Resource Estimation from Reliability Specifications in Embedded Cyber-Physical Systems

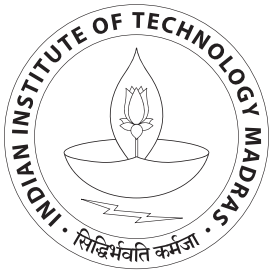
A Project Report

submitted by

GINJU V GEORGE

in partial fulfilment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY



DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.

May 2017

THESIS CERTIFICATE

This is to certify that the thesis entitled **Resource Estimation from Reliability Specifications in Embedded Cyber-Physical Systems**, submitted by **Ginju V George**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology**, is a bona fide record of the research work carried out by her under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Aritra Hazra
Assistant Professor
Dept. of Computer Science and Engineering
IIT-Madras, 600 036

Place: Chennai

Date:

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my guide, Dr. Aritra Hazra for his valuable guidance, encouragement and advice. His immense motivation helped me in making firm commitment towards my project work.

I would like to thank my co-guide Dr. Nitin Chandrachoodan and faculty advisor Dr. Shreepad Karmalkar who have patiently listened, evaluated, and guided us through out the program.

I would like to thank my family and friends who has supported me whole-heartedly.

ABSTRACT

KEYWORDS: Reliability, Formal Specifications, Assertions,Constraint satisfaction Problems, Resource Estimation, Safety-Critical Systems

Current usage of Formal specifications is limited to describing the correctness and timing related attributes of a system to aid its design and verification. Recent advancement paved ways to express other important design attributes such as power and reliability, using the formal specifications. Still there are other areas where the formal specifications can be used to abstract the attributes of the system. Previous research works [Hazra *et al.*, 2016] shows the extension of specification formalism to incorporate reliability behaviour of a design, using the temporal as well as spatial redundancy artifacts, into formal specifications. Since reliability is ensured by temporal or spatial replication of actions, the resource requirement of reliable systems render higher design and production cost. The design is developed from the requirements abstracted by the formal specifications, so it becomes imperative to study the resource requirements at an early stage leveraging the redundancy provisions from these reliability specifications. This project is an enabler for this and tries to study the relationship between the reliability requirements (specified upfront) and the number of processing resources (cores) required for the development of the safety critical system due to physical or temporal replication implied by spatial or temporal redundancy in the specified properties.

In this work, we address the question of finding optimum number of processing resources required to meet all the formal reliability specifications with sufficient reliability implied by introduction of appropriate redundancy artifacts. We model this problem as a*Constraint Satisfaction Problem (CSP)* so as to efficiently find the minimum possible number of resources. The timing restrictions binding the event sequences implied by the given set of reliability specifications in conjunction with associated correctness specifications as well as the resource restriction imposed on the design are modelled as a Constraint Satisfaction Problem (CSP) or a Constraint Optimization Problem (COP) and constraint solvers like Satisfiability Modulo Theories (SMT) solvers or Integer Linear Programming (ILP) solvers are invoked to estimate the number of processor-cores required to ensure specific reliability targets in the design. The practical application of this approach in estimating the number of design units required in safety critical designs with specific reliability targets is also addressed in this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

IITM	Indian Institute of Technology, Madras
CPS	Cyber-Physical system
CSP	Constraint Satisfaction Problem
COP	Constraint Optimization Problem
SMT	Satisfiability Modulo Theories
ILP	Integer Linear Programming
SVA	System Verilog Assertion
API	Application Programming Interface

NOTATIONS

ε_I	Sense-event
ε_O	Outcome-event
ε_A	Action-event

CHAPTER 1

INTRODUCTION

This chapter introduces the resource estimation problem formally. The motivation for selecting this work as well as the contributions made are highlighted.

This chapter introduces the formal estimation of resources in embedded CPS system from reliability specifications. The schematic representation of the proposed framework is explained highlighting the contributions made. The motivation for choosing the topic as well as the applicability of this approach in Safety-Critical systems are also briefly stated.

1.1 Overview and Proposed Framework

Formal specifications express the system at a higher level of abstraction, completely characterizing the system requirements using formal semantics. All the properties that should be satisfied by a system is congregated into formal specifications. A given set of formal specifications can be satisfied by designing the system in different ways. The variability of the designs satisfying the same formal specifications confirms the universal behaviour of the formal approach to specify system requirements. Formal specifications could be used for unambiguous description of these systems, and also to establish that their implementations exhibit the expected functionality. Early analysis of the system as well as prediction of expected behaviour of the system is thus possible using formal specifications. Timing analysis of the

system using formal specifications was widely studied in the past [Alur and Dill, 1994; Alur *et al.*, 1992; Dixit *et al.*, 2010, 2014; Maler, 2014]. Estimation of other artifacts like power and reliability were also attempted recently. In recent works, formal assessment of reliability requirements of Cyber-Physical System (CPS) is achieved [Hazra *et al.*, 2016].

In [Hazra *et al.*, 2016], the use of spatial (physical duplication of components) and temporal redundancies (repetitions in control actions such as re-execution of some control component) in embedded CPS framework has been studied. The formalism for reliability specifications proposed in this work provides methods to formally assess the adequacy of the redundancy provisions to meet desired reliability goals for the system functionalities at an early stage of design. The reliability metric defined by different time binded event sequences representing a functional reliability specification, extracted in-accordance with its corresponding functional correctness specification is calculated by the framework proposed in this paper. The action-event sequences ensuring specific reliability targets can thus be extracted using the algorithm proposed in this work [Hazra *et al.*, 2016] and these sequences forms the primary input to the Resource Estimation Algorithm proposed in this work.

Figure 1.1 is the schematic representation of the embedded CPS framework. It consist of three modules namely *Sensor*, *Controller* and *Actuator*. The *sensor* senses input scenarios, the *controller* performs the control decision based on the sensed input and the *actuator* delivers actuation signals. The sensed input scenarios or sensing inputs are termed as sensed-events (ε_I), consequent actions are termed as action-events (ε_A) and resulting outcome activities are termed as outcome-events (ε_O). The controller receives the sensed-events, interacts with the plant model

with appropriate actions to produce the desired outcome-events. The outcome of an action may take place after many control cycles and may also be durable over a period of time. The correctness specifications relate the sense-events (ε_I) with the outcome-events (ε_O) where as reliability specifications relate them to the action-events (ε_A). The probability that an outcome-event happens given the action-event responsible for the same occurs helps in calculating the respective reliabilities associated with different timing schemes of action-events (termed as action strategies). Specific reliability targets are imposed on these action strategies so as to ensure end-to-end system reliability in safety critical systems. The reliable set of strategies are to be evaluated to estimate the minimum number of processing resources required in the design.

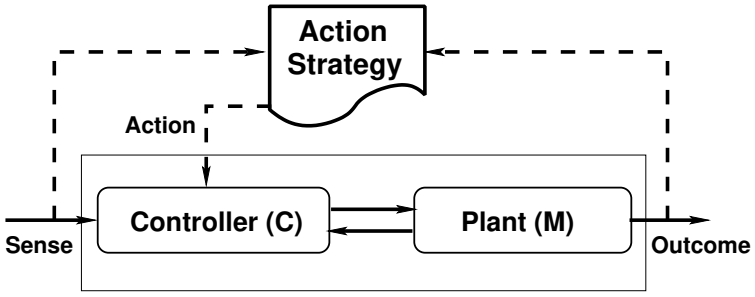


Figure 1.1: Schematic Representation of the Embedded CPS Framework

In this project, the estimation of required design resources is targeted by mathematical analysis of action sequences extracted from formal reliability specifications ensuring specific reliability targets. This project aims at developing an optimal algorithm to find the minimum number of resources required to meet the reliability requirements of a given set of formal reliability specifications and corresponding correctness specifications. Since simultaneous assertion of antecedent sequence comprising of sense-events (ε_I) in the specifications leads to finitely many possibilities in which the requirements can be met, the combinatorial method of searching through all possible combinations for satisfiability can explode in time. SMT

solvers which uses DPLL involving backtracking-based search procedures for deciding the satisfiability, has enhanced performance and is hence a better approach to solve the constraint optimization problems of this sort. A technique called Linear programming is also applicable when the problem is expressed in terms of conjunction of linear inequalities. Both these possibilities are explored with specific solvers chosen in each domain to estimate the number of processor cores required in the design so as to ensure specific reliability targets on formal reliability and correctness specifications.

Figure 1.2 gives the schematic representation of the proposed algorithm using SMT/ILP solvers. The approach used to solve this problem is by modeling the requirement specifications into timing constraints which relates the time of each outcome activity with respect to previous activity or absolute time. This problem can be modelled as a constraint satisfaction problem, where we have timing constraints extracted from the requirement specifications (SVA properties). The pre-conditions of all specifications are assumed to be asserted at time zero, so that the core requirement is most stringent. The task vs time relationships are extracted from the specifications. Task timing variables are created, declared and the relationship between them are specified with semantics of Z3 SMT solver/CPLEX ILP solver. The constraints are termed as assertion in SMT/ILP solvers. The resource bound is given as an optional input, using iterative bisection of this bound the optimum number of resources can be found iteratively by the SMT/ILP solver. This work aims in finding a possible scheduling of action-events responsible to cause desired outcome-events meeting specific reliability targets such that the resource requirement implied by the proposed scheduling scheme is the minimum possible one.

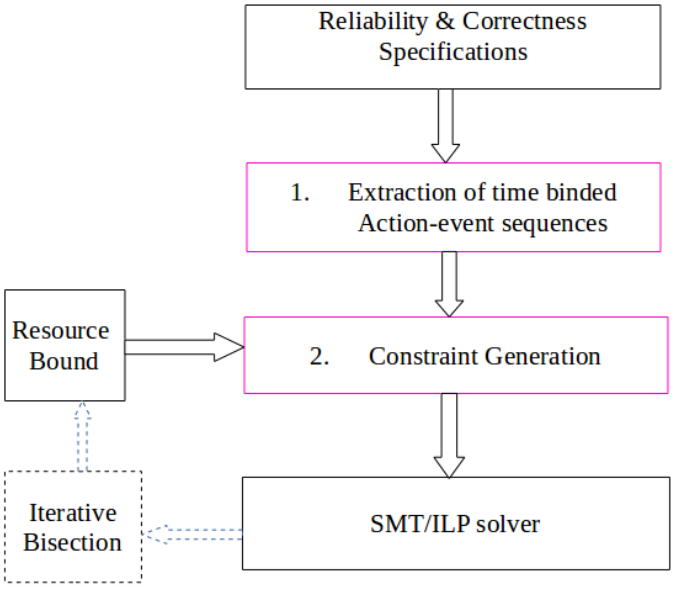


Figure 1.2: Schematic Representation of the Proposed Framework

1.2 Motivation

The motivation for doing this project is primarily the interest in formal specifications of CPS system and the challenges faced in ensuring reliability through redundancy. The requirement specifications implied by the System Verilog Assertions are to be exerted by the system components. Since redundancy can be included by physical and temporal replication of actions, multi-core CPS systems are needed for employing redundancy to ensure reliability. Using SVA assertions that could completely validate any system, to figure out the number of resources needed in the design, is of great advantage. The estimation of design resources from formal specifications impose great challenge, since the possible ways in which the requirements can be met increases exponentially for combinatorial search.

This project aims at formulating an algorithm to check the satisfiability of a given set of formal correctness and reliability specifications for a given number of

processor cores in a multi-core Cyber Physical System with inbuilt redundancy for reliability. Safety-critical embedded systems are designed with specific reliability targets, and design practices include the appropriate allocation of both spatial and temporal redundancies in the implementation to meet such requirements [Hazra *et al.*, 2016]. For spatial redundancy, the system components have to be physically replicated where as for temporal redundancy, the action-events are re-executes at definite time intervals so as to ensure higher probability of occurrence.

1.3 Contributions of this work

From Figure 1.2 which gives the schematic representation of the proposed algorithm using z3 SMT solver, the contributions of this projected are highlighted. The steps are summarized as below.

(Step 1) From the specification of the CPS, the action-event sequences are extracted such that all the antecedent sequences comprising of sense events are assumed to be asserted at time zero. The action-event sequences should comply with the correctness specifications. the formal assessment of the reliability is calculated for every such sequence and a target reliability is ensured so as to ensure specific reliability target of the system as a whole. This simultaneous assertion of sense event impose stringent resource requirements and hence ensures that our satisfiability check holds true in the extreme conditions.

(Step 2) The task timing variables are declared and the relationship between them

are specified as SMT/ILP solver assertions. The assertions are created with semantics of the chosen API of Z3 SMT solver or CPLEX ILP solver. The maximum value attainable by any of the variables is set as the simulation time over which the algorithm is run. The resource bound is specified to create the resource constraint. This bound is iteratively bisected and by solving the constraints (time as well as resource constraints) in Z3/CPLEX to predict the satisfiability of the imposed constraints, the optimum number of resources required is estimated.

The mathematical model for the resource estimation algorithm proposed in this work is explained with suitable examples in Chapter 3. The practical application of this approach in estimating the number of design units required in safety critical designs with specific reliability targets is Chapter 3 of this thesis. The implementation details and experimental results evaluating the scalability of the approach are discussed in Chapter 4.

1.4 Organization of the thesis

The background and the theoretical framework for the problem is discussed in Chapter 2. The modeling of the problem into Constraint Satisfaction problems requires proofs that existing scheduling and resource allocation problems does not fit into the scheme of allocation required by this problem. So a detailed study on bin packing, scheduling and task allocation problems are discussed in Chapter 2. The different solvers available for CSPs are also discussed in this chapter. The formal problem statement and proposed algorithm is explained in Chapter 3. The

experimental results and case study are presented in Chapter 4. The report is concluded with future scope in Chapter 5.

CHAPTER 2

Background and Related Work

This chapter introduces the theoretical background of the work done in this project

This chapter briefly introduces the formal reliability and correctness specification in CPS system in an attempt to present the available inputs of the problem presented and the required formats of the input. The brief on System Verilog Assertions is exempted assuming a previous knowledge of the same by the reader. The complete syntax of the SVA can be found in the SVA manual (System Verilog 3.1a Language Reference Manual). Since the approach presented in this paper is based on the Constraint Satisfaction Problems, similar problems like Bin Packing, Scheduling and Task Allocations are discussed in the same perspective so as to highlight the novelty of the proposed Resource Estimation Algorithm. There are wide varieties of SMT/ILP solvers available to solve Constraint Satisfaction Problems (CSPs), hence the reason for choosing Z3 SMT solver and CPLEX ILP Solver are highlighted in further sections. The architectural frameworks of both these solvers are also discussed

2.1 Formal Assessment of Reliability Specifications

Formal specification development is an integral part of the design process especially in integrated circuits and systems. The formal assessment of Reliability specifications [Hazra *et al.*, 2016] defines the formal use of reliability requirements

for estimating reliability from redundancy artifacts employed. Reliability can be ensured by using spatial [Kameyama and Higuchi, 1980; Kim and Shin, 1996; Lorczak *et al.*, 1989; Shin and Kim, 1994; Avizienis and Kelly, 1984; Liming and Avizienis, 1978]or temporal redundancy [Geist *et al.*, 1988; Krishna and Singh, 1993; Kim, 1999; Kim and Kim, 2004]. In [Hazra *et al.*, 2016] , the reliability specifications are used to formally determine what form of behavioral redundancy is needed at the component-level so that an end-to-end feature is implemented with a desired level of reliability.

Figure 1.1 is the schematic representation of the embedded CPS framework. The events that are responsible for sensing inputs are termed as sensed-events (ε_I), consequent actions are termed as action-events (ε_A) and resulting outcome activities are termed as outcome-events (ε_O). The controller receives the sensed-events, interacts with the plant model with appropriate actions to produce the desired outcome-events. The outcome of an action is time bounded with respect to time of occurrence of the sense-event. The outcome of an action may take place after many control cycles and may also be durable over a period of time.

The reliability of the CPS is governed by the fault-free execution from sensing to the outcome activity (Figure 1.1). This means that after sensing an input scenario, how reliably a action (actuations) can be performed after the controller decision being undertaken properly determines the reliability. In this model, the reliability of the outcome activities are attributed from the unreliable computational platform (inside the plant model) where the action tasks are scheduled, computed and applied. To meet the desired reliability of the outcome behavior, the fine-grain control incorporates various physical redundancy attributes inside the controller.

The activities of a CPS can be visualized in terms of sequence of events. The

early-stage specification narrates the outcome activity based on a sensed scenario which is to be met with a specified reliability value. Such specifications describe the correctness requirements for a CPS. In addition to this, the meta-level *action-strategy* specifies the supervisory control where the redundancies in actions are described for a sensed scenario. Such requirements are termed as the reliability specifications for CPS. The formal definition of these specifications is as given below.

From [Hazra *et al.*, 2016], the correctness requirement, ${P_S}^{\text{cor}}$, for a given subsystem, S, is formally expressed as

$$ {P_S}^{\text{cor}} : (\psi_I \rightarrow \#\#[a : b]\psi_O) $$

Here, the antecedent sequence, ψ_I, $\psi_I \epsilon \varepsilon_I$ denotes the pre-condition and ψ_O,$\psi_O \epsilon \varepsilon_O$ the consequent sequence denotes the outcome-activity.

The reliability requirement, ${P_S}^{\text{rel}}$ for a given subsystem, S, is formally expressed as

$$ {P_S}^{\text{rel}} : \psi_I \rightarrow \#\#[a : b]\psi_A) $$

Here, the antecedent sequence, ψ_I, denotes the pre-condition for any event and the consequent sequence, ψ_A,$\psi_I \epsilon \varepsilon_A$ denotes the disjunction-free action.

The spatial redundancy is formally represented using a property, ${P_S}^{\text{Srel}}$, as ${P_S}^{\text{Srel}} : (E_I \rightarrow \#\#[a : b](E_A[\sim n]))$, where E_I denote a sense-event and E_A denotes an action-event. This definition is used as an optional extension to System Verilog Assertions to represent spatial redundancy.

The temporal redundancy is formally represented using a property, ${P_S}^{\text{Trel}}$, as ${P_S}^{\text{Trel}} : (E_I \rightarrow \#\#[a : b](E_A[= n]))$ for nonconsecutive temporal repetitions and ${P_S}^{\text{Trel}} : (E_I \rightarrow \#\#[a : b](E_A[*n]))$ for consecutive temporal repetitions.

Given a set of properties ${P_S}^{\text{cor}}$, ${P_S}^{\text{rel}}$, ${P_S}^{\text{Srel}}$ and ${P_S}^{\text{Trel}}$ of the multi-core CPS, this project aims at checking the requirement satisfiability with the given number of cores (resource bound) ie whether the given number of cores can assure the satisfiability of the worst case requirements of the system when all the sense-events in the system are asserted simultaneously and finally converging to the optimum number of processing units required by iterative bisection of resource bound. An example showing the correctness and reliability specifications of the Adaptive Cruise Control System in Automobiles is given below.

Example 1. *Adaptive Cruise Control (ACC) supports features to – (a) maintain a minimum following interval to a lead vehicle in the same lane and (b) control the vehicle speed whenever any lead obstacle is present. Let us consider the functional requirements of ACC, which senses the proximity of any lead vehicle and a leading obstacle by the sense-events,* `lead_gap` *and* $\texttt{lead}_{\texttt{o}}\texttt{bs}$*, respectively. The required action-events issued by the ACC controller for reducing throttle by* 10% *and applying proper pressure in wheel-brakes are denoted as,* `act1` *and* `act2`*, respectively. The corresponding outcome-events are given as,* `thrt_adj` *and* `brk_adj`*. Now, consider the following two functional correctness requirements of ACC as follows:*

(a) `ACC_C1`*:* As soon as a lead obstacle is sensed, then within a total of 200 `ms`, the throttle is adjusted (reduced by 10%) followed by the application of wheel brakes after 50-100 `ms`. *Formally, this property is expressed as:*

$$\texttt{ACC\_C1} : \texttt{lead\_obs} \rightarrow \#\#[1:2]\ \texttt{thrt\_adj}\ \#\#[1:2]\ \texttt{brk\_adj}$$

(b) ACC_C2*:* Whenever a lead vehicle is sensed in a close proximity, then within a total of 300 ms, the throttle is adjusted (reduced by 10%) followed by the application of wheel brakes after 50-150 ms. *Formally, this property is expressed as:*

$$\texttt{ACC\_C2}: \texttt{lead\_gap} \rightarrow \#\#[1:3]\,\texttt{thrt\_adj}\,\#\#[1:3]\,\texttt{brk\_adj}$$

It is to be noted that a single time-unit delay is considered to be of 50 ms *in all the above mentioned functional specifications.*

Suppose, the desired reliability of the given correctness requirements be 0.95 *and* 0.98 *for* ACC_P1 *and* ACC_P2*, respectively. Here, the actions* act1 *and* act2 *are responsible for producing the outcome-events* thrt_adj *and* brk_adj*, respectively. Suppose, the outcome-events are unreliable and their reliability values are given as follows:*

$$\begin{aligned} R_{\texttt{thrt\_adj}} &= \mathit{Prob}(\texttt{thrt\_adj} \mid \texttt{act1}) = 0.8, \\ R_{\texttt{brk\_adj}} &= \mathit{Prob}(\texttt{brk\_adj} \mid \texttt{act2}) = 0.9. \end{aligned}$$

Since the outcomes are unreliable, the ACC must issue the action-events with appropriate redundancy in order to meet the desired reliability target, as given by the following reliability specifications.

(a) ACC_R1*:* As soon as a lead obstacle is sensed, then the throttle-reduction action-event is scheduled in two processors parallelly within $50 - 100$ ms, followed by the brake-apply action-event which is also scheduled in two processors parallelly within next $50 - 100$ ms. *Formally, this property is expressed as:*

$$\texttt{ACC\_R1}: \texttt{lead\_obs} \rightarrow \#\#[1:2]\,\texttt{act1}[\sim 2]\,\#\#[1:2]\,\texttt{act2}[\sim 2]$$

(b) ACC_R2*:* Whenever a lead vehicle is sensed in a close proximity, then the overall action of 10% throttle-reduction applied successively twice, followed by brake-application in the next time-unit is re-executed twice within an overall time

limit of 300 ms. *Formally, this property is expressed as:*

$$\texttt{ACC\_R2}: \texttt{lead\_gap} \rightarrow \texttt{\#\#[1:3]}\,(\texttt{act1[*2]}\,\texttt{\#\#1}\,\texttt{act2})\texttt{[=2]}$$

Now, given the reliability specifications, ACC_R1 *and* ACC_R2 *and assuming that the sense-events (*lead_obs *and* lead_gap*) are present in Cycle-0, the reliability for the correctness specifications,* ACC_P1 *and* ACC_P2 *are calculated as per the previous work [Hazra* et al.*, 2016] in Table 2.1 and Table 2.2, respectively, with respect to each action/control strategy*[1].

The highlighted rows of Table 2.1 and Table 2.2 indicate the action-strategies that meet the desired reliability requirements for both properties of ACC *subsystem. We call all these strategies meeting the reliability targets as the* admissible *action-strategies.* □

Table 2.1: Possible Options of Action-Events for ACC_R1

Possible Options	Action Events (Cycle-wise)				Computed Reliability
	Cycle-1	Cycle-2	Cycle-3	Cycle-4	
(1A)	act1 act1	act2 act2			0.9504
(1B)	act1 act1		act2 act2		0.9504
(1C)		act1 act1	act2 act2		0.9504
(1D)		act1 act1		act2 act2	0.9504

Table 2.2: Possible Options of Action-Events for ACC_R2

Possible Options	Action Events (Cycle-wise)						Computed Reliability
	Cycle-1	Cycle-2	Cycle-3	Cycle-4	Cycle-5	Cycle-6	
(2A)	act1	act1 act1	act2 act1	act2			0.9942
(2B)	act1	act1	act2 act1	act1	act2		0.9863
(2C)	act1	act1	act2	act1	act1	act2	0.8202
(2D)		act1	act1 act1	act2 act1	act2		0.9942
(2E)		act1	act1	act2 act1	act1	act2	0.9238
(2F)			act1	act1 act1	act2 act1	act2	0.8202

Example 1 also shows the formal assessment of the reliability specifications. The algorithms proposed in the work [Hazra *et al.*, 2016] is thus helpful in extracting

[1]The method for formal reliability assessment is presented in [Hazra *et al.*, 2016] in details.

the reliable action strategies from the reliability specifications. The reliability requirements along with the correctness specifications of a system can be assessed for synthesizing and implementing reliable designs. This could enable an early estimation of the required resources thereby achieving a reduction in the cost of the overall safety-critical systems.

2.2 Constraint Satisfaction Problems

Constraint-satisfaction problems are found in software and hardware verification , test-case generation, scheduling, planning, and graph problems which use logical formulas for describing states and transformations between these states. A constraint satisfaction problem (or CSP) is defined by a set of variables, $X_1, X_2, \ldots, X_n$ and a set of constraints, $C_1, C_2, \ldots, C_m$. Each variable X_i has a nonempty domain D_i of possible values. Each constraint C_i involves some subset of the variables and specifies the allowable combinations of values for that subset.

Any state of the problem is defined by an assignment of values to some or all of the variables, $\{X_i = v_i, X_j = v_j, \ldots\}$. An assignment that does not violate any constraints is called a consistent or legal assignment. A complete assignment is the one in which every variable is assigned a value. A solution to a CSP is a complete assignment that satisfies all the constraints. Some CSPs also require a solution that maximizes an objective function.

The most well-known constraint satisfaction problem is propositional satisfiability, or SAT. SAT also known as Propositional Satisfiability is defined over Boolean Expressions. A Boolean Logic formula is constructed using Boolean variables interconnected by Boolean operators AND, NOT and OR. A

Boolean logic formula is said to be satisfiable, if we could assign the variables with true/false values such that the Boolean formula turns out to get a true value for that set of variables. Thus SAT or Boolean Satisfiability problem aims in finding true/false assignments that could make any Boolean Function true. SAT solvers are employed to solve such problems efficiently. If we need to define satisfiability over richer constructs like integer or real arithmetic, strings and other data structures like arrays, sets etc. SMT or Satisfiability Modulo theories are to be used, since SAT is defined only for Boolean logic formulas.

Some well known constraint satisfaction problems like packing, task scheduling and task allocation are similar to this project. These problems are extensively studied and many algorithms were developed so far to handle them. The differences and similarities between these problems and the concerned problem are highlighted in this section.

2.2.1 Packing, Task Scheduling and Task Allocation Problems

The packing problem is a well-known constraint satisfaction problem and there are many versions for this problem. In one of the variants of this problem, we have a sequence of pieces $P_1, P_2, \ldots, P_n$ with size in $(0, 1]$. We have an infinite number of bins each with capacity 1. Each piece must be assigned to a bin. Further, the sum of the sizes of the pieces assigned to any bin may not exceed its capacity. The goal is to minimize the number of bins used [van Stee, 2015].

Packing problem is a resource allocation and minimization problem. A possible way to model the resource estimation problem as bin packing is to set each time slot as the bin number and we have to place the action-events in appropriate bins such that the height of the bins or the number of cores is not exceeded. The height

of all tasks is 1 here, and the user specified number of cores is the bin height. Due to definite relationship between the timing variables corresponding to the tasks, the placement of any task in such a bin, will automatically fix the other variables. So the bin packing algorithms where we have freedom of choice of the bins, are not fit for my problem.

Scheduling is the allocation of shared resources over time to competing requirements. In job shop scheduling we have a set of n jobs $\{J_1, J_2, \ldots, J_n\}$ to be scheduled on m Machines $\{M_1, M_2, \ldots, M_m\}$. Each job will be having n different tasks which need to be scheduled on one of the m Machines. The scheduling should be such that the make span or the total time required to complete all the n jobs is minimum. Each machine can process only one task at a time, and once a task is processed in a machine, it should not be interrupted [De Moura and Bjørner, 2011].

Task Scheduling problems are different from the resource estimation problem where we have a set of n properties each with some number of action-events to be executed in a particular sequence. The time to start any action-event is specified and the action-events inside a property also have definite timing relationships with any other action-event within the same property. The action-events can be executed on any of the m machines. Any action-event that falls in the same time slot can be shared if both are matching. The problem is to find out whether we can meet all the properties with the alloted number of resources. Since action-event sharing is not considered in task scheduling algorithms, they are not efficient in solving this problem.

In the task allocation problem, we have a task T consisting of m modules $\{t_1, t_2, \ldots, t_m\}$ and a distributed computer system with n processors $\{p_1, p_2, \ldots, p_n\}$.

Each of the modules comprising a task will execute on one of the processors and communicate with some other modules of the task. The problem is to allocate each of the m modules to one of the n processors such that an objective cost function is minimized subject to certain resource limitations and constraints imposed by the application or environment [Shatz *et al.*, 1992].

The task allocation problem is different from project problem in action-event sharing. Since property requirements can be met by action-event sharing, considerable reduction in resources can be achieved. In task allocation algorithms task sharing is not considered and so they are not suitable for the project discussed. The key to the resource estimation problem is sharing of action-events resulting in Resource Reduction. The timing of action-events bear a definite relationship with the time at which the sense-event happens which makes the resource estimation problem a constraint satisfaction problem. The proper allocation of time window can result in event sharing between reliability specifications resulting in reduction in the number of cores required. So satisfiability of all the specifications of a system can be achieved by a minimum number of resources. Example 2 shows how the sharing of action events can result in reduction in number of resources required.

2.3 Solvers for Constraint Satisfaction/Optimization Problems

SMT is also known as predicate satisfiability problem. The propositional satisfiability problem or SAT is a special case of the predicate satisfiability problem. SMT generalizes SAT by equality reasoning, arithmetic, fixed-size bit-vectors, arrays, quantifiers and other useful first-order theories. An SMT solver is a tool for de-

ciding the satisfiability of formulas in these theories. Z3 is an SMT solver from Microsoft Research. Other well known SMT solvers are VeriT, CVC4, MathSAT and Yices.

The earlier SMT solvers converted instances to Boolean SAT instances and this approach was known as the eager approach. It was complex due to difficulty in representation, but had the merit of using existing efficient SAT solvers for solving [Barrett *et al.*, 2009] Eventually the difficulties incurred led to the development of DPLL (Davis-Putman-Logemann-Loveland) style search with theory specific solvers that could solve predicates of concerned theories in conjunction. DPLL is a backtracking based search algorithm for deciding satisfiability of Boolean logic formulas in conjunctive normal form (AND of ORs). The Z3 SMT solver developed by Microsoft Research is used for this project.

An integer programming problem is a mathematical optimization/feasibility program with integer variables. In ILP the objective function and the constraints are linear. Integer programming is an NP-hard problem. A special case, 0-1 integer linear programming, in which the unknowns are binary is similar to the SAT problem. Similar to SMT, ILP is also used in solving different constraint optimization problems like capital budgeting and scheduling. SMT is more versatile in the sense that the type of variables is not limited to integers. Mixed Integer Linear Programming problems cover the floating point assertions based problems. ILP problem is very much similar to CSP, since ILP is a problem of finding a solution to a set of constraints (linear) that impose conditions that the variables (integers) must satisfy.

There are many open-source and commercial ILP solvers available. The widely

used ILP solvers are LP_SOLVE, CLP, SCIP, GUROBI, CPLEX and XPRESS. The CPLEX ILP solver is proved to be the best according to the studies [Meindl and Templ, 2012]. The IBM ILOG CPLEX Optimization Studio [IBM, 2012] is a commercial solver designed to tackle large scale mixed or integer linear problems. This software features several APIs and interfaces so that it is possible to connect the solver to different program languages and programs. A stand-alone executable is provided where an optimizer is also accessible through modeling systems.

2.3.1 Z3 Architecture

Z3 is a high-performance theorem prover being developed at Microsoft Research. Z3 combines several theory solvers into a combined framework and can be used to prove theorems and find counter-examples for non-theorems. Z3 integrates a modern DPLL based SAT solver, a core theory solver that handles inequalities and uninterpreted functions, satellite solvers for arithmetic, arrays, etc. and an E-matching abstract machine for quantifiers. Z3 is implemented in C++ and is modular in structure and hence new theories can be added without modifying the core [Moura and Bjørner, 2008].

The simplifier reduces the constraints using standard reduction rules. Compiler converts it into a data structure comprising of clauses and closure-nodes. The congruence closure core receives the truth assignments and equalities and inequalities thus created are propagated using E-graphs. Theory combination involves creation of implied equalities. The SAT solver effectively prunes all the boolean equalities. For quantifier reasoning z3 uses an approach that works over an E-graph to instantiate quantified variables. Z3 uses new algorithms that identify

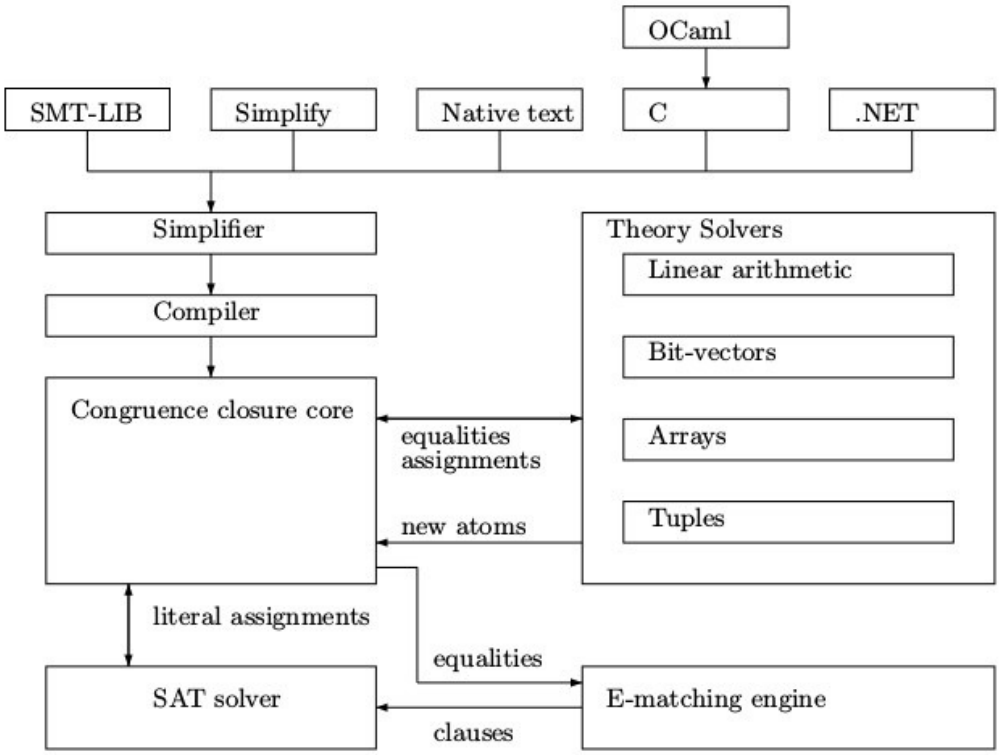


Figure 2.1: Z3 Architecture

matches on E-graphs. Theory Solvers in Z3 uses a linear arithmetic solver. Z3 has the ability to produce models as part of the output. Models assign values to the variables in the input.

2.3.2 CPLEX Architecture

The CPLEX Optimizer is implemented in the C programming language. It supports different types of mathematical optimization and offers interfaces other than C. The IBM ILOG CPLEX Optimizer solves integer programming problems using either primal or dual variants of the simplex method or the barrier interior point method. It can also solve convex and non-convex quadratic programming problems and convex quadratically constrained problems. The CPLEX Optimizer has a modeling layer called Concert that provides interfaces to the C++, C#, and Java languages.

There is a Python language interface based on the C interface. In addition to these interfaces, connectors to Microsoft Excel and MATLAB are also provided. The IBM ILOG CPLEX Optimization Studio comprises of the CPLEX Optimizer (mathematical programming), the IBM ILOG CPLEX Optimizer (constraint programming), an Optimization Programming Language (OPL) and a tightly integrated IDE (Integrated Development Environment).

An additional feature known as concurrent optimization is present in CPLEX making it faster. Concurrent optimization allows CPLEX to reach the solution in parallel since different solvers can be used simultaneously [Lima and Grossmann, 2011]. The different methods available to solve LP problems include Simplex method, primal Simplex method and Barrier method. The simplex method was the very first method designed for solving Linear Programs (LPs). This method is most widely used today and there are efficient implementations of the primal and dual simplex methods available in the Optimizer. The feasible region is the solution space of a set of constraints as per the nomenclature of Mathematical Programming. Each value possible for the objective function of an LP constitutes a hyper plane/a level set. A fundamental theory of simplex algorithm theory is that the optimal value of the LP objective function will occur when the level set meets the boundary of the feasible region. The optimal level set either intersects a single point of the feasible region, then the solution is unique. If optimal level set intersects a boundary set of the feasible region, we have an infinite set of solutions. The barrier method search for solutions not strictly on the boundary of the feasible region, so it could only reach an approximation of an optimal solution. The number of barrier iterations required to complete a problem depend on the required proximity to the optimal solution than the number of decision variables in the problem. Unlike the simplex method the barrier method completes any

problem in almost same number of iterations despite the size of the problem. With concurrent optimization different threads uses different methods to solve the problem. The barrier solver when applied to large problems may be faster than the simplex based solvers. The simplex may be superior if started from an advanced basis. So concurrent Optimization in CPLEX makes it faster due to the multiple threads executed parallelly using different solver techniques

2.3.3 Comparison of Constraint Solvers

The SMT solver is a constraint satisfaction solver, but not a constraint optimization solver where as ILP is a constraint optimization as well as a feasibility solver. Constraint satisfaction problems (CSPs) is the problem defined over a set of objects whose state must satisfy a number of constraints or limitations. the satisfiability of a set of constraints is targeted in CSPs. A Constraint Optimization Problem (COP) is the problem of finding a possible assignment to all variables that satisfies all hard constraints (constraints that must be satisfied) optimizing the global cost function. COP is a CSP with a objective function that is to be optimized. But every CSP can be viewed as a COP when not all constraints are satisfiable, that is finding an assignment that maximizes the number of satisfied constraints.

When modelling the Resource Estimation Problem as a CSP or a COP, there are two types of constraints to be considered. The first type is he timing constraints that define definite timing relationship between the action-events constituting the feasible action-event strategies meeting specific reliability targets. The second type is the resource constraint, this restraint is to be kept on all viable time value or the maximum depth of time over which the action strategy can exist assuming the sense-event corresponding to the reliability specifications happens at time zero.

Both the solvers, Z3 is a CSP solver and ILP is a COP solver will share the same timing and resource constraints. But since ILP requires an objective function, in addition to the constraints presented to Z3, a cost function is to be provided. The objective function is either a cost function or energy function which is to be minimized, or a reward function or utility function, which is to be maximized. Since Z3 could give the satisfiability we can check the satisfiability of the given set of reliability specifications with respect to any specific number of cores. In CPLEX, if the number of time cycles where the resource constraint is met is added up and the resulting sum is defined as the objective function, maximization of the same can result in the optimum solution of the problem resulting in a satisfiable solution. Hence we are finding an assignment that maximizes the number of satisfied resource constraints in ILP to model the problem as a CSP.

CHAPTER 3

Resource Estimation Algorithm

This chapter describes the mathematical framework of the proposed Resource Estimation Algorithm

In this chapter, the formal statement of the problem is derived and also the resource estimation framework is described with suitable examples. The inputs, assumptions and desired output for the problem are defined and a block diagram of the proposed framework in also included to illustrate the proposed framework. The specification parsing and constraint creation are expressed with mathematical notations and suitable examples are augmented for better understanding.

3.1 Formal Problem Statement

The problem of finding the optimal number of computing resources (processors) considering the simultaneous occurrences of a set of sense-events is formally defined on:

Inputs:

(i) A set of formal correctness and reliability properties of the system.

(ii) The reliability of each outcome-event with respect to the action-event corresponding to it.

(iii) The reliability target of reliability specifications with respect to each correctness specification.

(iv) A set of sense-events that may occur simultaneously.

(v) A set of admissible action-strategies (sequence of action-events) corresponding to every sense-event derived from the corresponding reliability properties of the system.

(vi) A pessimistic resource bound ($\mathcal{T}$).

Aim: Determine the appropriate choice of time binding for every action-strategy with respect to every sense-event so that the overall resource/processor requirement is minimized.

Assumptions:

(i) Same actions participating under two different action-strategies corresponding to different sense-events can be executed together in the same processor.

(ii) Execution of every action takes unit time (one cycle) in the processor.

(iii) Within one action-strategy, the participating actions appear as disjunction-free.

Table 3.1: Possible Executions of Action-Events and Required Resources for ACC Subsystem

Strategy Combinations	Possible Executions of Action-Events (Cycle-wise)					Resource Requirement
	Cycle-1	Cycle-2	Cycle-3	Cycle-4	Cycle-5	
1A+2A	⟨act1, act1⟩	⟨act1, act1, act2, act2⟩	⟨act1, act2⟩	⟨act2⟩		4
1A+2B	⟨act1, act1⟩	⟨act1, act2, act2⟩	⟨act1, act2⟩	⟨act1⟩	⟨act2⟩	3
1A+2D	⟨act1, act1⟩	⟨act1, act2, act2⟩	⟨act1, act1⟩	⟨act1, act2⟩	⟨act2⟩	3
1B+2A	⟨act1, act1⟩	⟨act1, act1⟩	⟨act1, act2, act2⟩	⟨act2⟩		3
1B+2B	⟨act1, act1⟩	⟨act1⟩	⟨act1, act2, act2⟩	⟨act1⟩	⟨act2⟩	3
1B+2D	⟨act1, act1⟩	⟨act1⟩	⟨act1, act1, act2, act2⟩	⟨act1, act2⟩	⟨act2⟩	4
1C+2A	⟨act1⟩	⟨act1, act1⟩	⟨act1, act2, act2⟩	⟨act2⟩		3
1C+2B	⟨act1⟩	⟨act1, act1⟩	⟨act1, act2, act2⟩	⟨act1⟩	⟨act2⟩	3
1C+2D		⟨act1, act1⟩	⟨act1, act1, act2, act2⟩	⟨act1, act2⟩	⟨act2⟩	4
1D+2A	⟨act1⟩	⟨act1, act1⟩	⟨act1, act2⟩	⟨act2, act2⟩		2
1D+2B	⟨act1⟩	⟨act1, act1⟩	⟨act1, act2⟩	⟨act1, act2, act2⟩	⟨act2⟩	3
1D+2D		⟨act1, act1⟩	⟨act1, act1⟩	⟨act1, act2, act2⟩	⟨act2⟩	3

Example 2. *Let us revisit Example 1 where we derive the admissible action-strategies for the two specifications of* ACC *subsystem in the highlighted rows of Table 2.1 and Table 2.2.*

Suppose, the sense-events, `lead_obs` *and* `lead_gap`*, occur simultaneously at Cycle-0. Then, Table 3.1 denotes possible execution options for the action-events combining the both the strategies. For example, if we try to ascertain Options* (1A) *with* (2A)*, then we need* two *processors in Cycle-1 to execute two parallel* `act1` *events among which one* `act1` *event of Option* (1A) *is shared/paired with the* `act1` *from Option* (2A).

Such task sharing helps us to reduce the number of processor requirements. We find that (1A + 2A) *combination requires* 4 *processors to execute the action strategies. Table 3.1 presents all the possible combinations and the required resources while executing each of these combinations of action strategies. It may be noted that, the minimum number of resources (*2*) is required when we perform the actions as per the Options* (1D) *and* (2A) *for both the properties. Hence, the next challenge is to bind the action-events with respect to Cycles so that the required resources are minimized.* □

3.2 Resource Estimation Framework

The required number of resources can be computed from a set of admissible action-strategies (corresponding to sense-events) derived from the reliability properties of a system, assuming that the corresponding set of sense-events occur simultaneously. The minimum count of execution units depends on the optimal allocation of the actions to appropriate processor cores in every execution time/cycle so as to maximize the sharing of action executions. The entire problem can be modeled as a Constraint Satisfaction Problem (CSP) which can be solved invoking an SMT or as a Constraint Optimization Problem (COP) which can be solved invoking an ILP solver.

Figure 3.1 illustrates the primary steps in this framework. The steps that are involved here are primarily categorized in two broad parts, namely, *[1] Parsing and Action Representation* and *[2] Constraint Generation*.

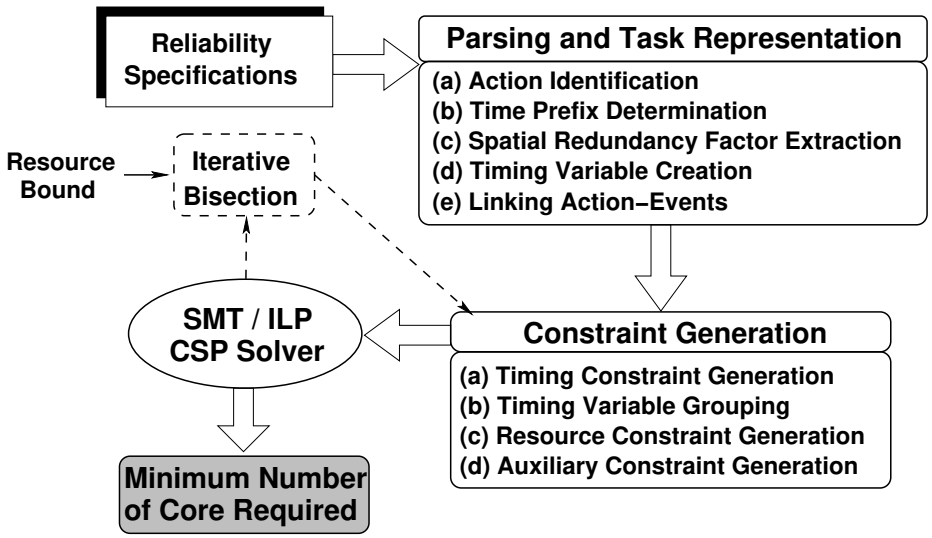


Figure 3.1: Resource Estimation Framework

3.2.1 Parsing and Action Representation

From the reliability properties and the derived action-strategies, the first step is to identify relevant information of the action-event occurrences, their timing and redundancy related information so that the required constraints can be derived later. We present various stages of parsing and action representation as follows.

(a) *Action Identification:* This step identifies the set of action-events from the reliability specifications. For reliability specifications having redundant actions, duplicates of the action-events are created.

(b) *Time Prefix Determination:* The time prefix of an action-event, extracted from the reliability specification, is typically given as either $\#\#t_0$ (fixed-time) or $\#\#[t_1 : t_2]$ (time-range) where $t_0, t_1, t_2 \in \mathbb{N}$, the set of all non-negative integers. The lower and upper bound of the time prefixes are represented using a doublet, $\langle t_0, t_0 \rangle$, for the fixed time prefix and a doublet, $\langle t_1, t_2 \rangle$, for representing the variable time prefix. The redundancies are treated as follows:

- In case of spatial redundancy (m-times), all replicated actions (from 2^{nd} to m^{th} occurrences) have $\langle 0, 0 \rangle$ as the time-prefix.

- In case of consecutive temporal redundancy (n-times), let the time prefix for the last action in the i^{th} execution is extracted as $\langle t_1,\ t_2 \rangle$. Then, the first action of $(i+1)^{th}$ $(i+1 \in [2, n])$ re-executed sequence starts at $(t_1 + 1)$. The time prefix of all other actions other than the first one is retained as such.
- In case of non-consecutive temporal redundancy (k-times), let the time prefix for the first action in the first execution is extracted as $\langle t_1,\ t_2 \rangle$. Then, the first action of i^{th} $(i \in [1, k])$ re-executed sequence can start earliest at $(t_1 + i)$ and latest by Δ[1] – thereby having $\langle (t_1 + i), \Delta \rangle$ as the time-prefix. The time prefixes of the other actions are extracted from the specification and remain same across all their re-executions.

(c) *Spatial Redundancy Factor Extraction:* Action-events with spatial redundancy of m are represented by associating each replication with a spatial redundancy factor from 1 to m. A default spatial redundancy factor of 1 is assigned to action-events having no spatial redundancy.

(d) *Timing Variable Creation:* For every action-event identified from the reliability properties, unique timing variables are created for each occurrence (replicated or re-executed) of that action, which will be used to derive the constraints relating to the time of execution of these action-events.

(e) *Linking Action-Events:* For every action-event appearing in the reliability specification, the timing-related constraints will be generated either from the absolute time of its occurrence or in relative to the timing of its preceding event. Hence, we need to map each timing variable (corresponding to an action-event) to the timing variable of its preceding action – thereby aiding to the generation of timing constraints for action-event sequences with respect to an action-strategy. The first action-event in the reliability specification is not linked to any other events, since it is depended only on the time of occurrence

[1] Δ is computed from the reliability specification considering the given reliability target of the corresponding correctness property. Intuitively, this upper bound in time comes from the fact that delaying the start (beyond Δ) in re-executing the same sequence reduces the number of possible satisfaction of the correctness property and hence the specified reliability remains unmet (Literature Hazra *et al.* [2016] provides more details).

of sense-event. Hence the timing variable corresponding to the first action-event has its linked variable as ϕ. For redundancies, the action-event links are done as follows:

- In case of spatial redundancy (m-times), all replicated actions (from 2^{nd} to m^{th} occurrences) are linked with the first occurrence of that action.
- In case of consecutive temporal redundancy (k-times), the first action in the i^{th} re-execution ($i \in [1, k]$) is linked with the last action in the $(i-1)^{th}$ re-execution.
- In case of non-consecutive temporal redundancy (k-times), the first action in the i^{th} re-execution ($i \in [1, k]$) is linked with the first action in the $(i-1)^{th}$ re-execution.

These stages of parsing and action representation are illustrated in details the the following Example 3.

Example 3. *Consider the reliability specifications,* `ACC_R1` *and* `ACC_R2`*, with their corresponding correctness specifications,* `ACC_C1` *and* `ACC_C2`*, as given in Example 1. Table 3.2 and Table 3.3 show the information extracted for these properties after* parsing and action representation *stage, assuming that the sense-events,* `lead_obs` *and* `lead_gap`*, for the properties happen together at time* $t = 0$. □

Table 3.2: Action-event Information Extracted for `ACC_R1`

Action	Time Prefix	Sp.-Red. Factor	Time Var.	Link Var.
`act1`	$\langle 1, 2 \rangle$	1	τ_{11}	ϕ
`act1`	$\langle 0, 0 \rangle$	2	τ_{12}	τ_{11}
`act2`	$\langle 1, 2 \rangle$	1	τ_{13}	τ_{12}
`act2`	$\langle 0, 0 \rangle$	2	τ_{14}	τ_{13}

3.2.2 Constraint Generation

Primarily, the set of generated constraints can be of the following two types – (i) timing-related constraints, and (ii) resource-related constraints. However, the

Table 3.3: Action-event Information Extracted for ACC_R2

Action	Time Prefix	Sp.-Red. Factor	Time Var.	Link Var.
act1	$\langle 1, 3 \rangle$	1	τ_{21}	ϕ
act1	$\langle 1, 1 \rangle$	1	τ_{22}	τ_{21}
act2	$\langle 1, 1 \rangle$	1	τ_{23}	τ_{22}
act1	$\langle 1, \Delta^{\dagger} \rangle$	1	τ_{24}	τ_{21}
act1	$\langle 1, 1 \rangle$	1	τ_{25}	τ_{24}
act2	$\langle 1, 1 \rangle$	1	τ_{26}	τ_{25}

$^{\dagger}\Delta$ is computed from ACC_R2 w.r.t. the reliability targets of ACC_C2

detailed stages, needed to derive these constraints to be used by the CSP/COP solver, are given below.

(a) *Timing Constraint Generation:* Given a reliability specification, $\texttt{Spec}_\texttt{i}$, let the timing variable corresponding to an action, $\texttt{act}_\texttt{j}$, be τ_{ix}. It has the time-prefix $\langle t_x^1, t_x^2 \rangle$ and is linked with another action, $\texttt{act}_\texttt{k}$, having the timing variable as τ_{iy} ($y < x$). Then, the timing constraints for τ_{ix} are generated as,

$$\tau_{ix} \geq \begin{cases} t_x^1, & \text{if } \tau_{iy} = \phi \\ \tau_{iy} + t_x^1, & \text{otherwise} \end{cases} \tag{3.1}$$

$$\tau_{ix} \leq \begin{cases} t_x^2, & \text{if } \tau_{iy} = \phi \\ \tau_{iy} + t_x^2, & \text{otherwise} \end{cases} \tag{3.2}$$

If $t_x^1 = t_x^2$, then the timing constraint is simplified as,

$$\tau_{ix} = \begin{cases} t_x^1, & \text{if } \tau_{iy} = \phi \\ \tau_{iy} + t_x^1, & \text{otherwise} \end{cases} \tag{3.3}$$

(b) *Timing Variable Grouping:* Given two reliability specifications, $\texttt{Spec}_\texttt{i}$ and $\texttt{Spec}_\texttt{j}$, having a common action, $\texttt{act}_\texttt{k}$, let the timing variable corresponding to that action, $\texttt{act}_\texttt{k}$, be τ_{ix} and τ_{jy}, respectively. Then, we can put τ_{ix} and τ_{jy} together forming one group, $\mathcal{G}_l$, i.e., $\tau_{ix}, \tau_{jy} \in \mathcal{G}_l$. It may be noted that, all the timing variables belonging to one group, say $\mathcal{G}_l$, are associated with the same action-event, say $\texttt{act}_\texttt{k}$, and are assigned with the same redundancy factor from

different properties. In case of spatial redundancy (say, action $\texttt{act}_\texttt{s}$ in $\texttt{Spec}_\texttt{i}$ is replicated m-times), the timing variables, $\tau_{is_2}, \tau_{is_3}, \ldots, \tau_{is_m}$, corresponding to every replicated action (2^{nd} to m^{th} occurrence) forms a singleton group, i.e., $\mathcal{G}_{l_2} = \{\tau_{is_2}\}, \mathcal{G}_{l_3} = \{\tau_{is_3}\}, \ldots, \mathcal{G}_{l_m} = \{\tau_{is_m}\}$ if there is no associated spatial redundancy for the same action ($\texttt{act}_\texttt{s}$ in other properties. If the same task say, action $\texttt{act}_\texttt{s}$ exists in another specification $\texttt{Spec}_\texttt{j}$ with spatial redundancy n-times, depending on the relative value of n with respect to m, the grouping varies as given below

- If $n \leq m$, the actions $\tau_{js_1}, \tau_{js_2}, \tau_{js_3}, \ldots, \tau_{js_n}$ are added to the groups G_{l_1}, $G_{l_2}, G_{l_3}, \ldots, G_{l_n}$ respectively.
- If $n \geq m$, the actions $\tau_{js_1}, \tau_{js_2}, \tau_{js_3}, \ldots, \tau_{js_m}$ is added to respective groups $G_{l_1}, G_{l_2}, G_{l_3}, \ldots, G_{l_m}$ and singleton groups $G_{l_m+1} = \{\tau_{js_{m+1}}\}, G_{l_m+2} = \{\tau_{js_{m+2}}\}$, $G_{l_m+3} = \{\tau_{js_{m+3}}\}, \ldots, G_{l_n} = \{\tau_{js_n}\}$are created.

(c) *Resource Constraint Generation:*

Pairing of timing variables into groups helps us to generate resource constraints such that, if $\tau_{ik_1}, \tau_{jk_2} \in \mathcal{G}_l$ then the corresponding actions (say, $\texttt{act}_{\texttt{k}_1}$ from $\texttt{Spec}_\texttt{i}$ and $\texttt{act}_{\texttt{k}_2}$ from $\texttt{Spec}_\texttt{j}$) are same, i.e. $\texttt{act}_{\texttt{k}_1} = \texttt{act}_{\texttt{k}_2}$. Therefore, these actions can be executed once whenever possible – resulting in a reduction in the execution resources (processors). Let an action-event, executing at cycle-t, is represented using the timing variable τ_{ij}; then the required resource due to the execution of only that action is given as,

$$\texttt{count}^t(\tau_{ij}) = \begin{cases} 1, & \text{if } \tau_{ij} = t \\ 0, & \text{otherwise} \end{cases}$$

The number of resources required at $\texttt{Cycle}$-t by the group of timing variables representing an action is denoted as,

$$\texttt{count}^t(\mathcal{G}_l) = \begin{cases} 1, & \text{if } \exists \tau_{ij} \in \mathcal{G}_l, \text{ such that } \tau_{ij} = t \\ 0, & \text{otherwise} \end{cases}$$

Suppose we are given with n specifications, $\texttt{Spec}_1$, $\texttt{Spec}_2$, ..., $\texttt{Spec}_\texttt{n}$. We choose all timing variables belonging to each $\texttt{Spec}_\texttt{i}$ ($i \in [1, n]$) from the group,

$$\mathcal{G}_l = \{\tau_{1x_1}, \tau_{1y_1}, \ldots, \tau_{2x_2}, \tau_{2y_2}, \ldots, \tau_{nx_n}, \tau_{ny_n}, \ldots\}$$

and create n sub-groups of $\mathcal{G}_l$ as,

$$\mathcal{S}^i_{\mathcal{G}_l} = \{\tau_{ix_i}, \tau_{iy_i}, \ldots\},\ \forall i \in [1, n]$$

Now, the required resource count for each of these sub-groups, $\mathcal{S}^i_{\mathcal{G}_l}$, is denoted as,

$$\texttt{count}^t(\mathcal{S}^i_{\mathcal{G}_l}) \ = \sum_{\forall \tau_{ij} \in \mathcal{S}^i_{\mathcal{G}_l}} \texttt{count}^t(\tau_{ij})$$

The sub-group count indicates the number of coincident action-events belonging to the same specification (implied by temporal redundancy). Hence, we indicate the total number of resources required at $\texttt{Cycle}$-t to execute the actions from these n sub-groups of $\mathcal{G}_l$ as,

$$\texttt{count}^t_{\texttt{SUB}}(\mathcal{G}_l) = \underset{1 \le i \le n}{\texttt{MAX}}[\texttt{count}^t(\mathcal{S}^i_{\mathcal{G}_l})]$$

Let all the timing variables corresponding to the action-event, $\texttt{act}_\texttt{k}$, appear in m groups, $\mathcal{G}_{l_1}, \mathcal{G}_{l_2}, \ldots, \mathcal{G}_{l_m}$. Then, we define,

$$\texttt{count}^t_{\texttt{SUP}}(\mathcal{G}_{l_j}) = \sum_{j=1}^{m} \texttt{count}^t(\mathcal{G}_{l_j}),\ \forall j \in [1, m]$$

A pertinent point to note here is that, for a group, $\mathcal{G}_l$, if there is no temporal redundancy involved (in the specification) for the representative action of that group, then we find, $[\texttt{count}^t_{\texttt{SUB}}(\mathcal{G}_l) - \texttt{count}^t_{\texttt{SUP}}(\mathcal{G}_l)] \le 0$ and the required resources at $\texttt{Cycle}$-t becomes, $\texttt{count}^t = \sum_l \texttt{count}^t(\mathcal{G}_l)$. Hence, the generated

resource constraint for Cycle-t is derived as follows:

$$\texttt{count}^t = \begin{cases} \sum\limits_{\forall l} \texttt{count}^t(\mathcal{G}_l) \textbf{ ; if } [\texttt{count}^t_{\texttt{SUB}}(\mathcal{G}_l) - \texttt{count}^t_{\texttt{SUP}}(\mathcal{G}_l)] \leq 0 \\ \sum\limits_{\forall l} [\texttt{count}^t(\mathcal{G}_l) + \{\texttt{count}^t_{\texttt{SUB}}(\mathcal{G}_l) - \texttt{count}^t_{\texttt{SUP}}(\mathcal{G}_l)\}] \textbf{ ; otherwise} \end{cases} \tag{3.4}$$

(d) *Auxiliary Constraint Addition:*

Two types of auxiliary (additional) constraints are derived to make the constraint generation process complete. These are discussed below.

- Admissibility Constraint: We have noticed in Section 2.1 that, among all the action-strategies that are generated from a reliability specification with respect to the given reliability target of its corresponding correctness property, there is a subset of action-strategies which are *admissible*. Let us assume that the last execution of an action-event can happen latest at Cycle-$\mathcal{T}$ such that all action-strategies remain admissible. Then, we need to add a constraint with respect to the timing variable, τ_{ij}, corresponding to the last executed action as,

$$\tau_{ij} \leq \mathcal{T}$$

- Resource Limit Constraint: We start with a pessimistic bound on the number of resources required and gradually refine that limit. For a given choice of maximum number of resources, Γ, these constraints are as follows:

$$\texttt{count}^t \leq \Gamma, \ \forall t \in [1, \mathcal{T}]$$

All these generated constraints can also be used in a constraint optimization tool/solver (like ILP solvers). There, we need to add the additional objective function providing the minimization or maximization requirement. Since we are minimizing the number of resources (processors) in this case, Γ is defined as an integer decision variable and the objective function will be:

$$\texttt{minimize}(\Gamma)$$

Example 4. *Let us revisit Table 3.2 and Table 3.3 generated in Example 3 and derive the required set of constraints.*

- *Timing Constraints: The timing constraints with respect to the property,* `ACC_R1`*, are derived as:*

$$\begin{aligned} \tau_{11} \geq 1 \quad ; \quad \tau_{11} \leq 2 \\ \tau_{13} \geq \tau_{12} + 1 \quad ; \quad \tau_{13} \leq \tau_{12} + 2 \\ \tau_{12} = \tau_{11} + 0 \quad ; \quad \tau_{14} = \tau_{13} + 0 \end{aligned}$$

  Similarly, the timing constraints with respect to the property, `ACC_R2`*, are derived as:*

$$\begin{aligned} \tau_{21} \geq 1 \quad ; \quad \tau_{21} \leq 2 \\ \tau_{22} = \tau_{21} + 1 \quad ; \quad \tau_{23} = \tau_{22} + 1 \\ \tau_{24} \geq \tau_{21} + 1 \quad ; \quad \tau_{24} \leq \tau_{21} + \Delta \\ \tau_{25} = \tau_{24} + 1 \quad ; \quad \tau_{26} = \tau_{25} + 1 \end{aligned}$$

  Now, $\Delta = 2$ *is extracted from the logical satisfaction of* `ACC_C2` *using* `ACC_R2` *subject to specific reliability targets.*

- *Timing Variable Groups: The timing variables for each action-events is grouped as follows:*

$$\begin{aligned} \mathcal{G}_1 = \{\tau_{11}, \tau_{21}, \tau_{22}, \tau_{24}, \tau_{25}\} \quad ; \quad \mathcal{G}_2 = \{\tau_{12}\} \\ \mathcal{G}_3 = \{\tau_{13}, \tau_{23}, \tau_{26}\} \quad ; \quad \mathcal{G}_4 = \{\tau_{14}\} \end{aligned}$$

  Here, all the timing variables present in the groups $\mathcal{G}_1$ *and* $\mathcal{G}_2$ *correspond to* `act1` *action-event and that are present in the groups* $\mathcal{G}_3$ *and* $\mathcal{G}_4$ *correspond to* `act2` *action-event.*

- *Resource Constraints: The resource constraints are derived using Table 3.4 for each*

`Cycle`-t. *Finally, we produce the following constraints (*$\forall t \in [1,5]$*):*

$$\texttt{count}^t = \begin{cases} \sum_{l=1}^{4} \texttt{count}^t(\mathcal{G}_l) \textbf{ ; if } [\texttt{count}^t_{\text{SUB}}(\mathcal{G}_l) - \texttt{count}^t_{\text{SUP}}(\mathcal{G}_l)] \leq 0 \\ \sum_{l=1}^{4} [\texttt{count}^t(\mathcal{G}_l) + \{\texttt{count}^t_{\text{SUB}}(\mathcal{G}_l) - \texttt{count}^t_{\text{SUP}}(\mathcal{G}_l)\}] \textbf{ ; otherwise} \end{cases}$$

Table 3.4: Resource Constraint Generation (for `Cycle`-t)

Groups	$\texttt{count}^t(\mathcal{G}_l)$	$\texttt{count}^t(\mathcal{S}^1_{\mathcal{G}_l})$	$\texttt{count}^t(\mathcal{S}^2_{\mathcal{G}_l})$	$\texttt{count}^t_{\text{SUP}}(\mathcal{G}_l)$
$\mathcal{G}_1$	$(\tau_{11} == t) \vee$ $(\tau_{21} == t) \vee$ $(\tau_{22} == t) \vee$ $(\tau_{24} == t) \vee$ $(\tau_{25} == t)$	$(\tau_{11} == t)$	$(\tau_{21} == t) +$ $(\tau_{22} == t) +$ $(\tau_{24} == t) +$ $(\tau_{25} == t)$	$\texttt{count}^t(\mathcal{G}_1) +$ $\texttt{count}^t(\mathcal{G}_2)$
$\mathcal{G}_2$	$(\tau_{12} == t)$	$(\tau_{12} == t)$	0	$\texttt{count}^t(\mathcal{G}_1) +$ $\texttt{count}^t(\mathcal{G}_2)$
$\mathcal{G}_3$	$(\tau_{13} == t) \vee$ $(\tau_{23} == t) \vee$ $(\tau_{26} == t)$	$(\tau_{13} == t)$	$(\tau_{23} == t) \vee$ $(\tau_{26} == t)$	$\texttt{count}^t(\mathcal{G}_3) +$ $\texttt{count}^t(\mathcal{G}_4)$
$\mathcal{G}_4$	$(\tau_{14} == t)$	$(\tau_{14} == t)$	0	$\texttt{count}^t(\mathcal{G}_3) +$ $\texttt{count}^t(\mathcal{G}_4)$

- *Auxiliary Constraints: The admissibility constraint, here, restricts the last action-event,* `act2`*, to happen latest by* 5^{th}*-cycle, i.e.* $\mathcal{T} = 5$ *(Refer to Table 3.2 and Table 3.3 of Example 3 for admissible action-strategies). Hence, we add,*

$$\tau_{26} \leq 5$$

Suppose, we set the resource limits as $\Gamma = 2$*, then the added constraints are as follows:*

$$\texttt{count}^1 \leq 2, \quad \texttt{count}^2 \leq 2, \quad \texttt{count}^3 \leq 2,$$
$$\texttt{count}^4 \leq 2, \qquad \textit{and} \qquad \texttt{count}^5 \leq 2$$

If we fed all the respective constraints in a SMT/ILP solver, then we find the following

satisfiable valuation of the variables:

$$\tau_{11} = 2, \quad \tau_{12} = 2, \quad \tau_{13} = 4, \quad \tau_{14} = 4,$$
$$\tau_{21} = 1, \quad \tau_{22} = 2, \quad \tau_{23} = 3,$$
$$\tau_{24} = 2, \quad \tau_{25} = 3, \quad \tau_{26} = 4.$$

This indicates that, both `act1` *and* `act2` *of* `ACC_R1` *is replicated twice in* `Cycle`-2 *and* `Cycle`-4*, respectively. Two consecutive* `act1` *followed by* `act2` *of* `ACC_R2` *is executed for the first time in* `Cycle`-1*,* `Cycle`-2 *and* `Cycle`-3*, respectively. Again, the re-execution of the same sequence happens when two consecutive* `act1` *followed by* `act2` *is executed for the second time in* `Cycle`-2*,* `Cycle`-3 *and* `Cycle`-4*, respectively. This result is also evident from Example 3, where the choice of Options* $(1D) + (2A)$ *produces this outcome as shown in Table 3.1.* □

The resource constraints, auxiliary constraints and the timing constraints are together fed to a SMT solver to check the satisfiability or the resource and timing constraints along with the objective function is fed to an ILP solver to minimize the objective function. Since we defined the number of resources required as the objective function to be minimized in ILP, the minimum number of resources can be found in a single run. For the SMT solver, if the constraints can be satisfied, then we conclude that the given resource limit, Γ, is sufficient for the admissible action strategies to be scheduled. However, an unsatisfiable outcome denotes that the resource limit needs to be increased further.

Now, to derive the optimal (minimum) number of required resources in SMT solver, we start from a pessimistic limit and proceed on iteratively bisecting the limit (in a similar manner as done in binary search technique) until we converge

into finding the minimum value of the resource limit. To illustrate the procedure, let us assume that we start with a given $\Gamma = 4$ in our example and derive the satisfiable valuations. Then, we bisect the limit and make $\Gamma = \lfloor \frac{(1+4)}{2} \rfloor = 2$ and still we can derive satisfiable valuations as illustrated in Example 4. Next, when we further bisect and make $\Gamma = \lfloor \frac{(1+2)}{2} \rfloor = 1$, then the constraints become unsatisfiable. Hence, we conclude that the minimum number of resources required to execute the admissible strategies is 2.

CHAPTER 4

Implementation Details and Experimental Results

This chapter presents the implementation details and the experimental results

In this chapter the implementation details and Experimental results are presented. The scalability of the implemented logic was thoroughly studied. The time response of the proposed algorithm showed exponential behaviour with increase in simulation time or sequential depth of the action-strategies extracted form reliability specifications.

4.1 Implementation Details

The C++ API of the Z3 and CPLEX solvers were used in the implementation. Lexical Analyzer Flex is used to analyze the input patterns and Bison is used as parser. Flex is an open source lexical analyzer for C++ Flex perform pattern-matching on text, it generates scanners/programs which recognize lexical patterns in text. Bison, is a parser generator. Bison reads a specification of a context-free language, warns about any parsing ambiguities and creates a parser (C, C++ or Java) which reads sequences of tokens and resolves whether the sequence correlates to the syntax specified by the grammar. Since the tokens are provided by flex we must provide the means to communicate between the parser and the lexer. The data extraction and the resource estimation code can be added in the parser itself. The C++ API of Z3 and CPLEX allows to instantiate appropriate objects to model the

problem as Constraint Satisfaction Problem and Constraint Optimization Problem respectively.

An outline of object model of Z3 C++ API consist of a context. The constraints are added to the solver defined over the context and solver status can be checked using a check function. The set of values for the variables on which the constraints are defined is referred to as a model as per the convention of the Z3. Whereas the C++ API of the CPLEX solver defines an environment and a model defined over the environment. As per CPLEX convention model defines the entirety of the constraints and the objective function. A CPLEX object is defined and the model is attached to the CPLEX object. This can be solved using a solve function, also the status of the solving can be checked. If there is a feasible or optimal solution, the values for the variables can be extracted from CPLEX object.

4.2 Experimental Results

In this section, the experimental results performed to check the scalability of this approach on an 8-Core Intel Xeon Ivy bridge $E5-2650v2$ series processor ($20M$ Cache, $2.60GHz$) with $4 \times 8GB$ RAM are presented. Figure 4.1 and 4.2 shows the 3D plots on the scalability experiments performed. We have used Z3 SMT solver Moura and Bjørner [2008] developed by Microsoft Corporation and CPLEX Optimizer (ILP solver) Meindl and Templ [2012] developed by IBM. The time response of the proposed approach subject to varying sequential depth and number of action-events were analyzed. The graphs depict exponential increase in execution time in determining the optimal resources as the sequential depth of the specifications increase. The time response with varying number of properties and

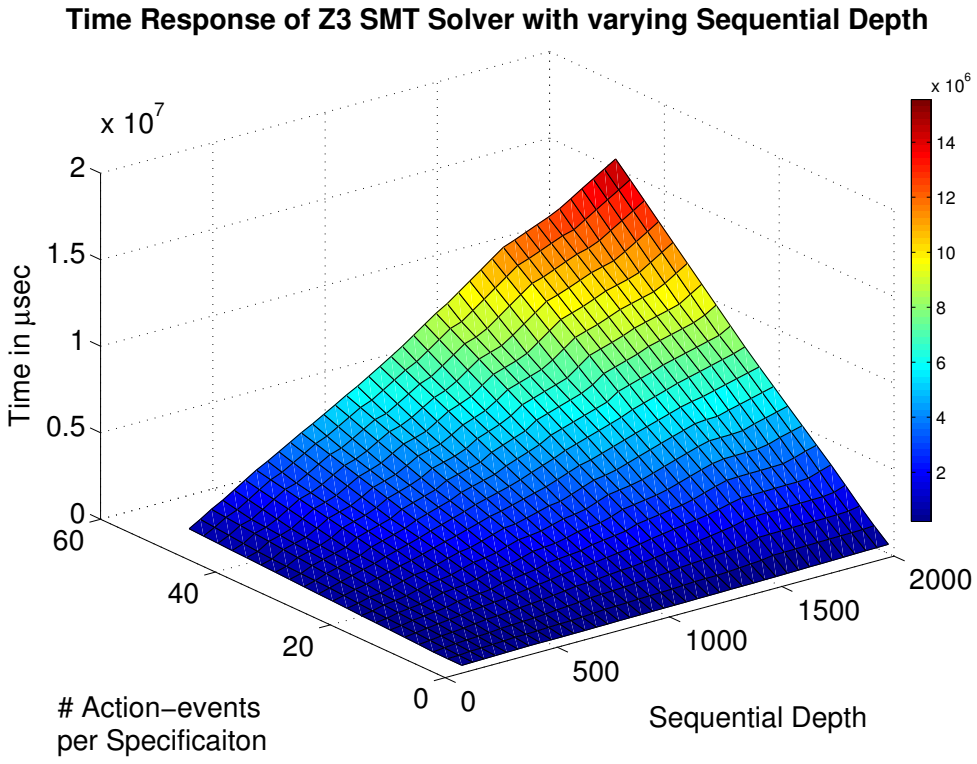
varying number of action-events per property are also studied. This response also has exponential behavior as indicated in Figures 4.1 and 4.2.

In Figure 4.1 the response of Z3 and CPLEX solvers to variations in sequential depth is shown. The Z-axis in all the graphs represents the time required to find the minimum number of resources in Microseconds. The X-axis indicates the sequential depth (varied upto 2000 time units) and the Y-axis indicates the Number of Action-events in Specification (varied upto 50). For smaller sequential depth, the variation in execution time with variation in number of action-events is rather small, but as the sequential depth increases, the execution time varies rapidly as number of action-events increase. The increases in sequential depth results in increase in number of resource constraints. The increase in number of action-events results in increase in the length and complexity of resource constraints as well. But for same number of action events, the number of timing constraints remain the same. The increase in number of resource constraints results in higher rate of increase in execution time than the increase in length of resource constraints.

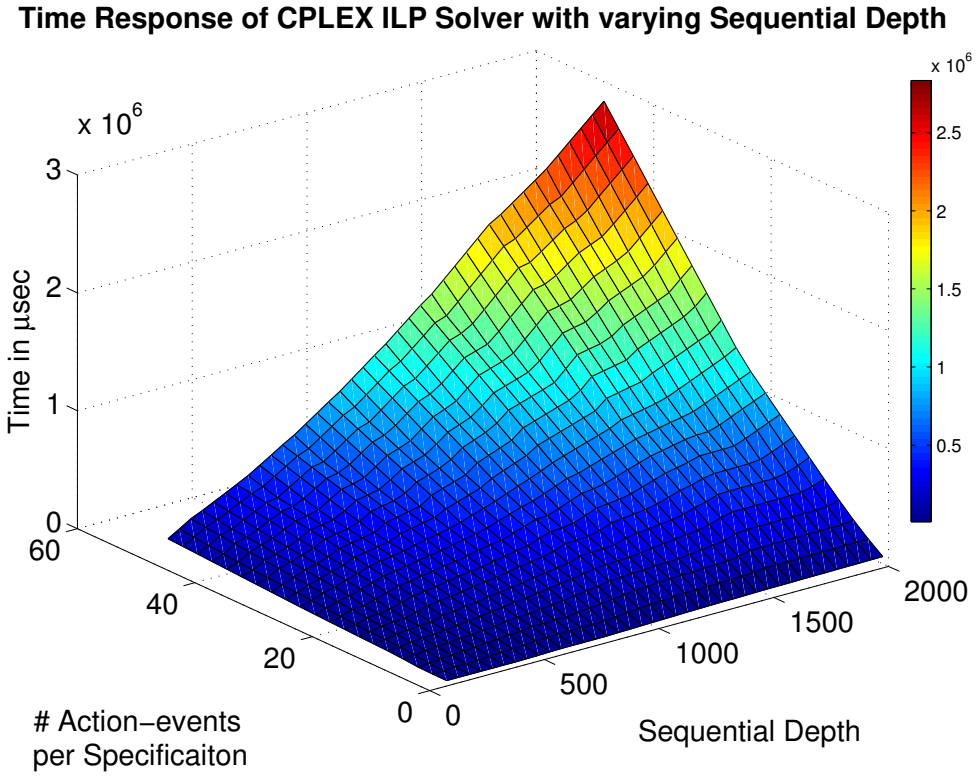
In Figure 4.2 the response of Z3 and CPLEX solvers to variations in number of specifications. The Z-axis in all the graphs represents the time required to find the minimum number of resources in Microseconds. The X-axis indicates the number of Specifications (varied from 0 to 200/500) and the Y-axis indicates the Number of Action-events per Specification (varied upto 80/100). For smaller number of specifications, the variation in execution time with variation in number of action-events is rather small, but as the number of specifications increases the execution time varies rapidly. Here the increase along the X and Y axis results in increase in number of timing constraints. This increase in number of timing constraints can be attributed to increase in number of action-events and number of specifications.

There will be variation in number of resource constraints, implied by both increase in number of action-events and increase in number of properties, since time of execution can vary in both these cases. Both X-axis and Y-axis has similar effects in increasing the length and number of resource constraints. Thus the behaviour is truly exponential in this case. The increase in timing constraints have lesser effect on the time of execution than the increase in the length/complexity of the resource constraints.

Comparing the performance of the Z3 solver with the CPLEX solver, the CPLEX is significantly faster in both scenarios. The primary reason behind this behaviour is the fact that CPLEX ILP solver is a COP solver. The search space of the ILP solver is limited to the optimal hyper plane which maximizes the objective function and hence the ILP solver will be faster in reaching the minimum number of resources. The calculation of the optimal plane offsets the difference slightly, still there is an appreciable variation in execution time between the two solvers.

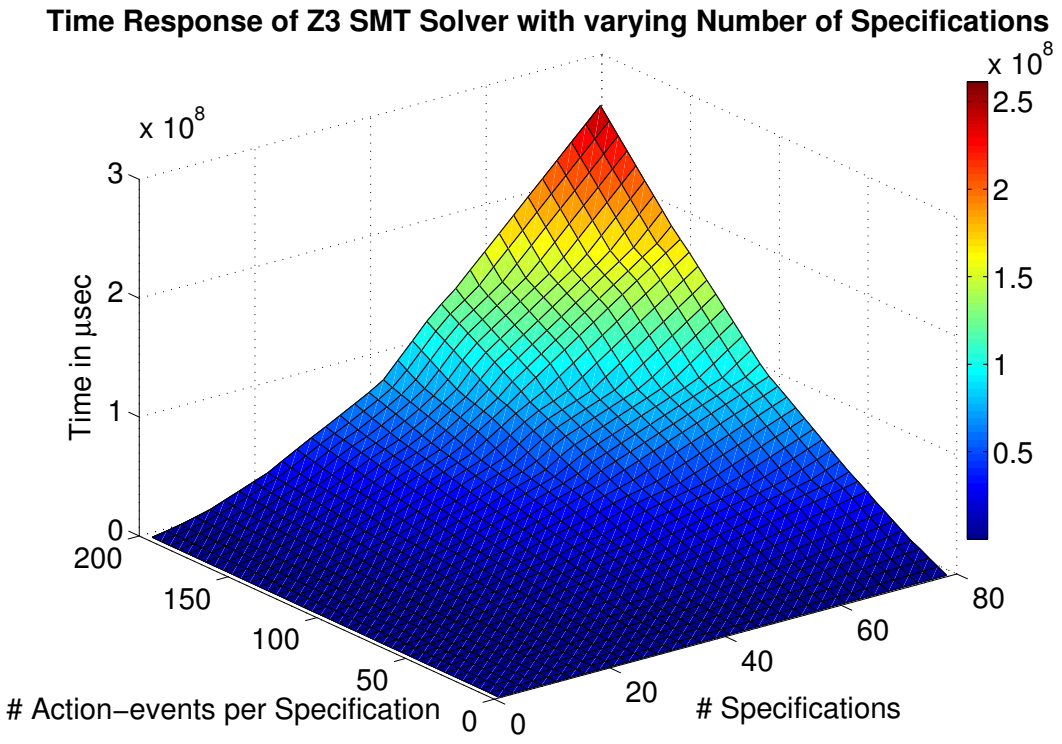


(a) Z3 SMT Solver Response

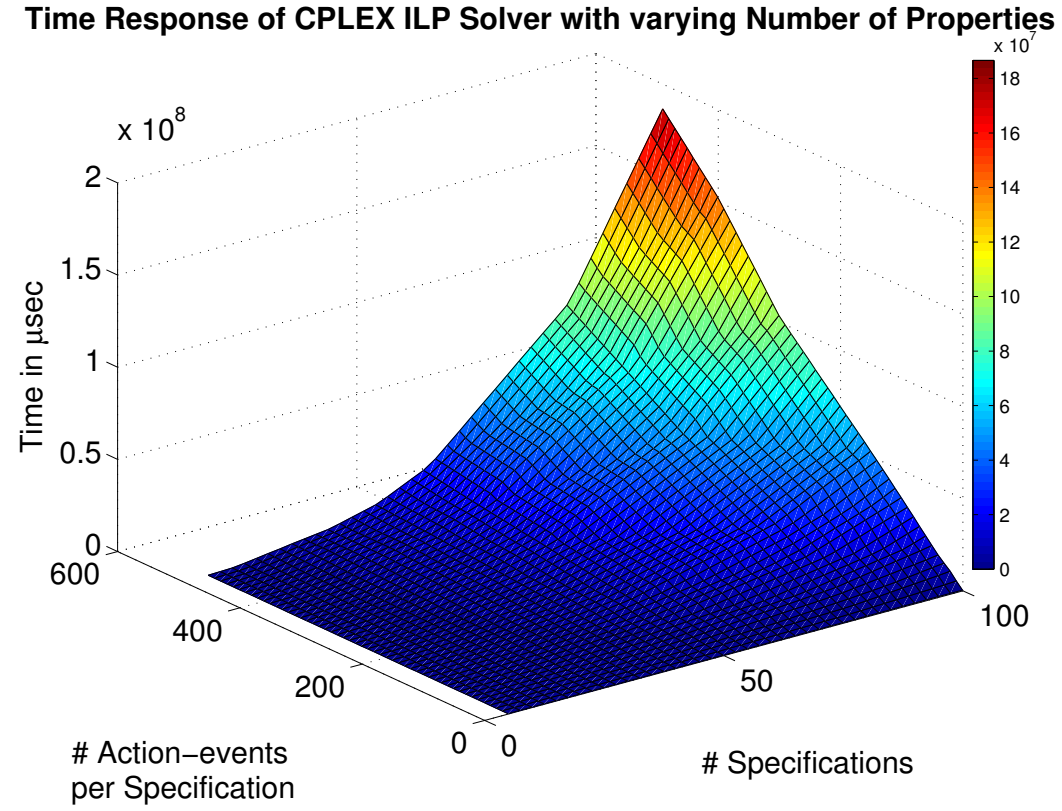


(b) CPLEX ILP Solver Response

Figure 4.1: 3D Graphs for Scalability Experiments on Sequential Depth



(a) Z3 SMT Solver Response



(b) CPLEX ILP Solver Response

Figure 4.2: 3D Graphs for Scalability Experiments on Number of Specifications

CHAPTER 5

Conclusion

In this work, an algorithm to find the optimum number of processing resources required to meet the formal reliability specifications and corresponding correctness specifications with specific reliability target was proposed. The problem is modelled as a *Constraint Satisfaction Problem (CSP)* or *a Constraint Optimization Problem (COP)* using an SMT solver or an ILP solver respectively. Scalability experiments were performed on the implemented logic and the results were analyzed. Early Estimation of required resources from formal assessment of reliability specifications could be helpful in the reduction of the cost of the overall safety-critical system. Only reliability specifications were considered in the proposed algorithm so far. The power constraints of the system can be added to further extend the scope of this work to check the satisfiability of the system requirements with reliability as well as power constraints.

REFERENCES

Alur, R. and **D. L. Dill** (1994). A Theory of Timed Automata. *Theoretical Computer Science*, **126**(2), 183–235.

Alur, R., **A. Itai**, **R. P. Kurshan**, and **M. Yannakakis**, Timing Verification by Successive Approximation. *In Computer-Aided Verification (CAV)*. 1992.

Avizienis, A. and **J. P. J. Kelly** (1984). Fault Tolerance by Design Diversity: Concepts and Experiments. *IEEE Computers*, **17**(8), 67–80.

Barrett, C. W., **R. Sebastiani**, **S. A. Seshia**, and **C. Tinelli** (2009). Satisfiability modulo theories. *Handbook of satisfiability*, **185**, 825–885.

De Moura, L. and **N. Bjørner** (2011). Satisfiability modulo theories: introduction and applications. *Communications of the ACM*, **54**(9), 69–77.

Dixit, M. G., **P. Dasgupta**, and **S. Ramesh**, Taming the Component Timing: A CBD Methodology for Real-time Embedded Systems. *In Design Automation and Test in Europe (DATE)*. 2010.

Dixit, M. G., **S. Ramesh**, and **P. Dasgupta** (2014). Time-budgeting: A Component Based Development Methodology for Real-time Embedded Systems. *Formal Aspects of Computing Journal*, **26**(3), 591–621.

Geist, R., **R. Raynolds**, and **J. Westall** (1988). Selection of a Checkpoint Interval in a Critical-Task Environment. *IEEE Transactions on Reliability*, **37**(4), 395–400.

Hazra, A., **P. Dasgupta**, and **P. P. Chakrabarti** (2016). Formal Assessment of Reliability Specifications in Embedded Cyber-Physical Systems. *Elsevier Journal of Applied Logic (JAL)*, **18**, 71–104.

Kameyama, M. and **T. Higuchi** (1980). Design of Dependent Failure-tolerant Microprocessor System using Triple Modular Redundancy. *IEEE Transactions on Reliability*, **C-29**(2), 202–206.

Kim, B. K., Reliability Analysis of Real-time Controllers with Dual-modular Temporal Redundancy. *In Sixth International Conference on Real-Time Computing Systems and Applications*. 1999.

Kim, H. and **K. G. Shin** (1996). Design and Analysis of an Optimal Instruction Retry Policy for TMR Controller Computers. *IEEE Transactions on Computers*, **45**(11), 1217–1225.

Kim, J. K. and **B. K. Kim** (2004). Probabilistic Schedulability Analysis of Harmonic Multi-Task Systems with Dual-Modular Temporal Redundancy. *Journal of Real-Time System*, **26**(2), 199–222.

Krishna, C. M. and **A. D. Singh** (1993). Reliability of Checkpointed Real-Time Systems using Time Redundancy. *IEEE Transactions on Reliability*, **42**(3), 427–435.

Lima, R. M. and **I. E. Grossmann** (2011). Computational advances in solving mixed integer linear programming problems.

Liming, C. and **A. Avizienis**, N-Version Programming: A Fault-Tolerance Approach to Reliability. *In Proceedings of Fault-Tolerant Computing Symposium*. 1978.

Lorczak, P. R., **A. K. Caglayan**, and **D. E. Eckhardt**, A Theoretical Investigation of Generalized Voters for Redundant Systems. *In Proceedings of Fault-Tolerant Computing Symposium*. 1989.

Maler, O. (2014). The Unmet Challenge of Times Systems. *Lecture Notes in Computer Science*, **8415**, 177–192.

Meindl, B. and **M. Templ** (2012). Analysis of commercial and free and open source solvers for linear optimization problems. *Eurostat and Statistics Netherlands within the project ESSnet on common tools and harmonised methodology for SDC in the ESS*.

Moura, L. D. and **N. Bjørner**, Z3: An Efficient SMT Solver. *In Proceedings of the Theory and Practice of Software (ETAPS) and 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. 2008.

Shatz, S. M., **J.-P. Wang**, and **M. Goto** (1992). Task allocation for maximizing reliability of distributed computer systems. *IEEE Transactions on Computers*, **41**(9), 1156–1168.

Shin, K. G. and **H. Kim** (1994). A Time Redundancy Approach to TMR Failures using Fault-state Likelihoods. *IEEE Transactions on Computers*, **43**(10), 1151–1162.

van Stee, R. (2015). Sigact news online algorithms column 26: Bin packing in multiple dimensions. *ACM SIGACT News*, **46**(2), 105–112.

List of Publications

Ongoing:

Ginju V. George and Aritra Hazra; Early-stage Resource Estimation from Reliability Specifications in Embedded Cyber-Physical Systems; Manuscript under preparation (to be submitted to an IEEE Journal), 2017.