of an input data entity.

Definition 7. *Interleaving Set of an input data entity ($I_{l_i^j}(k)$)* is the set of input data entities that are possible to interleave with the input data entity of interest. The interleaving set of an input data entity l_i^j is described by equation

$$\begin{aligned} I_{l_i^j}(k) = \{ & l_m^n \mid l_m^n \in \mathbb{L}, m \neq i, \\ & nP_m + \phi_m + rj_{m,max} > jP_i + \phi_i \quad if\ l_m^n.t \leq l_i^j.t, \\ & jP_i + \phi_i + rj_{i,max} > nP_m + \phi_m \qquad if\ l_i^j.t < l_m^n.t \} \end{aligned} \tag{3.16}$$

where $rj_{i,max}$ is the maximum release time jitter.

Proposition 2. Let l_i^j and l_i^{j+MH/P_i} be input data entities. Assume that each elements of $I_{l_i^j}(k)$ is an element of $\mathbb{L}$, then $L_{l_i^{j+MH/P_i}}(k)$ is congruent to $I_{l_i^j}(k)$ with shift MH, i.e. $I_{l_i^{j+MH/P_i}}(k) \equiv I_{l_i^j}(k-MH)$ shift MH.

Proof. Let l_a^b be an element of $I_{l_i^j}(k)$.

For the case $l_a^b.t < l_i^j.t$:

$$bP_a + \phi_a + rj_{a,max} > jP_i + \phi_i$$
$$bP_a + \phi_a + rj_{a,max} + nH > jP_i + \phi_i + MH$$
$$(b + MH/P_a)P_a + \phi_a + rj_{a,max} > (j + MH/P_i)P_i + \phi_i$$
$$Thus,\ l_a^{b+MH/P_a} \in I_{l_i^{j+MH/P_i}}(k)$$

For the case $l_a^b.t > l_i^j.t$:

$$jP_i + \phi_i + rj_{i,max} > bP_a + \phi_a$$
$$jP_i + \phi_i + rj_{i,max} + MH > bP_a + \phi_a + MH$$
$$(j + MH/P_i)P_i + \phi_i + rj_{i,max} > (b + MH/P_a)P_a + \phi_a$$
$$Thus,\ l_a^{b+MH/P_a} \in I_{l_i^{j+MH/P_i}}(k)$$

Proposition 3. Two similar input data process sequences, $\varsigma^1(k)$ and $\varsigma^2(k)$, have the same interleaving set.

Proof.

$$\begin{aligned} I_{\varsigma^1(k)}(k) &= \bigcup_{n=1}^{N} I_{\varsigma_n^1(k)}(k) - L_{\varsigma^1(k)}(k) = \bigcup_{n=1}^{N} I_{\varsigma_n^2(k)}(k) - L_{\varsigma^2(k)}(k) \\ &= I_{\varsigma^2(k)}(k) \end{aligned}$$

□

Proposition 4. Assume that $\varsigma^1(k)$ and $\varsigma^2(k)$ are input data process sequences that are described in Equation 3.14 and Equation 3.15. Assume also that T_1 is large enough that each elements of $I_{\varsigma^1(k)}(k)$ is an element of $\mathbb{L}$. If $\varsigma^2(k) \equiv \varsigma^1(k - MH)$, then $I_{\varsigma^2(k)}(k)$ is congruent to $I_{\varsigma^1(k)}(k)$ shift MH, i.e. $I_{\varsigma^2(k)}(k) \equiv I_{\varsigma^1(k)}(k)$ shift MH.

Proof.

$$\begin{aligned} I_{\varsigma^2(k)}(k) &= \bigcup_{n=1}^{N} I_{\varsigma_n^2(k)}(k) - L_{\varsigma^2(k)}(k) \equiv \bigcup_{n=1}^{N} I_{\varsigma_n^1(k)}(k - nH) - L_{\varsigma^1(k)}(k - nH) \\ &\equiv I_{\varsigma^1(k)}(k) \end{aligned}$$

□

Definition 8. The complement of an input data process sequence domain, denoted $\overline{L}_{\varsigma(k)}(k)$, is the set which are not elements of $L_{\varsigma(k)}(k)$ in domain $\mathbb{L}$.

Proposition 5. Let $\varsigma^1(k)$ be an input data process sequence similar with $\gamma^{\Delta H_1^M}(k)$, and $\varsigma^2(k)$ be an input data process sequence similar with $\gamma^{\Delta H_1^{M+N}}(k)$. $\overline{L}_{\varsigma^2(k)}(k)$ is congruent to $\overline{L}_{\varsigma^1(k)}(k)$ shift NH.

Proof. Let $\overline{\varsigma}^1(k)$ and $\overline{\varsigma}^2(k)$ be an input data process sequences as described by

$$\begin{aligned}\overline{\varsigma}^1(k) &= \sum_{i=1}^{N} \sum_{j=\lfloor \phi_{min}+M\times H/P_i \rfloor}^{\infty} l_i^j \delta(k - jP_i - \phi_i) \\ \overline{\varsigma}^2(k) &= \sum_{i=1}^{N} \sum_{j=\lfloor \phi_{min}+(M+N)\times H/P_i \rfloor}^{\infty} l_i^j \delta(k - jP_i - \phi_i)\end{aligned} \tag{3.17}$$

Since $\overline{\varsigma}^2(k)$ is congruent to $\overline{\varsigma}^1(k)$ shift H, $L_{\overline{\varsigma}^2(k)}(k)$ is congruent to $L_{\overline{\varsigma}^1(k)}(k)$ shift NH. On the other hand $L_{\overline{\varsigma}^2(k)}(k) = \overline{L}_{\varsigma^2(k)}(k)$ and $L_{\overline{\varsigma}^1(k)}(k) = \overline{L}_{\varsigma^1(k)}(k)$, therefore $\overline{L}_{\varsigma^2(k)}(k)$ is congruent to $\overline{L}_{\varsigma^1(k)}(k)$ shift NH. □

Definition 9. $\varsigma^{\varepsilon}(k)$ is an empty input data process sequence. $L_{\varsigma^{\varepsilon}(k)}(k)$ is an empty set and $\overline{L}_{\varsigma^{\varepsilon}(k)}(k)$ is equal to $\mathbb{L}$. $\overline{L}_{\varsigma^1(k)}(k)$ is congruent to $\overline{L}_{\varsigma^{\varepsilon}(k)}(k)$ shift MH.

Proposition 6. Suppose that $\varsigma_n^1(k)$ and $\varsigma^2(k)$ are input data process sequences that are described by equations

$$\varsigma^1(k) = \sum_{i=1}^{N} \sum_{j=0}^{\lfloor T/P_i \rfloor} l_i^j \delta(k - jP_i - \phi_i - rj_{i,j}^{'}) \tag{3.18}$$

$$\varsigma^2(k) = \sum_{i=1}^{N} \sum_{j=0}^{\lfloor (T+nH)/P_i \rfloor} l_i^j \delta(k - jP_i - \phi_i - rj_{i,j}^{'}) \tag{3.19}$$

Assume that $\varsigma^2(k)$ is similar with an input data process sequence described by

$$\varsigma^2(k) \sim \varsigma^4(k) = \sum_{h=1}^{M} \varsigma^{\Delta H_h}(k).\varsigma^3(k) = \varsigma^{\Delta H_1^M}(k).\varsigma^3(k) \tag{3.20}$$

If $\varsigma^3(k) \equiv \varsigma^1(k - MH)$, then $I_{\varsigma^3(k)}(k) - L_{\Delta H_1^M}(k) \equiv I_{\varsigma^1(k)}(k - MH)$.

Proof. Let l_a^{j+MH/P_b} is a member of $\varsigma^3(k)$. If l_a^{b+MH/P_b} is a member of $I_{\varsigma^3(k)}(k)$ and $l_a^{j+MH/P_b}.t < l_i^{j+MH/P_i}.t$, then l_a^b is a member of $\varsigma^1(k)$. On the other hand, if

$l_a^{j+MH/P_b} \in L_{\Delta H_1^{M-1}}(k)$, then $(b + MH/P_a)P_a + \phi_a < \phi_{min} + MH$. Thus, $l_a^b \notin \mathbb{L}$ because $bP_a + \phi_a < \phi_{min}$ is not true.

□

Definition 10. *min* is a function that describes the input data entity having minimum timestamp in a set of input data entities, such as interleaving set (I) or domain of an input data process sequence (L).

$$min\{S\} : S \rightarrow l_i^j \mid l_i^j \in S, \forall l_a^b.t \in S : l_i^j.t \leq l_a^b.t$$

Definition 11. *Successor Input Data Entity Set (S)* of an input data process sequence, $\varsigma(k)$, is the set of input data entities that may possibly be processed after $\varsigma(k)$. The possible input data entities that are not processed by $\varsigma(k)$ that may possibly be processed after $\varsigma(k)$.

The set of input data entities that are not processed by $\varsigma(k)$ is $\overline{L}_{\varsigma(k)}(k)$.

On the other hand $S_{\varsigma(k)}(k)$ is a subset of $\overline{L}_{\varsigma(k)}(k)$ according to some restrictions: The first restriction is that the input data entities belonging to the same task context are processed sequentially. Therefore, $S_{\varsigma(k)}(k)$ cannot consist of multiple input data entities belonging to a task. The second restriction is about the release time jitter. Let l_i^j be an input data entity. An input data entity, l_a^b having a timestamp greater than $\phi_i + jP_i + rj_{i,max}$ cannot be processed if l_i^j does not processed. In other words $S_{\varsigma(k)}(k)$ cannot consist of two input data entities like l_i^j and l_a^b. With respect to these restrictions successor set of an input data process sequence is described by equation

$$S_{\varsigma(k)}(k) = min\{\overline{L}_{\varsigma(k)}(k)\} \cup (I_{min\{\overline{L}_{\varsigma(k)}(k)\}}(k) - L_{\varsigma(k)}(k)) \tag{3.21}$$

The successor set of an empty input data process sequence, $\varsigma^{\varepsilon}(k) = \varepsilon$, is the union of interleaving set of the input data entity which has the least time stamp, l_1^0, and the related input data entity.

$$S_{\varsigma^{\varepsilon}(k)}(k) = min\{\mathbb{L}\} \cup I_{min\{\mathbb{L}\}}(k) = I(l_1^0) \cup l_1^0 \tag{3.22}$$

Proposition 7. Two similar input data process sequences, $\varsigma^1(k)$ and $\varsigma^2(k)$, have the same successor input data entity set.

Proof.

$$
\begin{aligned}
S_{\varsigma^1(k)}(k) &= min\{\overline{L}_{\varsigma^1(k)}(k)\} \cup (I_{min\{\overline{L}_{\varsigma^1(k)}(k)\}}(k) - L_{\varsigma^1(k)}(k)) \\
&= min\{\overline{L}_{\varsigma^2(k)}(k)\} \cup (I_{min\{\overline{L}_{\varsigma^2(k)}(k)\}}(k) - L_{\varsigma^2(k)}(k)) \\
&= S_{\varsigma^2(k)}(k)
\end{aligned}
$$

□

Proposition 8. Suppose that $\varsigma_n^1(k)$ and $\varsigma^2(k)$ are input data process sequences that are described in Proposition 6. If $\varsigma^3(k) \cong \varsigma^1(k-MH)$, then $S_{\varsigma^2(k)}(k) \cong S_{\varsigma^1(k)}(k-MH)$.

Proof.

$$
\begin{aligned}
S_{\varsigma^2(k)}(k) = S_{\varsigma^4(k)}(k) &= min\{\overline{L}_{\varsigma^4(k)}(k)\} \cup (I_{min\{\overline{L}_{\varsigma^4(k)}(k)\}}(k) - L_{\varsigma^4(k)}(k)) \\
&= min\{\overline{L}_{\varsigma^1(k)}(k-MH)\} \cup \\
&\qquad (I_{min\{\overline{L}_{\varsigma^1(k)}(k-MH)\}}(k) - L_{\Delta H_1^M}(k) \cup L_{\varsigma^3(k)}(k)) \\
&= min\{\overline{L}_{\varsigma^1(k)}(k-MH)\} \cup (I_{min\{\overline{L}_{\varsigma^1(k)}(k)\}}(k-MH) - L_{\varsigma^3(k)}(k)) \\
&= min\{\overline{L}_{\varsigma^1(k)}(k-MH)\} \cup \\
&\qquad (I_{min\{\overline{L}_{\varsigma^1(k)}(k)\}}(k-MH) - L_{\varsigma^1(k)}(k-MH)) \\
&= S_{\varsigma^1(k)}(k-MH)
\end{aligned}
$$

□

Proposition 9. If ς_2 is an input data process sequence that is similar with an input data process sequence $\varsigma^{\Delta H_1}(k).\varsigma^3(k)$, then there is an input data process sequence $\varsigma^1(k)$ such that $\varsigma^3(k) \equiv \varsigma^1(k-H)$.

Proof. $S_{\varsigma^{\Delta H_1}(k)}(k) \equiv S_{\varsigma^{\varepsilon}(k)}(k-H)$ according to Proposition 8. For an input data entity l_i^j, which is element of $S_{\varsigma^{\varepsilon}(k)}(k)$, there is an input data entity l_i^{j+H/P_i}, which is element of $S_{\varsigma^{\Delta H_1}(k)}(k)$. Let $\varsigma^1(k)'$ be $\{l_i^j\}$. Then $\varsigma^3(k)'$ is $\{l_i^{j+H/P_i}\}$, which is congruent to $\varsigma^1(k-H)'$ shift H, i.e. $\varsigma^3(k)' \equiv \varsigma^1(k-H)'$. Therefore, $S_{\varsigma^{\Delta H_1}(k).\varsigma^3(k)'}(k)$ is congruent to $S_{\varsigma^1(k)'}(k)$ shift H according to Proposition 8. Iteratively for all successive input data process sequences $\varsigma^{\Delta H_1}(k)$ and $\varsigma^{\varepsilon}(k)$ are congruent to each other with H shift. □

3.2 Data Abstraction Mapping

The computational result produced by concurrent tasks is updated by processing an input data entity (l_i^j) according to the common functional operation of the task set. The functional operation uses present the computational result and input data entity to update the computational result data value according to the relation:

$$f(g_{n-1}, l_i^j) = g_{n-1} \circ l_i^j : g_{n-1} \xrightarrow{\Psi(l_i^j)} g_n \tag{3.23}$$

where g_n and g_{n-1} are updated and present computational result data values, respectively, l_i^j is the processed input data entity, $\circ$ dentoes the functional operation, and $\Psi(l_i^j)$ is the process of input data entity. Input data entities, which are members of an input data process sequence $\varsigma(k)$, are processed iteratively. Equation 3.24 describes the update of computational result data by processing input data entities of input data process sequence $\varsigma(k)$.

$$f(g_{n-1}, \varsigma_n(k).\varsigma_{n+1}(k).\varsigma_{n+2}(k)) = g_{n+2} \Rightarrow \begin{cases} g_n := g_{n-1} \circ \varsigma_n(k) \\ g_{n+1} := g_n \circ \varsigma_{n+1}(k) \\ g_{n+2} := g_{n+1} \circ \varsigma_{n+2}(k) \end{cases} \tag{3.24}$$

According to Equation 3.24, the computational result data value updates can be described as transitions described as:

$$g_{n-1} \xrightarrow{\Psi(\varsigma(k))} g_{n+2} = g_{n-1} \xrightarrow{\Psi(\varsigma_n(k))} g_n \xrightarrow{\Psi(\varsigma_{n+1}(k))} g_{n+1} \xrightarrow{\Psi(\varsigma_{n+2}(k))} g_{n+2} \tag{3.25}$$

If the functional operation is not order independent, processing an input data entity in different orders results in different computational result data values depending on the process order of input data entity in the execution sequence. The input data process order dependency of computational result data changes with respect to the characteristic of functional operation. In this study we focus on the functional operations that are order dependent with confluence.

Definition 12. Let $\varsigma^1(k)$ be an ordered input data process sequence in the time interval $\Delta T = [0, T]$, whose length is N. Suppose that $\varsigma^2(k)$ is a process sequence which is similar to $\varsigma^1(k)$, but the orders of input data entities are different except the N^{th}

member. In other words, $L_{\varsigma^1(k)}(k) = L_{\varsigma^2(k)}(k)$, $\varsigma_n^1(k) \neq \varsigma_n^2(k)$ for $n < N$, and $\varsigma_N^1(k) = \varsigma_N^2(k)$.

A functional operation is order dependent with confluence, if $f(g_0, \pi^{\varsigma^1(k)}(M)) \neq f(g_0, \pi^{\varsigma^2(k)}(M))$ for each $M < N$, and $f(g_0, \varsigma^1(k)) = f(g_0, \varsigma^2(k))$.

If the functional operation of the task set is order dependent with confluence, processing an out of order input data item results in different computational result data values from the computational result data value obtained when input data is processed in order; that is, usefulness of computational data is degraded. Figure 3.3 illustrates order dependency with confluence on the system configuration defined in Figure 3.1 and Figure 3.2. While the input data generation sequence is $\{l_1^0, l_2^0, l_1^1, l_2^1, l_1^2, l_2^2\}$, the processed input data sequence is $\{l_1^0, l_2^0, l_2^1, l_1^1, l_1^2, l_2^2\}$, where $l_1^1.t < l_2^1.t$. Therefore, $f(g_{k-1}, l_1^0.l_2^0.l_2^1.l_1^2)$ and $f(g_{k-1}, l_1^0.l_2^0.l_2^1.l_1^2)$ result in different computational result data values, such as $|g_3 - g_3^{'}| > 0$, and $|g_4 - g_4^{'}| > 0$.

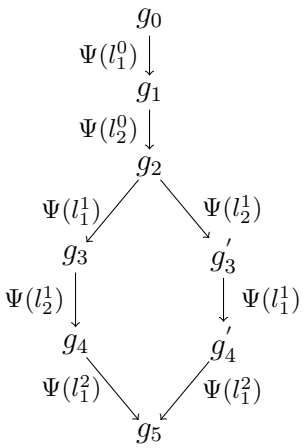


Figure 3.3: Example of two periodic tasks with release time jitter

We define an abstraction method to represent the computational result with finite states, while the transitions between states represent the process sequence of input data entities, by abstracting data value and time. The abstract input data entity is equal to input data entity's ID, i.e, $\mathcal{A}_l(l_i^j) = l_i^j.I_i$ to preserve information.

$$\mathcal{A}_l(\Psi(l_i^j)) = \Psi(\mathcal{A}_l(l_i^j)) = \Psi(l_i^j.I_i) \tag{3.26}$$

Abstract computational result state comprises information about whether the current

values of computational result data are erroneous, and the previously processed input data entities which are abstracted from data value and time. If $\varsigma(k)$ is an input data process sequence of size N, the computational result g_i obtained by processing $\varsigma(k)$ is $f(g_{init}, \varsigma(k))$. The output function of a state, λ, determines the output of the state as accurate and erroneous depending on the processing orders of input data as described by Equation 3.27. Figure 3.4 shows the state transitions between accurate and erroneous states.

$$\lambda : \begin{cases} \mathcal{A}(g_i) \mapsto \alpha & \text{if } \varsigma_N(k) \text{ is in order} \\ \mathcal{A}(g_i) \mapsto \varepsilon & \text{else } \varsigma_N(k) \text{ is out of order} \end{cases} \tag{3.27}$$

Definition 13. Let $\varsigma(k)$ be an input data process sequence with a size N. Let $L_<(l_i^j)$ be a set of input data entities which have a smaller timestamp than $l_i^j.t$, $L_>(l_i^j)$ be a set of input data entities which have a greater timestamp than $l_i^j.t$, and $L_{\varsigma(k)}^{N-1}(k)$ is equal to $L_{\varsigma(k)}(k) - \varsigma_N(k)$.

$\varsigma_N(k)$ is in order if $L_{\varsigma(k)}^{N-1}(k) \supset L_<(\varsigma_n(k))$ and $L_{\varsigma(k)}^{N-1}(k) \cap L_>(\varsigma_n(k)) = \emptyset$.

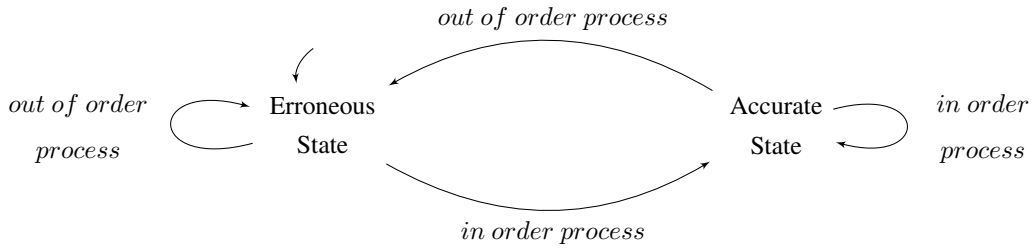


Figure 3.4: Abstract Central Estimation State Diagram

According to the abstraction of computational result data value to computational result data state, the output in a computational result state indicates the input data process order. Therefore, the input data process order semantics are described by the output. On the other hand, the processed input data entities are still important for determining computational result state, because the next states of a computational result state depend on the processed input data entities. If g_n is a computational result that is equal to $f(g_{init}, \varsigma(k))$, the abstract state of g_n, ($\mathcal{A}(g_n)$), consists of two attributes: output, O, and the abstracted processed input data entity set, $L^{\varsigma(k)}(k)$ as described by equation

$$\mathcal{A}(g_i) = \langle O, \mathcal{A}_l(L^{\varsigma(k)}(k)) \rangle \tag{3.28}$$

where $O \in \{\alpha, \epsilon\}$, and $\mathcal{A}_l(L^{\varsigma(k)}(k))$ is a multiset of $l_i^j.I_i$ for each l_i^j which is element of $L^{\varsigma(k)}(k)$.

Definition 14. Let g_n be a computational result which is equal to $f(g_{init}, \varsigma(k))$. Assume that the successor input data set of $\varsigma(k)$ is $S_{\varsigma(k)}(k)$. The successor computational result set of g_n, $Post(g_n)$, is

$$Post(g_n) = \bigcup_{S_{\varsigma(k)}(k)} f(g_n, l_i^j) \qquad l_i^j \in S_{\varsigma(k)}(k) \tag{3.29}$$

Proposition 10. Assume that $\varsigma^1(k)$ and $\varsigma^2(k)$ are two similar input data process sequences, $\varsigma^1(k) \sim \varsigma^2(k)$. $\varsigma^1(k)$ and $\varsigma^2(k)$ have the same the successor set, $S_{\varsigma^1(k)}(k) = S_{\varsigma^2(k)}(k) = S$, according to Proposition 7. For each l_i^j we have $g_1' \in Post(f(g_{init}, \varsigma^1(k)))$ and $g_2' \in Post(f(g_{init}, \varsigma^2(k)))$ such that $g_1' \in Post(f(g_{init}, \varsigma^1(k)l_i^j))$ and $g_2' \in Post(f(g_{init}, \varsigma^2(k)l_i^j))$, where $l_i^j \in S$, $\mathcal{A}(g_1').O = \mathcal{A}(g_2').O$

Proof. Let g_1 and g_2 be computational results such that $g_1 = f(g_{init}, \varsigma^1(k))$ and $g_2 = f(g_{init}, \varsigma^2(k))$. For each element $g_1' \in Post(g_1)$, there exists an element $g_2' \in Post(g_2)$ such that $g_1' = f(g_1, l_i^j)$ and $g_2' = f(g_2, l_i^j)$, where $l_i^j \in S$. Let $\varsigma^1(k)'$ be an input data sequence equals to $\varsigma^1(k).l_i^j$, and $\varsigma^2(k)'$ be an input data sequence equals to $\varsigma^2(k).l_i^j$. If $L_{\varsigma^1(k)}(k) \supset L_<(l_i^j)$ and $L_{\varsigma^1(k)}(k) \cap L_>(l_i^j) = \emptyset$, then $L_{\varsigma^2(k)}(k) \supset L_<(l_i^j)$ and $L_{\varsigma^2(k)}(k) \cap L_>(l_i^j) = \emptyset$. Therefore, $\mathcal{A}(f(g_1, l_i^j)).O = \mathcal{A}(f(g_2, l_i^j)).O$ for each $l_i^j \in S$. □

Proposition 11. Assume that $\varsigma^1(k)$ and $\varsigma^2(k)$ are two input data process sequences; that is $\varsigma^2(k) \sim \varsigma^{\Delta H_1^M}(k).\varsigma^3(k)$, where $\varsigma^3(k) \equiv \varsigma^1(k - MH)$. $S_{\varsigma^2(k)}(k)$ is congruent to $S_{\varsigma^1(k)}(k)$ shift MH according to Proposition 8. For each pair of elements $g_1' \in Post(f(g_{init}, \varsigma^1(k)))$ and $g_2' \in Post(f(g_{init}, \varsigma^2(k)))$ such that $g_1' = f(g_{init}, \varsigma^1(k).l_i^j)$ and $g_2' = f(g_{init}, \varsigma^2(k).l_i^{j+MH/P_i})$, where $l_i^j \in S_{\varsigma^1(k)}(k)$ and
$l_i^{j+MH/P_i} \in S_{\varsigma^2(k)}(k)$, $\mathcal{A}(g_1').O = \mathcal{A}(g_2').O$

Proof. Let g_1 and g_2 be computational results such that $g_1 = f(g_{init}, \varsigma^1(k))$ and $g_2 = f(g_{init}, \varsigma^2(k))$. For each element $g_1' \in Post(g_1)$, there exists an element $g_2' \in Post(g_2)$ such that $g_1' = f(g_1, l_i^j)$ and $g_2' = f(g_2, l_i^{j+MH/P_i})$, where $l_i^j \in S_{\varsigma^1(k)}(k)$ and $l_i^{j+MH/P_i} \in S_{\varsigma^2(k)}(k)$. Let $\varsigma^1(k)'$ be an input data sequence equals to $\varsigma^1(k).l_i^j$, and $\varsigma^2(k)'$ be an input data sequence equals to $\varsigma^2(k).l_i^j$. If $L_{\varsigma^1(k)}(k) \supset L_<(l_i^j)$ and

$L_{\varsigma^1(k)}(k) \cap L_{>}(l_i^j) = \emptyset$, then $(L_{\varsigma^{\Delta H_1^M}(k)}(k) \cup L_{\varsigma^1(k-MH)}(k)) \supset L_{<}(l_i^{j+MH/P_i})$ and $(L_{\varsigma^{\Delta H_1^M}(k)}(k) \cup L_{\varsigma^1(k)}(k-MH)) \cap L_{>}(l_i^{j+MH/P_i}) = \emptyset$. Therefore, $\mathcal{A}(f(g_1, l_i^j)).O = \mathcal{A}(f(g_2, l_i^{j+MH/P_i})).O$ for each $l_i^j \in S_{\varsigma^1(k)}(k)$ and $l_i^{j+MH/P_i} \in S_{\varsigma^2(k)}(k)$. □

Definition 15. Let g_e and g_f be two computational results such as $g_e = f(g_{init}, \varsigma^e(k))$ and $g_f = f(g_{init}, \varsigma^f(k))$. Assume that l_a^b is an element of $S_{\varsigma^e(k)}(k)$ and l_c^d is an element of $S_{\varsigma^f(k)}(k)$, where l_a^b. The abstract computational result states g_e and g_f are equal if

1. $\mathcal{A}(g_e).O = \mathcal{A}(g_f).O$
2. $\mathcal{A}_l(S_{\varsigma^e(k)}(k)) = \mathcal{A}_l(S_{\varsigma^f(k)}(k))$
3. $\mathcal{A}(f(g_e, l_a^b)).O = \mathcal{A}(f(g_f, l_c^d)).O$ for each elements $l_a^b \in S_{\varsigma^e(k)}(k)$ and $l_c^d \in S_{\varsigma^f(k)}(k)$, where $\mathcal{A}_l(l_a^b) = \mathcal{A}_l(l_c^d)$

Lemma 1. Assume that $\varsigma^1(k)$ and $\varsigma^2(k)$ are two input data process sequences. Let g_1 be equal to $f(g_{init}, \varsigma^1(k))$, and g_2 be equal to $f(g_{init}, \varsigma^2(k))$. If $\varsigma^1(k)$ and $\varsigma^2(k)$ are similar, $\varsigma^1(k) \sim \varsigma^2(k)$, then $\mathcal{A}(Post(g_1)) = \mathcal{A}(Post(g_2))$

Proof. Suppose that $g_1' \in Post(f(g_{init}, \varsigma^1(k)))$ and $g_2' \in Post(f(g_{init}, \varsigma^2(k)))$ such that $g_1' = f(g_{init}, \varsigma^1(k).l_i^j)$ and $g_2' = f(g_{init}, \varsigma^2(k).l_i^j)$, where $l_i^j \in S$. $\mathcal{A}(g_1').O = \mathcal{A}(g_2').O$ according to Proposition 10. Let $\varsigma^1(k)'$ be an input data sequence equals to $\varsigma^1(k).l_i^j$, and $\varsigma^2(k)'$ be an input data sequence equals to $\varsigma^2(k).l_i^j$. Since $\varsigma^1(k) \sim \varsigma^2(k)$, $\varsigma^1(k)'$ is similar to $\varsigma^2(k)'$, and $S_{\varsigma^1(k)'}(k) = S_{\varsigma^2(k)'}(k) = S'$.
Thus, $\mathcal{A}_l(S_{\varsigma^1(k)'}(k)) = \mathcal{A}_l(S_{\varsigma^2(k)'}(k))$. According to Proposition 10, $\mathcal{A}(f(g_1', l_a^b)).O = \mathcal{A}(f(g_2', l_a^b)).O$ for each $l_a^b \in S'$. Therefore, $\mathcal{A}(Post(g_1)) = \mathcal{A}(Post(g_2))$. □

Lemma 2. Suppose that $\varsigma^1(k)$ and $\varsigma^2(k)$ are two input data process sequences; that is $\varsigma^2(k) \sim \varsigma^{\Delta H_1^M}(k).\varsigma^3(k)$. Let g_1 be equal to $f(g_{init}, \varsigma^1(k))$, and g_2 be equal to $f(g_{init}, \varsigma^2(k))$. If $\varsigma^3(k) \equiv \varsigma^1(k - MH)$, then $\mathcal{A}(Post(g_1)) = \mathcal{A}(Post(g_2))$.

Proof. Suppose that $g_1' \in Post(f(g_{init}, \varsigma^1(k)))$ and $g_2' \in Post(f(g_{init}, \varsigma^2(k)))$ such that $g_1' = f(g_{init}, \varsigma^1(k).l_i^j)$ and $g_2' = f(g_{init}, \varsigma^2(k).l_i^{j+MH/P_i})$, where $l_i^j \in S_{\varsigma^1(k)}(k)$ and $l_i^{j+MH/P_i} \in S_{\varsigma^2(k)}(k)$. $\mathcal{A}(g_1').O = \mathcal{A}(g_2').O$ according to Proposition 11. Let $\varsigma^1(k)'$ be an input data sequence equals to $\varsigma^1(k).l_i^j$, and $\varsigma^2(k)'$ be an input data sequence equals to $\varsigma^2(k).l_i^{j+MH/P_i}$. $\varsigma^2(k)'$ is similar with $\varsigma^{\Delta H_1^M}(k).\varsigma^3(k)'$, where

$\varsigma^3(k)' = \varsigma^3(k).l_i^{j+MH/P_i}$. Since $\varsigma^3(k) \equiv \varsigma^1(k - MH)$, $\varsigma^3(k)'$ is congruent to $\varsigma^1(k - MH)'$, and $S_{\varsigma^2(k)'}(k) \equiv S_{\varsigma^1(k)'}(k - MH)$. Thus, $\mathcal{A}_l(S_{\varsigma^1(k)'}(k)) = \mathcal{A}_l(S_{\varsigma^2(k)'}(k))$. According to Proposition 11, $\mathcal{A}(f(g_1', l_a^b)).O = \mathcal{A}(f(g_2', l_a^{b+MH/P_a})).O$ for each $l_a^b \in S_{\varsigma^1(k)'}(k)$ and $l_a^{b+MH/P_a} \in S_{\varsigma^2(k)'}(k)$. Therefore, $\mathcal{A}(Post(g_1)) = \mathcal{A}(Post(g_2))$. □

Theorem 1. A task set has finite abstract computational result states according to the proposed abstraction method, if functional operation is order dependent with confluence.

Proof. Let $\varsigma^2(k)$ be an input data process sequence, and l_i^j be a member of $\varsigma^2(k)$, where $l_i^j.t > \phi_{min} + 2H$. The members of $\varsigma^{\Delta H_1}(k)$ have to be processed before l_i^j. Let l_a^b be a member of $\varsigma^{\Delta H_1}(k)$. l_a^b can be processed at a later time $\phi_a + (H/Pa - 1)P_a + rj_{a,max}$, which is equal to $\phi_a + H$ according to the restriction of maximum release time jitter of a task is the period of the related task. As the relation $\phi_a - \phi_{min} < H$ holds, all members of $\varsigma^{\Delta H_1}(k)$ are guaranteed to be processed before l_i^j. Therefore, $\varsigma^2(k)$ is similar with an input data process sequence like $\varsigma^{\Delta H_1}(k).\varsigma^3(k)$. According to Proposition 9 there exists an input data process sequence $\varsigma^1(k)$, which $\varsigma^3(k)$ is congruent to $\varsigma^1(k)$ shift H. Let g_2 be equal to $f(g_{init}, \varsigma^2(k))$, and g_1 be equal to $f(g_{init}, \varsigma^1(k))$. According to Lemma 2, the abstract successor state set of g_2 is equal to the abstract successor state set of g_1, i.e. $\mathcal{A}(Post(g_1)) = \mathcal{A}(Post(g_2))$. Therefore, the task set has finite abstract result states. □

Corollary 1. All finite abstract computational result states of a task set can be obtained in a time interval $[0, \phi_{min} + 2H]$.

3.3 Data Abstraction Algorithm

In this section we present an algorithm that constructs a finite state automaton of computational result data of a task set in accordance with the abstraction method presented above. The input of the algorithm is a task set in which each task is defined with a ternary structure whose attributes are task ID, task period, P, and offset of the task, ϕ. An automaton is a seven-tuple $M = (Q, \Sigma, \Lambda, \delta, q_0, \lambda)$ in which Q is the finite set of states, Σ is the finite set of transition actions, $\delta : Q \times \Sigma \mapsto Q$ is the finite

set of transition actions, q_0 is the initial state, Λ is set of state output propositions, and $\lambda : Q \mapsto \Lambda$ is the state output labeling function. For the automaton representing a computational result data, a transition action corresponds to the processing of an input data item by a task in the task set. According to the abstraction described in Equation 3.26, the set of transition actions is the task IDs in the task set, $\Sigma = \{\tau_1.id, \tau_2.id, ..., \tau_N.id\}$. The set of state output propositions is $\Lambda = \{\alpha, \varepsilon\}$ according to Equation 3.27.

Algorithm 1: GLOBAL RESULT FINITE STATE AUTOMATON CREATER

DataStructure: $task(ID, P, \phi)$

DataStructure: Input Data Entity $l = (ID, TaskID, t)$

DataStructure: $State(ID, \varsigma(k), output, Post)$

DataStructure: $SuccState(l, State)$

DataStructure: $Node(ID, \varsigma(k), l, children)$

Input: Task Set $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_N\}$

Output: Finite State Automaton $(\Sigma, Q, q_0, \delta, F)$

$\Delta E =$ `FindExaminationTimeWindow`$(\mathcal{T})$;

$\mathbb{L}^{\Delta E} =$ `CreateLocalDataEntityDomain`$(\mathcal{T}, \Delta E)$;

$root = new\ Node(0, null, l(0, 0, 0), null)$;

`CreateProcessGraph`$(root, \mathbb{L}^{\Delta E})$;

$StateinitState =$ `CreateState`$(root)$;

$StateSpace.add(initState)$;

`CreateStateSpace`$(StateSpace, root)$;

$(\Sigma, \Lambda, Q, q_0, \delta, \lambda) =$

`CreateFiniteStateAutomaton`$(initState, StateSpace)$;

The algorithm first determines the time window, denoted ΔE, to create and examine all possible input data process sequences in the examination window. Secondly, input data entity domain in the examination time window, $\mathbb{L}^{\Delta E}$, is determined. Afterwards, a process graph is constructed and all possible input data process sequences are explored. By examining the input data process sequences, abstracted states of all possible computational results are obtained and the state space is generated. Finally, the algorithm constructs a finite state automaton, $M = (\Sigma, \Lambda, Q, q_0, \delta, \lambda)$. The com-

putational result automaton has no final state. Algorithm 1 constructs an automaton for computational result.

The examination time window, ΔE, is determined according to Corollary 1 which stipulates that it is possible to determine all computational result states in two hyperperiod intervals. All Procedures mentioned in this section hereafter are presented in the Appendix to this dissertation. Procedure 1 describes ΔE. After determination of ΔE, the algorithm constructs the input data entity domain at as described in Procedure 2. By using the input data entity domain in ΔE, the process graph is created with respect to $L^{\Delta E}$. In the process graph an edge is labeled by an input data entity, and a node represents a computational result state obtained by processing the input data entity that is the label on an incoming edge. A path to a node represents an input data process sequence. The process graph is constructed using the $Node$ data structure. Each node has a unique ID, input data process sequence, incoming edge label, and outgoing children nodes attributes. The construction of a process graph starts with a reference node. The initial reference node is described as $Node(0, null, l(0, 0, 0), null)$ in Algorithm 1. In Procedure 4, the successor input data entity set is determined according to Equation 3.21. For each input data entity element of a successor input data entity set, a new node is created by assigning the attribute values. Procedure 4 describes the creation of a process graph.

By using the process graph all possible input data process sequences can be described within ΔE. The described algorithm constructs the state space of computational result by defining a state for each node as described in Procedure 3. A state is represented by a $State$ data structure in the algorithm. Each state has a unique ID, input data process sequence, abstract output, and successor state ($Post$) attributes. The output attribute of a state is determined according to the output function described in Equation 3.27. Each state in the state space corresponds to an abstract state of computational result according to the abstraction method. The algorithm does not add a state to the state space, if there already exists an equivalent state. Equivalence of states are examined in Procedure 6 according to Definition 15. State space construction is described by Procedure 5. Finally, the algorithm constructs the finite state automaton as described in Procedure 7.

CHAPTER 4

CASE STUDY: MULTI SENSOR TRACK TO TRACK DATA FUSION

4.1 Multi Sensor Data Fusion and Out of Sequence Track Problem

Multi sensor data fusion (MSDF) corresponds to the process of combining information from different sensors to provide a robust and complete description of objects or events on an observed environment. Combining the outputs of multiple sensors that can observe different signatures in the environment usually provides more accurate information than using a single sensor. Normally, to increase the sorts of information that can be sensed, an MSDF system often consists of dissimilar sensors. Moreover, for the same number of resources, sensors with different data rates (asynchronous sensors) can provide better coverage than synchronous sensors [33]. In a centralized fusion architecture there is a single data fusion node to aggregate all sensory data [35]. In many practical situations, because of communication constraints, each local sensor has its own information processing unit and outputs tracking reports rather than raw measurements to the fusion node [34], in a configuration referred as a track to track data fusion system. The central fusion node produces a system track by aggregating sensory local tracks.

A multi sensor track to track data fusion system consists of:

1. A collection of sensors that generate local tracks,
2. An central processing component, supervisory controller, to aggregate sensory local tracks, manage sensors and possibly actuators,

3. A high-speed communication network to enable the process.

Figure 4.1 depicts a sample track to track data fusion system architecture. A sensor S_i makes observations, Y_i, and generates tracks that consist of estimation, $\hat{X}_i$, and error covariance matrix of the estimation P_i. Kalman Filtering (KF) [36] is the most commonly used technique in target tracking to generate local tracks from sensor observations [37]. At the supervisory controller, low-level data processing is implemented. Low-level data processing involves data association, data alignment, sensor registration, and the position and ID fusion [38]. At the end of low-level data processing, the supervisory controller aggregates local tracks by using a track to track fusion algorithm and updates central system tracks. The supervisory controller manages sensors to start or stop tracking of a specified target.

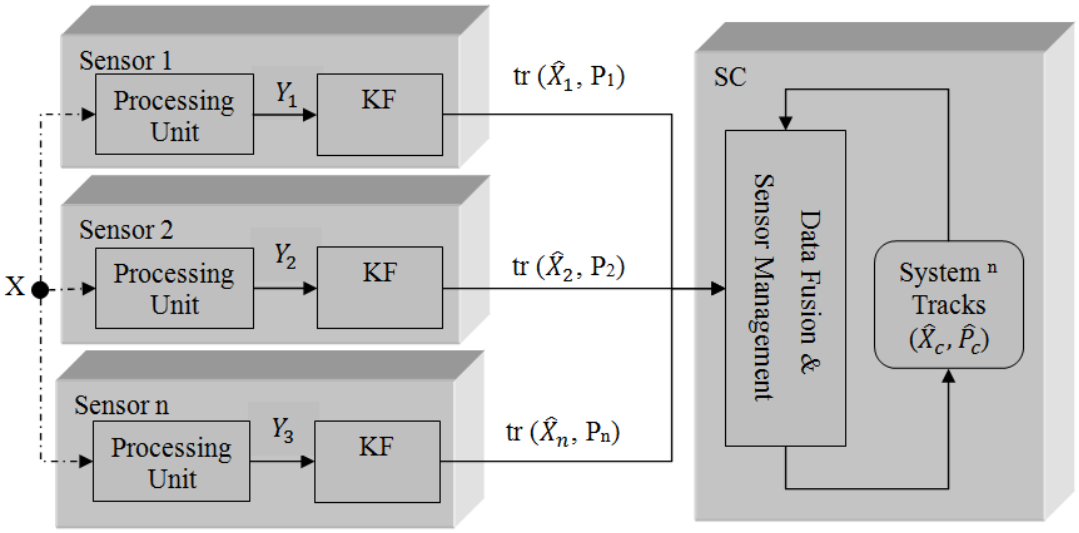
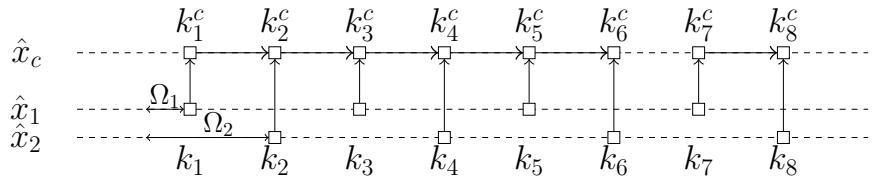


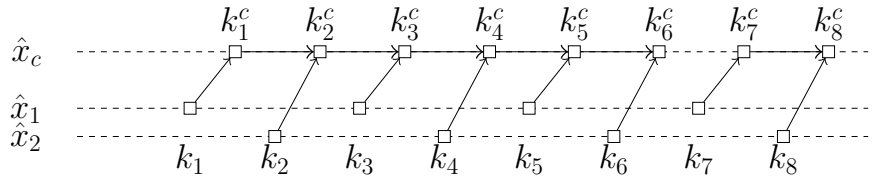
Figure 4.1: Track to Track Data Fusion System

In an MSDF system each sensor operates with its own clock, and may generate tracks at a different rate than the others because of its signature characteristics and processing speed. Moreover, the communication network may lead to different communication delays for different sensor tracks. Therefore, the MSDF system has to consider the coordination between sensors and cater for parameters such as different sampling rates and communication delays [45]. Communication delay in the transmission of a local track from sensors to the fusion center may result in the reception of sensory tracks out of sequence; that is, there may be no guarantee that sensory tracks are fused in the order they have originated [39]. This incident is known as the out-of-sequence

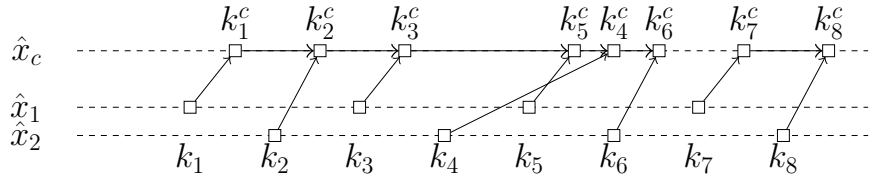
track (OOST) problem. OOST problem corresponds to a situation in which a central track is already updated up to a given time and then, a track from a "late" sensor arrives with an earlier time stamp [40]. In order to tackle the delayed and OOST problem, algorithmic solutions have been proposed [39, 40, 41, 42, 43, 44]. These solutions aim to aggregate sensory track estimations sequentially, thereby guaranteeing the consistency of central track estimation. However, fusion of out of sequence tracks cause increase in the central track estimation error until an in order local track is aggregated.



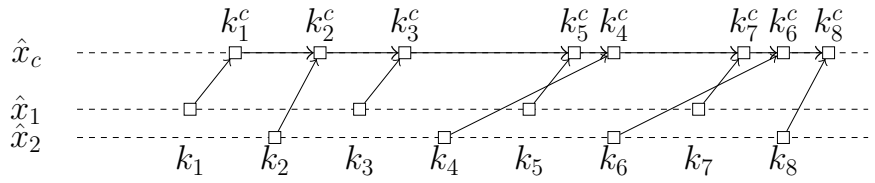
(a) Case 1



(b) Case 2



(c) Case 3



(d) Case 4

Figure 4.2: Track-to-Track Fusion

In order to illustrate the effect of OOST on central track estimation error, a simple system consisting of two sensors and a data fusion center is simulated with four different system configurations. The sensors observe the environment and generate periodical tracks. The periods of sensors P_1, and P_2 are equal to each other. Each

sensor starts the tracking process with different offsets, Ω_1, and Ω_2. Sensors' offsets are caused by track command communication delays. The phase difference between sensor periods, $\varphi_{12} = \Omega_2 - \Omega_1$, is equal to $P/2$. In the first case, the supervisory controller receives and aggregates the sensor tracks without any communication delay $\tau_k = \tau_k^1 = \tau_k^2 = 0$, for all k as seen in Figure 4.2.a. In the second case the supervisory controller receives and aggregates the sensor tracks with a communication delay $P/10$, $\tau_k = \tau_k^1 = \tau_k^2 > 0$ for all k as seen in Figure 4.2.b. The third case treats the OOST scenario. The supervisory controller receives and aggregates the sensory tracks with communication delay $P/10$ for all k except $k_{22} = 211$. The track report of Sensor 2, generated at k_{22}, is received and aggregates with a time delay $3P/4$ after the track generated at a later time, k_{21}, as seen in Figure 4.2.c. In the forth case, illustrated in Figure 4.2.d, two successive tracks of sensor 2, generated at $k_{22} = 211$ and $k_{24} = 231$ are out of sequence.

In the simulation the target moves in two dimensions according to Equation 4.1 where F is the system matrix, and $W(k)$ is the zero-mean Gaussian white noise with covariance Q, in which $Q = q * I$, I is a 4x4 dimension identity matrix, and $X(k)$ is the target state vector, which consists of $[\ x_x(k) \quad x_y(k) \quad \dot{x}_x(k) \quad \dot{x}_y(k)\ ]^T$ where $[x_x(k), x_y(k)]$, and $[\dot{x}_x(k), \dot{x}_y(k)]$ represent position and velocity in the X and Y directions, respectively.

$$X(k+1) = FX(k) + W(k), \quad W = N(0, Q) \tag{4.1}$$

The target state vector is generated for time instances, $\{k\}_{k=1}^{N}$ according to Equation 4.1, and parameters, in which the system matrix is

$$F = \begin{bmatrix} 1 & 0 & \Delta T & 0 \\ 0 & 1 & 0 & \Delta T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $\Delta T = 0.05$. The initial state vector x[0] = $[\ 10 \quad 10 \quad 1 \quad 0\ ]^T$, and error covariance factor q = 1.

A sensor i makes observations, $Y_i(k)$, periodically by using target state vector accord-

ing to equation

$$Y_i(k) = HX(k) + V_i(k), \quad V = N(0, R_i) \tag{4.2}$$

where H is the measurement matrix, $X(k)$ is the target state vector, and $V_i(k)$ is the zero-mean white Gaussian measurement noise with covariance R_i, where $R_i = r_i * I$, and I is a 2x2 identity matrix. Sensors derive estimates from observations, using a Kalman Filter and generate tracks, consisting of sensor estimation, $\hat{X}_i(k)$, and covariance matrix, $P_i(k)$ according to Equation 4.3.

$$\hat{X}_i(k), P_i(k) = KF(\hat{X}_i(k-1), P_i(k-1), Y_i(k), F, H, Q, R_i) \tag{4.3}$$

In the simulation, the sensors make observations with the same period, $T = 20$, but with a phase difference between observation periods, where $\varphi_{12} = 10$. The parameters used in Equation 4.2 and Equation 4.3 denote the measurement matrix

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and the error covariance factor r_1 = 1.5, and r_2 = 1.1.

The supervisory controller updates its central estimation from the input provided by the sensory tracks. The asynchronous and OOST reports are aggregated by using Information Matrix Fusion (IMF) algorithm [44] in order to update central tracks. Equations 4.4 and 4.5 below, respectively, describe how the supervisory controller updates central track covariance and state vector.

$$P_c(k|k) = (P_c(k|k-1)^{-1} + (P_i(k|k))^{-1} - (P_i(k|k-1))^{-1})^{-1} \tag{4.4}$$

$$\begin{aligned} \hat{X}_c(k|k) = P_c(k|k)(P_c(k|k-1)^{-1}\hat{X}_c(k|k-1) + \\ P_i(k|k)^{-1}\hat{X}_i(k|k) - P_i(k|k-1)^{-1}\hat{X}_i(k|k-1)) \end{aligned} \tag{4.5}$$

where the prediction of state vector and track covariance are gathered by:

$$\hat{X}_c(k|k-1) = F(k)\hat{X}_c(k-1|k-1)$$

$$P_c(k|k-1) = F(k)P_c(k-1|k-1)F^T(k) + Q(k)$$

In order to evaluate the effect of delayed and out of sequence track reports, the root mean square error (RMSE) and trace of central error covariance matrix are calculated according to expressions 4.6 and 4.7 below, respectively. $X_p^i(k)$ denotes the true target position state, and $\hat{X}_{c,p}^i(k)$ denotes central estimation of target position at time k for simulation run i.

$$RMSE(k) = \sqrt{\frac{1}{M}\sum_{i=1}^{M} \|X_p^i(k) - \hat{X}_{c,p}^i(k)\|^2} \tag{4.6}$$

$$trace\{P(k)\}\ in\ position = \sum_{i=1}^{2} P(k)_{ii} \tag{4.7}$$

Figure 4.3 shows RMSE in position of central estimation in the first three cases for 250 simulation runs, and Figure 4.4 shows RMSE of central estimation in the first, second and fourth cases for 250 simulation runs. RMSE in case 1 is less than the RMSE in case 2. The occurrence of OOST causes an increase in the RMSE. The increase in RMSE continues if the successive out of sequence track reports are fused . The increase in RMSE continues until an in sequence track report is fused. Figures 4.5 and 4.6 show the trace of central error covariance which is consistent with RMSE behavior.

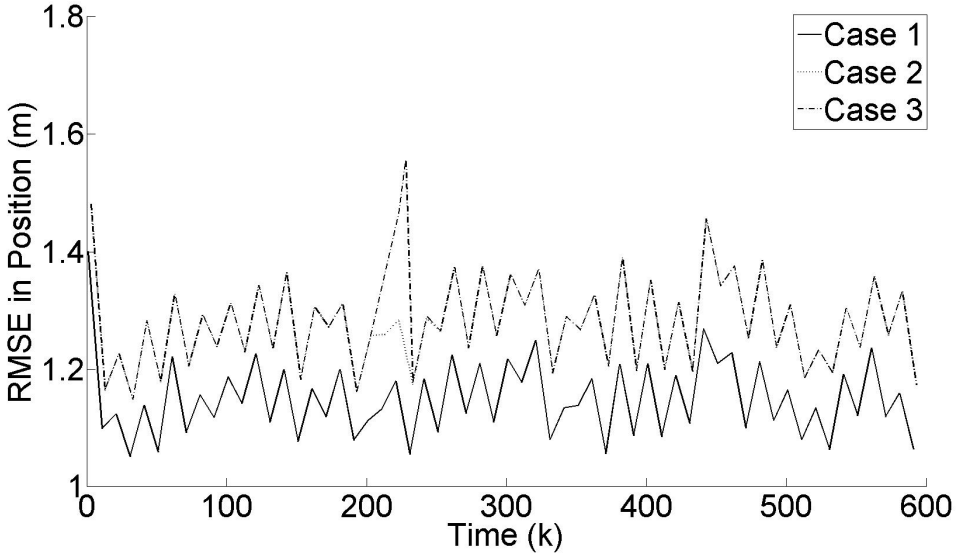


Figure 4.3: RMSE for cases 1, 2, and 3

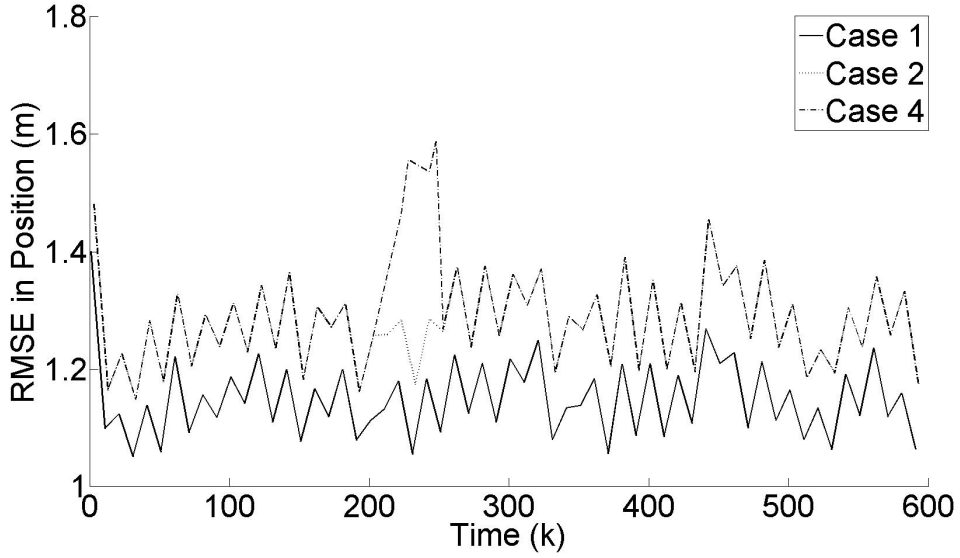


Figure 4.4: RMSE for cases 1, 2, and 4

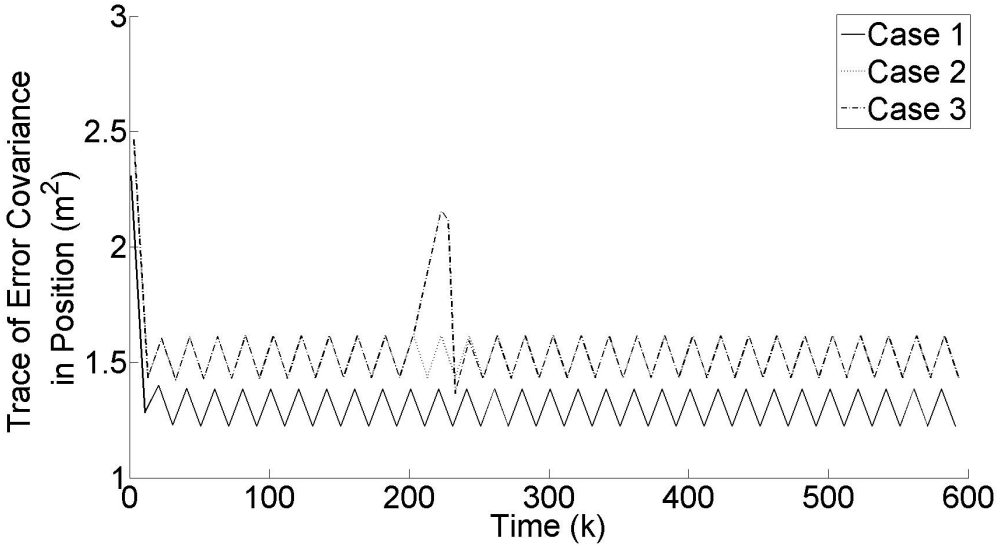


Figure 4.5: Trace of P(k) for cases 1, 2, and 3

Simulation results show that data fusion with communication delay in track report transmission and OOST problem are properly handled by the IMF algorithm. Even though fusion of an OOST report may cause an increase in the central estimation error, the IMF algorithm bounds the increase when the in sequence track is fused. However, from the point of view of modeling, the deviation in the estimation error originating from the OOST problem is important and should be properly handled for formal verification.

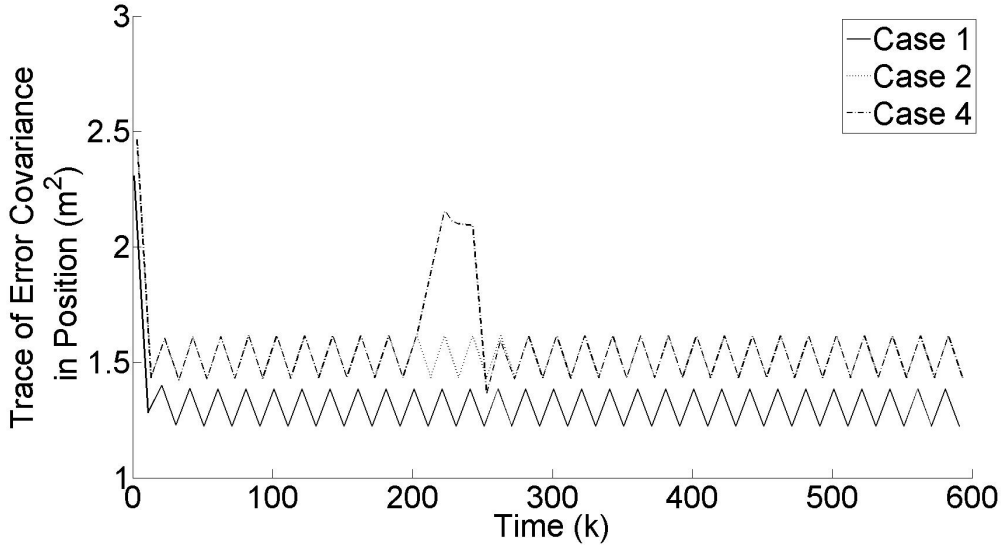


Figure 4.6: Trace of P(k) for cases 1, 2, and 4

4.2 Abstraction of Central Track Data

Formal verification of MSDF system via model checking requires modeling both the control behavior and data. MSDF system contains local and central track data over infinite domains. Moreover, the OOST problem causes degradation in the usefulness of central track data in MSDF systems and affects the control behavior. Therefore, representation of central track data with finite states is necessary for model checking by taking the OOST problem into account. To represent OOST problem properly in the system model, temporal behavior of the system should be represented. Temporal behavior of the system consists of sensor periods, phase difference between sensor periods and transmission delay of tracks from sensors to the supervisory controller. For finite state representation of the central track we apply the proposed abstraction method proposed in Chapter 3. The system to be modeled consists of a parallel collection of central track automaton and task automata:

$$A_{CT}||A_{\tau_1}||A_{\tau_2}||\cdots||A_{\tau_N} \tag{4.8}$$

Each task in the task set is dedicated to a sensor in the MSDF system that will aggregate and compute central track from the local tracks received from sensors.

The periods and phases of sensors are represented in the automaton corresponding to each task. The communication delay between a sensor and the supervisory controller is represented by release time jitter in the task automaton. A task automaton is defined with a template $A_\tau(intiID,\ int\ iOffset,\ int\ iRTJitterMin,\ int\ iRTJitterMax,\ int\ iPeriod)$, where the $iOffset$ parameter defines the offset of the task, the $iRTJitterMin$ and $iRTJitterMax$ parameters define the release time jitter range, and the $iPeriod$ parameter defines the period of the task. Figure 4.7 illustrates the behavior of the timed task automata.

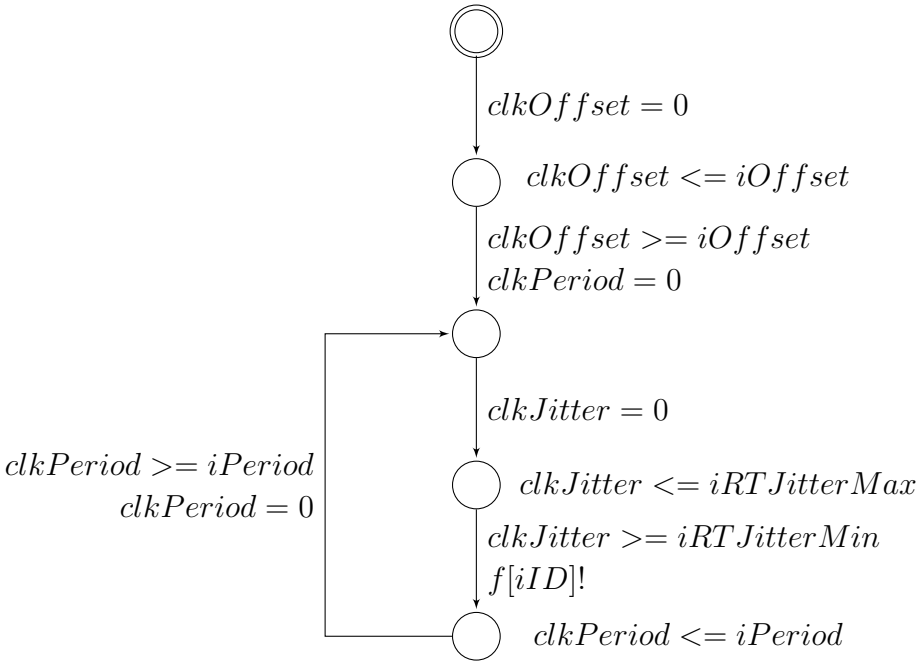


Figure 4.7: Task Automaton A_{τ_n}

The timed task automaton, A_τ, uses three clock variables to represent the timing requirements about offset, release time jitter and period. These clock variables are $clkOffset$, $clkRTJitter$, and $clkPeriod$, respectively. At the beginning A_τ starts to wait the offset time. When the offset time is reached A_τ starts its periodic process to update the central track. In the periodic cycle, A_τ generates $f[iID]$ action by taking the release time jitter into account. $f[ID]!$ represents the action generation of processing local track and update of central track.

Central track automaton, A_{CT}, models central track data over infinite domain with finite states according to the abstraction method presented in Chapter 3. The central track automaton covers all possible fusion orders of local tracks related with a

target, and represents conformable and anomalous error states for central track estimation. Each action of update of central track is represented by $f[ID]?$. Figure 4.8 depicts the central track automaton that models the central track data in a system configuration in which the target is tracked by 2 sensors, which are asynchronous with $\varphi = P/2$, having the same report generation periods, P, and the worst case communication delay for both sensors is P. In this configuration there are two task automata $A_\tau(0, 0, 0, P, P)$ and $A_\tau(1, P/2, 0, P, P)$. The central track automaton consists of 6 abstract states, $Q = \{q_0, q_1, ..., q6\}$, where q_0 is the initial state. The finite set of transition actions is $\{f[0], f[1]\}$. While the states q_1, q_4 are labeled as the conformable state output, states q_0, q_2, q_3, q_6 are labeled as the anomalous state output.

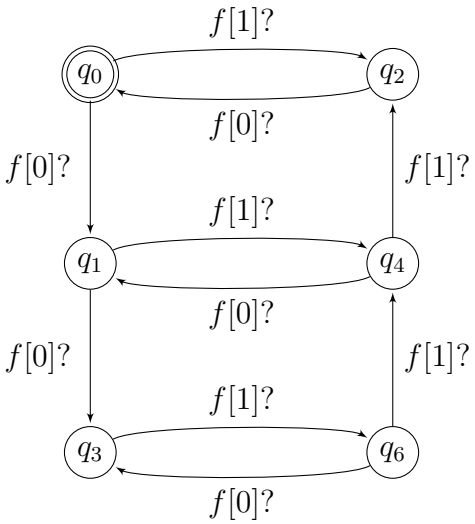


Figure 4.8: Central Track Automaton ($P_1 = P_2$)

Figure 4.9 shows the central track automaton in a larger system configuration in which a target is tracked by 3 sensors, which are asynchronous with $\varphi = P/3$, having the same report generation periods, P, and the worst case communication delay for all sensors is . In this configuration there are three task automata $A_\tau(0, 0, 0, P, P)$, $A_\tau(1, P/3, 0, P, P)$, and $A_\tau(2, 2P/3, 0, P, P)$. The central track automaton consists of 15 abstract states, $Q = \{q_0, q_1, ..., q_{14}\}$, where q_0 is the initial state. The finite set of transition actions is $\{f[0], f[1], f[2]\}$. While the states q_1, q_2, $q_3$are labeled as the conformable state output, all other states are labeled as the anomalous state output.

Central tracks, which are estimations of the state of a tracked target, effects the behavior of a MSDF system. However, target state and estimations have very large or

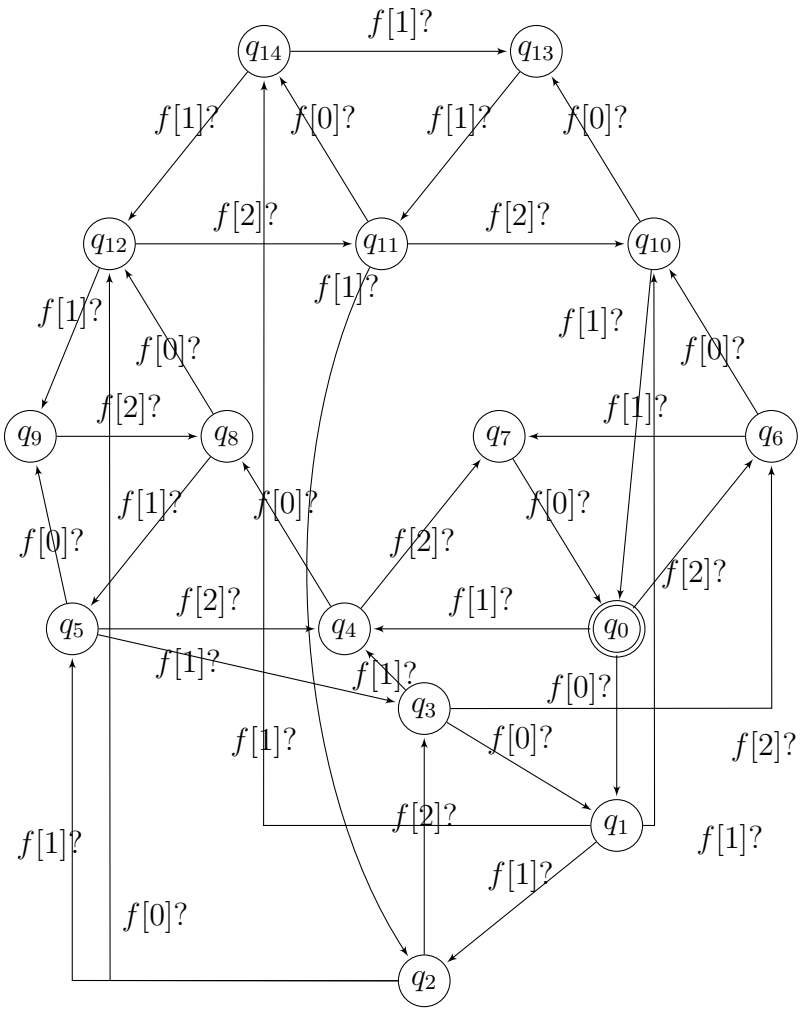


Figure 4.9: Central Track Automaton ($P_1 = P_2 = P_3$)

infinite data domains, and OOST problem causes additional expansion in the central track data domain by increasing the error in the central track estimation. Therefore, modeling local and central tracks with an abstraction by taking the OOST problem into account is necessary to apply model checking. The presented abstraction method is applied to represent the central tracks with abstraction in the system model. The increase in the estimation error because of OOST problem is addressed as degradation in the usefulness of central track estimation. The distributed sensor deployment is modeled as concurrent tasks automata, in which the communication delays between sensors and supervisory controller are represented with release time offset and jitter properties. Two different system configurations with varying number of sensors are inspected in the case study. The sensors are configured as having the same period and different release time offset to obtain most complex configuration. Abstraction of

the central track in the first configuration with 2 sensors results in an abstract central track automaton with 6 states and 10 transitions. Abstraction of the central track in the second configuration with 3 sensors results in an abstract central track automaton with 15 states and 29 transitions. The number of states and transitions in the abstract central track automata shows that the presented abstraction method enables enables to modeling the central track with finite states and transitions by preserving the representation of increase in the estimation error.

CHAPTER 5

CONCLUSION

In this study we have presented a data abstraction method for infinite domain data produced by a set of concurrent, asynchronous and periodic tasks collectively in a real time system. We developed the presented method according to two research questions:

1. Determination of an abstract set of data values;
2. Constructing a mapping between concrete data values and the determined abstract data set, preserving model characteristics for model checking.

The first question has been answered in terms of the real time properties of the task set, and a finite abstract data domain has been identified with respect to the usefulness of the computational result. The concrete data domain is abstracted according to the degradation in the usefulness of result. The abstract data domain contains two data values, which represent whether the usefulness is degraded or not.

To answer the second question, the concurrency and asynchrony in the real time system under consideration are investigated and sequentialization of the execution of concurrent tasks in the task set are identified by taking the asynchrony into account. According to the sequentialization of concurrency, rules to derive all possible input data process sequences in the task set are derived. Characteristics of the common operation of the task set, which cause degradation in usefulness of computational results when out of sequence input data is processed, and eliminates that degradation when in sequence input data is processed, have been formally defined. According to the

input data process sequences and functional operation characteristics, mapping rules of concrete data values to abstract data values have been formulated and the finiteness of the domain of abstract data values has been proved.

In agreement with the definition of the abstract data domain and the data abstraction mapping, an algorithm has been developed to construct that mapping. The algorithm constructs a labeled abstract data automaton according to the defined system properties.

To evaluate the contributions of this study, a case study has been conducted. In the case study, a multi sensor track to track data fusion system, which contains rich data, has been investigated and formal finite state models of data abstraction have been presented for two different system configurations.

Below, we summarize the contributions and limitations of this research, derive conclusions from the study and suggest future research directions based on the findings.

5.1 Contributions

The main contribution of this study is a data abstraction method that enables formal modeling of infinite domain data which has been produced by a periodic, concurrent and asynchronous task set. In this respect the formal definitions of the dependencies on the temporal behavior of the system, and proofs about the finiteness of the abstraction mapping have been presented.

Moreover, proposed data abstraction method contributes to avoiding the state explosion problem [15] even if the domain is finite, in two ways: State explosion problem refers to the enormous increase in the population of the global state space of a system model and is a serious drawback of model checking. The main reason for the state explosion problem is concurrency and asynchrony in the system model. The first contribution of the present study to the reduction of the state explosion problem is achieved by reducing the number of global states by bounding the concrete data domain using abstraction. The second contribution is achieved by representing the asynchronous communication channels of a distribution system with a set of task au-

tomata. Modeling asynchronous communication channels increases the number of global states in the system model, because it requires modeling additional queues and clocks to represent delays in asynchronous communication. Task automata uses release time offset and jitter to represent asynchronous communication properties.

Finally, a practical but rather valuable contribution is that specification of abstract data with finite state automata enables using model checking tools that do not allow variables to specify data.

5.2 Limitations and Future Work

The presented abstraction method is suitable for real time systems in which a computational result is produced by periodic tasks. The abstraction of computational result data produced by aperiodic and sporadic tasks are not handled in this study.

Also, the operation common to all members of the task set has to have mechanisms to cope with out of sequence data processing. In this study, we deal with the operations that eliminate the degradation in usefulness when in sequence data is processed to enable finiteness in the abstraction mapping. Other types of mechanisms are out of the scope of this study. As the short term future work, this research can be enriched with developing a mapping method for other types of mechanisms.

Furthermore, even if the presented abstraction method contributes to tackling the state explosion problem, it still remains vulnerable to it. The increasing number of tasks in the task set possibly cause higher levels of interleaving between task executions, and possible input data process sequences. Hence, the number of global states increases exponentially with the increasing number of tasks in the task set. The case study carried out in Chapter 4 exposes this situation with the two system configurations with different numbers of tasks. Developing architectural system design patterns to bound the asynchronous data interactions to reduce the state space and the system complexity in the system behavior might be performed as another future work.

Currently we are able to abstract the concrete data domain of computational result with a binary abstract data domain; that represent whether the usefulness is degraded

or not.

REFERENCES

[1] C. Baier, K. Joost-Pieter. Principles of model checking. Vol. 26202649. Cambridge: MIT press, 2008.

[2] E. M. Clarke, O. Grumberg, D.Peled. Model Checking. MIT Press,1999.

[3] J.W.S. Liu. Real-Time Systems. Prentice Hall,2000

[4] G.C.Buttazzo, G. Lipari, L. Abeni, M. Caccamo. Soft Real-Time Systems: Predictability vs. Efficiency. Springer, 2005

[5] A. Burns , A. Wellings. Concurrent and Real-Time Programming in Ada. Cambridge University Press, New York, NY, 2007

[6] M. Stonebraker, U.Çetintemel, S. Zdonik. The 8 requirements of real-time stream processing. ACM SIGMOD Record, 34(4), 42-47, 2005.

[7] J. Esparza. An automata-theoretic approach to software verification. In Developments in Language Theory, volume 2710 of LNCS, page 21. Springer, 2003.

[8] R. Alur , D. L. Dill. The theory of timed automata. Real-Time: Theory in Practice. volume 600 of Lecture Notes in Computer Science, pages 45–73, Springer Berlin Heidelberg, 1992.

[9] R. Alur , D. L. Dill. A theory of timed automata. Theoretical Computer Science, 126(2):183–235, 1994.

[10] R. Alur, C. Courcoubetis, D. L. Dill. Model-Checking for Real- Time Systems. In Proc. of the 5th Annual IEEE Symposium on Logic in Com- puter Science, 1990.

[11] J. Dingel, T. Filkorn. Model Checking for infinite state systems using data abstraction, assumption-commitment style reasoning and theorem proving. In Computer Aided Verification, pp. 54-69. Springer Berlin Heidelberg, 1995.

[12] T.A. Henzinger, X. Nicollin, J. Sifakis, S. Yovine. Symbolic model checking for real-time systems. Information and Computation, 111 (2), 193–244, 1994.

[13] E. M. Clarke, L. Flavio, T. Muralidhar. An abstraction technique for real-time verification. Next Generation Design and Verification Methodologies for Distributed Embedded Control Systems, Springer Netherlands, 1-17, 2007.

[14] N. Bogunovic, P. Edgar. Model checking procedures for infinite state systems. Engineering of Computer Based Systems, 2006, ECBS 2006, 13th Annual IEEE International Symposium and Workshop on. IEEE, 2006.

[15] E. M. Clarke, W. Klieber, M. Novacek , P. Zuliani. Model Checking and The State Space Explosion Problem, 2005

[16] E. W. Dijkstra. Guarded Commands, Nondeterminency, and Formal Derivation of Programs. Programming Languages, 1975

[17] R. Milner. Communication and Concurrency. Prentice-Hall International, Englewood Cliffs, 1989

[18] C.A.R.Hoare. Communicating Sequential Circuits. Prentice-Hall International, 1985

[19] G. M. Reed, A. W. Roscoe. A timed model for communicating sequential processes. In Lecture Notes in Computer Science, Automata, Languages, and Programming. Heidelberg, Germany: Springer-Verlag, 1986, vol. 226, pp. 314–323

[20] C. A.Petri. Kommunikation mit Automaten. PhD thesis, Institut fur Instrumentelle Mathematik, Bonn, 1962

[21] P.M. Merlin, D.J. Farber. Recoverability of communication protocols – implications of a theorical study. IEEE Transactions on Communications, 24, 53–70, 1976.

[22] J. Srba. Comparing the expressiveness of timed automata and timed extensions of Petri nets. In Formal Modeling and Analysis of Timed Systems. pp. 15-32, Springer Berlin Heidelberg, 2008.

[23] S. Yovine. KRONOS: a verification tool for real-time systems. International Journal on Software Tools for Technology Transfer, 1(1-2):123–133, December 1997.

[24] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL—a tool suite for automatic verification of real-time systems. In Lecture Notes in Computer Science, Hybrid Control Systems. Heidelberg, Germany: Springer-Verlag, vol. 1066, pp. 232–243, 1996.

[25] F. Wang. RED: Model-Checker for Timed Automata with Clock-Restriction Diagram. In Proc. of Workshop on Real-Time Tools, 2001.

[26] F. Laroussinie, K. G. Larsen. CMC: a tool for compositional model-checking of real-time systems. In Proc. IFIP TC6/WG6.1 Joint Int. Conf. Formal Description Techniques and Protocol Specification, Testing, and Verification, pp. 439–456, 1998.

[27] E. A. Emerson. The beginning of model checking: A personal perspective. Springer Berlin Heidelberg, 2008. 27-45.

[28] A. Pnueli. The temporal logic of programs. In 18th IEEE Symposium on Foundations of Computer Science (FOCS), pages 46–67. IEEE Computer Society Press, 1977.

[29] R. E. Bryant. Graph-based algorithms for boolean function manipulation. IEEE Transaction on Computers, pages 35(8):677–691, 1986.

[30] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic Model Checking. J. ACM, 50(5):752-794,2003.

[31] E.M. Clarke, O. Grimberg,D. E. Long, Model Checking and Abstraction,ACM Transactions on Programming Languages and System (TOPLAS), 16(5):1512–1542, September 1994.

[32] A. Biere, E. M. Clarke, R. Raimi, and Y. Zhu. Verifying Safety Properties of a Power PC Microprocessor Using Symbolic Model Checking without BDDs. In CAV, 1999.

[33] A. T. Alouani, J. E. Gray, D. H. McCabe. Theory of distributed estimation using multiple asynchronous sensors. Aerospace and Electronic Systems, IEEE Transactions on, 41(2), 717-722, 2005.

[34] T. Yuan, Y. Bar-Shalom, X. Tian. Heterogeneous track-to-track fusion. In Information Fusion (FUSION), 2011 Proceedings of the 14th International Conference on (pp. 1-8), IEEE,2011, July.

[35] M. E.Liggins, C. Y. Chong, I. Kadar, M. G. Alford, V. Vannicola, S. Thomopoulos. Distributed fusion architectures and algorithms for target tracking. Proceedings of the IEEE, 85(1), 95-107,1997.

[36] R. Kalman. A New Approach to Linear Filtering and Prediction Problems. Trans. ASME, J. Basic Eng., no. 82, pp. 34-45, 1960.

[37] Y. Bar-Shalom, X. R. Li. Multitarget-Multisensor Tracking: Principles and Techniques. YBS Publishing, Storrs, CT ,1995.

[38] A. Gad, M. Farooq, Data Fusion Architecture for Maritime Surveillance. Proceedings of the Fifth International Conference on Information Fusion, 2002.

[39] S. Challa, J. A. Legg, X. Wang. Track-To-Track Fusion Of Out-Of-Sequence Tracks. Proceedings of the Fifth International Conference on Information Fusion ,2002.

[40] A. Novoselsky, S.E. Sklarz, M. Dorfan. Track to track Fusion using Out-of-Sequence Track Information. Proceedings of the Fifth International Conference on Information Fusion, 2002.

[41] X. Tian and Y. Bar-Shalom. The optimal algorithm for asynchronous track-to-track fusion. In Proceedings of SPIE Conference on Signal and Data Processing of Small Targets, 7698-46 ,2010.

[42] X. Tian , Y. Bar-Shalom. Algorithms for asynchronous track-to-track fusion. Journal of Advances in Information Fusion, vol. 5, no. 2, pp. 128-138, 2010.

[43] X. Tian , Y. Bar-Shalom. On Algorithms for Asynchronous Track-to-Track Fusion. 13th Conference on Information Fusion, Edinburgh, Scotland, pp. 1-8, 2010.

[44] M. Aeberhard, A. Rauch, M. Rabiega, N. Kaempchen,T. Bertram. Track-to-Track Fusion with Asynchronous Sensors and Out-of-Sequence Tracks using Information Matrix Fusion for Advanced Driver Assistance Systems. Intelligent Vehicles Symposium (IV), 2012.

[45] Z. Kejun, S. Jianbo. A New Method for Asynchronous Multisensor Information Fusion. Lecture Notes in Computer Science Volume 3238, pp 410-423, 2004.

[46] E. M. Clarke, E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In Logic of Programs, volume 131 of Lecture Notes in Computer Science, pages 52–71. Springer-Verlag, 1981

[47] E. M. Clarke, E. A. Emerson, A.P. Sistla. Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications: A Practical Approach. In POPL, pages 117-126, 1983

[48] Y-W.Hsieh, P. L. Steven. Model abstraction for formal verification. In Proceedings of the conference on Design, automation and test in Europe, pp. 140-147. IEEE Computer Society, 1998.

[49] D. Kroening, A. S. Sanjit. Formal verification at higher levels of abstraction. In Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design, pp. 572-578. IEEE Press, 2007.

[50] E. M. Clarke, A. Biere and R. Raimi and Y. Zhu. Bounded model checking using satisfiability solving. Formal Methods in System Design, 19(1):7–34, 2001.

[51] S. Graf, S. Hassen. Construction of abstract state graphs with PVS. In Computer aided verification, pp. 72-83. Springer Berlin Heidelberg, 1997.

APPENDIX A

DATA ABSTRACTION ALGORITHM PROCEDURES

Procedure 1: FindExaminationTimeWindow

ArgumentIn: Task Set $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_N\}$

Return: Examination Time Window ΔE

$HyperPeriod = LCM(\{P_i\}_{i=1}^{N})$;

$\Delta E = 2 \times HyperPeriod$;

return ΔE;

Procedure 2: CreateLocalDataEntityDomain

ArgumentIn: Task Set $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_N\}$

ArgumentIn: Examination Time Window ΔE

Return: Domain $\mathbb{L}^{\Delta E}$

$ID = 0$;

for $i : 1$ ***to*** N **do**

    $t = \tau_i.\phi$;

    while $t \; is \; in \; \Delta E$ **do**

        $\mathbb{L}^{\Delta E}.add(l(ID, \tau_i.I, t))$;

        $ID++$;

        $t += \tau_i.P$;

    end

end

return $\mathbb{L}^{\Delta E}$;

Procedure 3: CreateState

ArgumentIn: Process Graph node n

Return: State state

Data: Set of Input Data Entities $L_{<}(l)$ and $L_{>}(l)$

$State\ state = State(stateId, null, 0, null)$;

if $n.\varsigma(k) \neq null$ **then**

  $state.\varsigma(k) = n.\varsigma(k)$;

  $N = state.\varsigma(k).size$;

  $L^{N-1}_{\varsigma(k)}(k) = L_{\varsigma(k)}(k) - n.l$;

  foreach *element* l *of* $\mathbb{L}^{\Delta E}$ **do**

    if $l.t < n.l.t$ **then**

      $L_{<}(n.l).add(l)$;

    end

    if $l.t > n.l.t$ **then**

      $L_{>}(n.l).add(l)$;

    end

  end

  if $L^{N-1}_{\varsigma(k)}(k) \supset L_{<}(n.l)\ \&\ L^{N-1}_{\varsigma(k)}(k) \cap L_{>}(n.l) = \emptyset$ **then**

    $state.output = \alpha$;

  else

    $state.output = \varepsilon$;

  end

end

return *state*;

Procedure 4: CreateProcessGraph

ArgumentInOut: Reference Node $refNode$

ArgumentIn: Domain $\mathbb{L}^{\Delta E}$

Data: $\overline{L}_{\varsigma(k)}(k)$ and Successor Input Data Entity Set S

foreach *element* l *of* $\mathbb{L}^{\Delta E}$ **do**

  if $l \notin refNode.\varsigma(k)$ **then**

    $\overline{L}_{\varsigma(k)}(k).add(l)$;

  end

end

if $\overline{L}_{\varsigma(k)}(k).size > 0$ **then**

  $l_{min} = min(\overline{L}_{\varsigma(k)}(k))$;

  $S.add(l_{min})$;

  foreach *element* l *of set* $\overline{L}_{\varsigma(k)}(k)$ *except* l_{min} **do**

    if $(l.t < l_{min}.t)\&(l.t + l.P > l_{min}.t)$ **then**

      $S.add(l)$;

    end

    if $(l.t \geq l_{min}.t)\&(l_{min}.t + l_{min}.P > l.t)$ **then**

      $S.add(l)$;

    end

  end

end

foreach *element* l *of set* S **do**

  if $\forall node \in RefNode.children;\ l.ID \neq childNode.l.ID$ **then**

    $childNode = new\ Node(nodeID, refNode.\varsigma(k).l, l, null))$;

    $refNode.setChild(childNode)$;

  end

end

if $refNode.Children.size > 0$ **then**

  foreach *node element of* $refNode.Children$ **do**

    `CreateProcessGraph(`$node, \mathbb{L}^{\Delta E}$`);`

  end

end

Procedure 5: CreateStateSpace creates the state space

ArgumentInOut: StateSpace Sp

ArgumentIn: Reference State refState

ArgumentIn: Process Graph node n

foreach *element chNode of n.children* **do**

$State\ state = \texttt{CreateState}(chNode)$;

$State\ eqState =$

$\texttt{FindEquivalentStateInStateSpace}(Sp, state)$;

if *eqState* == *null* **then**

$refState.Post.add(new\ SuccState(chNode.l, state))$;

$S.add(state)$;

$stateId++$;

$\texttt{CreateStateSpace}(Sp, state, chNode)$;

else

$refState.Post.add(new\ SuccState(chNode.l, eqState))$;

end

end

Procedure 6: FindEquivalentStateInStateSpace creates the state space

```
ArgumentIn: StateSpace Sp
ArgumentIn: State state
Return: State eqState
eqState = null;
bool EqStateExists;
foreach element st of Sp do
    foreach element succState of state.Post do
        EqStateExists = false;
        foreach element succSt of st.Post do
            if succState.l.TaskID = succSt.l.TaskID then
                if succState.state.output = succSt.state.output then
                    EqStateExists = true;
                    break;
                end
            end
        end
        if EqStateExists == false then
            break;
        end
    end
    if EqStateExists == true then
        eqState = st;
        break;
    end
end
return eqState;
```

Procedure 7: CreateFiniteStateAutomaton

ArgumentIn: Initial State initState

ArgumentIn: StateSpace Sp

Return: Finite State Automaton $(\Sigma, \Lambda, Q, q_0, \delta, \lambda)$

$q_0 = initState.stateId$;

foreach *element* τ *of task set* $\mathcal{T}$ **do**

    $\Sigma.add(\tau.ID)$;

end

$\Lambda = \{\alpha, \varepsilon\}$;

foreach *State state element of StateSpace S* **do**

    $Q.add(state.stateId)$;

    $\lambda.add(new\ \lambda(state.stateId \rightarrow state.output)$;

    foreach *next state nextState element of State state* **do**

        $\delta.add(new\ \delta((state.stateId, SuccState.l.TaskID) \rightarrow$

        $SuccState.state.stateId))$;

    end

end

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Dursun, Mustafa
Nationality: Turkish (TC)
Date and Place of Birth: 09/03/1980, Ankara
Marital Status: Marital Status
Phone: 5363356991
Fax: -

EDUCATION

Degree	Institution	Year of Graduation
M.S.	METU EE	2006
B.S.	METU EE	2003
High School	Ankara Atatürk Anadolu Lisesi	1998

PROFESSIONAL EXPERIENCE

Year	Place	Enrollment
2010-	ASELSAN Inc. 1	Software Engineer
2010-2012	UIUC Computer Science Department	Visiting Scholar
2002-2010	ASELSAN Inc. 1	Software Engineer

PUBLICATIONS

M. Tekkalmaz, Ö. Kaya, M. Dursun, T. Sarı Tekkalmaz, A. Doğru. Görev Kritik ve Gömülü Sistemler için Hata Yönetimi Kılavuz Mimarisi: T5D. UYMK, 2008.

M. Tekkalmaz, E. Gürler, M. Dursun. Görev Kritik ve Gömülü Sistemler için T5D Uyumlu Bir Hata Yönetimi Altyapısı Tasarımı ve Gerçeklemesi. UYMS, 2009.

M. Dursun. Gömülü Yazılımlar için Model-Tabanlı Bir Bilesen Referans Modeli (UML-CM). UYMS, 2010.

M. Dursun. A Lightweight Automated Test Framework based on UML Testing Profile for Component Testing. OMG Workshop, 2010.

M. Dursun, P.L. Wu, C. Kim, L. Sha. A Bounded Asynchronous Design of Command and Control Cyber-Physical Systems for Complexity Reduction. Technical Report. 2012.

M. Dursun, Ö. Kızılay. Zaman Tetikli Almaç Planlayıcı Yazılım Bile¸seni Tasarımı. UYMS, 2014.

M.Dursun, S. Bilgen. Data Abstraction in Real Time Systems for Model Checking. Submitted to Jornal of Systems and Software, 2015.