Design, Specification, and Synthesis of Aircraft Electric Power Systems Control Logic

Thesis by

Huan Xu

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

2013

(Submitted May 31, 2013)

Acknowledgments

My sincerest gratitude to my advisor, Richard Murray, for his support and encouragement throughout this entire journey. Not only has he been a constant source of guidance, wisdom, and perspective, he has also allowed me the freedom to pursue my own interests. His leadership is something I hope to be able to emulate, and I am truly grateful to be a part of his group.

I would like to thank my thesis committee members Mani Chandy, Joel Burdick, and Gerard Holzmann. Throughout courses, seminars, and discussions, their insight has been invaluable to this, and past work. I am truly grateful for the opportunity to interact with such talented and respected individuals. It has been a privilege.

I had a wonderful group of colleagues on this project. First and foremost, thanks to Ufuk Topcu and Necmiye Ozay who have been fantastic mentors. I would also like to thank Rich Poisson from United Technologies Aerospace Systems for his expert domain knowledge of aircraft electric power systems. Thanks to everyone from the Murray Group, including Eric Wolff, as well as visiting students Robert Rogersten and Quentin Maillet. Thanks to Alberto Sangiovanni-Vincentelli, Pierluigi Nuzzo, John Finn, and Alexandre Donze from UC Berkeley. A special thanks to everyone in MCE and CMS for their support and friendship, including Cheryl Geer, Chris Silva, and Maria Koeper. Most importantly, Anissa Scott, without whom the group could not function.

Thanks to my friends who have been a great support network throughout the years: Nicholas Boechler, Jon and Shannon Mihaly, Justin Brown, Jacob Notbohm, Jason Rabinovitch, Nick Parziale, Victoria Nardelli, Andrea Leonard, and Lindsay Claiborn. To Emmy Ruiz, Stacy Berger, Marlon Marshall, Saskia Pallais, and Zach Hoover, thank you for teaching me to honor the pursuit of work, and to never stop believing. Last but not least, I would like to thank my family. My parents moved to this country over two decades ago in search of the American dream. This work is a testament to their dedication and sacrifice. Thank you, eternally.

Abstract

Cyber-physical systems integrate computation, networking, and physical processes. Substantial research challenges exist in the design and verification of such large-scale, distributed sensing, actuation, and control systems. Rapidly improving technology and recent advances in control theory, networked systems, and computer science give us the opportunity to drastically improve our approach to integrated flow of information and cooperative behavior. Current systems rely on text-based specifications and manual design. Using new technology advances, we can create easier, more efficient, and cheaper ways of developing these control systems. This thesis will focus on design considerations for system topologies, ways to formally and automatically specify requirements, and methods to synthesize reactive control protocols, all within the context of an aircraft electric power system as a representative application area.

This thesis consists of three complementary parts: synthesis, specification, and design. The first section focuses on the synthesis of central and distributed reactive controllers for an aircraft electric power system. This approach incorporates methodologies from computer science and control. The resulting controllers are correct by construction with respect to system requirements, which are formulated using the specification language of linear temporal logic (LTL). The second section addresses how to formally specify requirements and introduces a domain-specific language for electric power systems. A software tool automatically converts high-level requirements into LTL and synthesizes a controller.

The final sections focus on design space exploration. A design methodology is proposed that uses mixed-integer linear programming to obtain candidate topologies, which are then used to synthesize controllers. The discrete-time control logic is then verified in real-time by two methods: hardware and simulation. Finally, the problem of partial observability and dynamic state estimation is explored. Given a set placement of sensors on an electric power system, measurements from these sensors can be used in conjunction with control logic to infer the state of the system.

Contents

Chapter 1

Introduction

1.1 Motivation

Significant challenges arise in the design and verification of modern large-scale cyber-physical systems. Such systems, comprising a network of sensors, actuators, and physical systems involve the integration of computation, networking, and dynamical processes. Equipped with both computing and communication functionalities, design considerations need to include such aspects as (1) differing temporal scales in underlying dynamics, (2) information flow between agents, and (3) coordination of behavior between agents. Moreover, these systems need to rapidly react to changing environmental conditions or operational situations. Applications of cyber-physical systems appear in diverse areas, including autonomous aircraft and vehicles, traffic monitoring and control, “energy-smart" systems, manufacturing, and health care. The broad range of concerns amongst these applications include intended behavior, reliability, survivability, security, and constrained energy availability. While progress has been made in the design of large-scale cyber-physical systems within the past few years, there is still a lack of formal design methodologies and technologies capable of providing guarantees on behavior and execution.

Consider the application area of modern aircraft avionics (i.e., electronics applied to aviation). Correctly designing such systems has become increasingly difficult; recent significant delays in delivery of advanced aircraft, both civil and military, have been due to unforeseen interactions between a large number of strongly interdependent, heterogenous subsystems [17, 18, 22, 23]. Advances in electronics technology have made the transition from conventional to more-electric aircraft (MEA) architectures possible. More-electric aircraft architectures provide improvements in reliability and maintainability, as well as the potential to reduce aircraft weight and volume. The concept of electric aircraft is not new; though considered by military aircraft designers since the 1940’s, the idea was

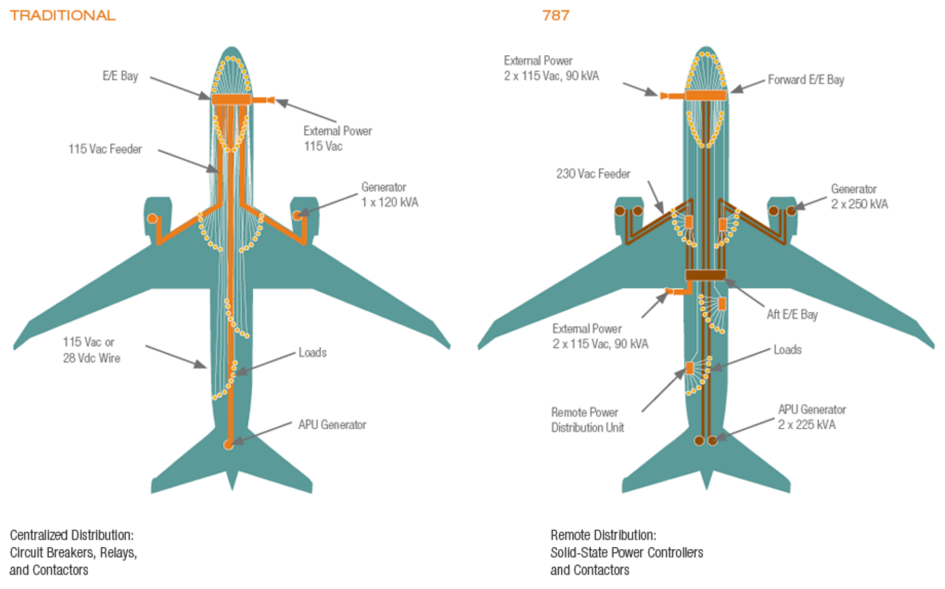


Figure 1.1: A comparison between the electric generation and distribution on a traditional aircraft and on the Boeing 787 (i.e., a more-electric aircraft) [85].

never implemented due to lack of electric power generation capabilities at that time as well as volume of required power conditioning equipment [21]. Conventional architectures utilize a combination of mechanical, hydraulic, electric, and pneumatic subsystems. The move towards more-electric aircraft increases efficiency by reducing power take-offs from the engines that would otherwise be needed to run hydraulic and pneumatic components. Moreover, use of electric systems provides opportunities for system-level performance optimization and decreases life-cycle costs. These architectures also introduce, however, new high-voltage electric networks and solutions for integrating additional subsystems.

Efforts have been made to re-use previously developed systems from conventional aircraft in more-electric aircraft [78], but additional high-voltage networks and electrically-powered components increase the system's complexity, and new designs for electric power systems need to behave according to certain properties or requirements determined by physical constraints or performance criteria. Figure 1.1 compares the difference between a traditional aircraft electric generation and distribution system to that of a more-electric aircraft system. Because safety of the aircraft is solely or mostly dependent on electric power, the electric power system on next-generation aircraft need to be highly reliable, and fault tolerant.

Analysis of all faults or errant behaviors in models is difficult due to the high complexity of systems and subsystem interactions. The process of verifying the correctness of a system with respect to specifications is expensive, both in terms of cost and time, which, as a result, has led to a greater emphasis on the use of formal methods to aid in safety and performance certification. The cost and time to allow for design changes near the end of the design cycle increases significantly. The growing need to rapidly and correctly design, implement, and commission large-scale systems requires new tool and techniques in modeling, analysis, design, and verification in order to provide a comprehensive and systematic solution to such problems.

1.2 Overview and Related Work

The overall objective of this thesis is to develop an initial framework for systematic design, specification, and synthesis of cyber-physical systems in order to provide a mathematical, formal guarantee of correctness of a system with respect to its requirements and desired behaviors. Within the context of an aircraft electric power system, of particular interest are systems in which low-level dynamics associated with hardware are integrated with high-level logics governing the overall behavior of the system.

The research presented in this thesis consists of three main components: design, specification, and synthesis. The following provides a quick overview of related work in each topic (presented in reverse order).

1.2.1 Formal Methods, Verification, and Synthesis

Formal methods have been utilized extensively in the computer science and control community to apply mathematical-based techniques to prove system correctness. Verification is a technique used to prove correctness of a control system with respect to a specific property. The most common forms of verification are theorem proving and model checking [20]. In model checking, the system is represented as a finite state machine and a specification, usually expressed in a temporal logic, is checked by efficiently searching the state space of the system. The benefit of model checking is that the process is fully automatic. Systems, however, are limited to a finite number of states. Because the search can be exhaustive, model checking faces a combinatorial blow-up of state space (otherwise known as state explosion). Theorem proving, is based on defining a set of axioms and inference rules to prove specific properties of the system. While the method is not limited to finite state systems, it usually requires skilled human interaction.

An alternative and complementary approach that extends the verification concept is to create a correct-by-design control program. Automatic design of control software provides a formal guarantee of system correctness, and can be used to reduce the time and cost of a system throughout its development cycle. Recently developed polynomial-time algorithms exist to construct finite-state automata from temporal logic specifications, from which automatic synthesis of digital design is possible. These designs are capable of satisfying a large class of properties (e.g., safety, response, liveness) in the presence of an adversarial environment [70].

Past work in the avionics field has focused on the analysis of aircraft performance and power optimization by using modeling libraries and simulations [91, 94]. Analysis of all faults or errant behaviors in models is difficult due to the high complexity of systems and subsystem interactions. Verifying the correctness of aircraft and other complex systems is thus difficult because of this intrinsic interleaving. While work has been done in this domain, verification of these systems requires a high level of time and domain expertise. As a result, this has led to a greater emphasis on the use of formal methods to aid in safety and performance certification. Of particular recent interest has been in the automatic synthesis of controllers for an electric power system designed so that the system satisfies all safety and reliability properties and requirements. The use of synthesis methods follows from their successful integration in verification of hardware and software systems in computer science, engineering, and robotics domains [33,38,45,48,73]. Previous work in [88] has applied formal synthesis of control protocols to enable dynamic reconfiguration of power in more-electric aircraft.

1.2.2 Specification and Requirements Capture

Current methods for requirements capture in systems design is performed in a non-rigorous and ad hoc manner. The Systems Modeling Language (SysML) is a general modeling language used in systems engineering application, and supports the specification, analysis, and design of a wide range of systems [32]. Developed in 2001 to customize the Unified Modeling Language (UML) [79], it is capable of modeling numerous applications in hardware, software, information processes, and facilities. Past work using SysML have included system architecture modeling, mobile phone production, as well as aircraft vehicle management systems [7,10,69,93]. While SysML semantics are expressive and flexible, allowing for a broad range of systems to be modeled, system requirements are still written in a text-based format that is ambiguous to analyze.

The use of formal mathematical languages (e.g., temporal logics) has garnered great interest due to their expressive power as well as their unambiguous meaning. An additional benefit is that methodologies from computer science and control incorporate temporal logics in design and verification. While the use of formal specification languages and correct-by construction synthesis methods is beneficial in the area of controller design, unfamiliarity of formal methods amongst engineers may provide a challenge to widespread implementation of formal methods.

Domain-specific languages have been proposed as a way to interface between industry engineers with domain knowledge with methods and tools used by computer scientists and software engineers. Domain-specific languages are languages adapted to a particular application or set of tasks. While general purpose languages (e.g., C or Java) may offer broader programming features, domain-specific languages (e.g., HTML or Verilog) provide more expressiveness and ease of use within a given domain [59]. Examples of languages used in the context of cyber-physical systems can be found in [4] and [13].

1.2.3 Design Space Exploration and State Estimation

Design space exploration examines design alternatives prior to implementation. Investigating design candidates is beneficial in many engineering tasks, including rapid prototyping, optimization, and system integration. The main challenge of design space exploration is the state space size that must be explored. For large system with millions (or billions) of possibilies, enumerating every choice can be prohibitive. Previous work has used SMT solvers to solve a set of global design constraints [43] and evolutionary algorithms for multi-objective design space exploration [87].

The process of design space exploration can benefit from the knowledge gained from state estimation, which can provide feedback in determining the set of candidates to analyze or explore. State estimation determines the current states of a system given some set of measured outputs. It has been widely used in detection and fault identification. Autonomous control systems rely on estimation in effective control of systems. The problem of estimating the state of a control system has been explored by several authors as a means for solving monitoring or surveillance problems. Estimation of electric power systems using optimization-based techniques is a well-established area [2, 16, 63]. In addition, a large body of work exists on diagnostics of electric power systems focusing on AC systems [24], as well as large vehicle systems. [53] examines the diagnostics for the international space station, [44] for an aircraft electric system, and [35] for a marine vehicle power system. For a DC system, [36] uses an optimization-based approach to estimate fault states. Past work in electric power system state estimation has focused on static, centralized estimation problems with continuous states.

1.3 Outline and Contributions

The scope of this thesis covers the framework for systematic design, specification, and synthesis of an aircraft electric power system. Chapter 2 provides background information on electric power systems. It also discusses various forms of temporal logics, including linear temporal logic, the language used mostly throughout this work, and finally introduces the formalisms for reactive and distributed synthesis. The main contribution of Chapter 3 is application of formal specifications in synthesizing centralized and distributed controllers for an aircraft electric power system. Additionally, timed specifications, i.e. requirements in which actions must occur within a given time bound, are formulated using linear temporal logic. Thus we present a timed version of synthesis for electric power system.

The automatic formalization of requirements into an electric power system domain-specific language is addressed in Chapter 4. The main contribution is an automatic specification generator tool *AES2gen* that receives as inputs a set of high-level primitives and automatically synthesizes controllers such as those described in Chapter 3. Requirements capture for various types of specifications is discussed, including ways to incorporate sequence-based specifications within an LTL framework.

Chapters 5-6 address the design aspect of electric power systems. Chapter 5 presents a design flow methodology for aircraft electric power systems. Multiple candidate topologies are generated using mixed-integer linear programming, for which we then automatically synthesize controllers that are then verified in simulation using the Breach toolbox [25]. The controllers are also implemented on a hardware testbed within a real-time framework. Chapter 6 explores the problem of sensor placement within an electric power system by proposing an algorithm for dynamic state estimation based on sensor measurements. Results from the algorithm are simulated on representative electric power system topologies. Finally, Chapter 7 concludes the work and discusses directions for research in the future.

Chapter 2

Background

2.1 Electric Power Systems

The standard electric power system for a passenger aircraft comprises a certain number of generators (e.g., one or two on the left and right sides of the aircraft) that serve as primary power sources. Generators supply power to a set of loads through dedicated AC buses. Typically, each AC bus delivers power to a DC bus through a transformer rectifier unit. Contactors are high-power switches that can control the flow of power by reconfiguring the topology of the electric power system and can establish connections between components. In the case of a generator or switch failure, an auxiliary power unit (APU) or battery may be used to power buses through a different reconfiguration of system components. Different reconfigurations of the system will change the open or closed status of contactors and thereby affect the power level of different buses or loads.

While standard topologies (i.e., structural arrangement of components) for electric power systems are already complex, next-generation aircraft are expected to become even more complicated, and thus more difficult to design. The move from pneumatic and hydraulic powered systems to electric powered ones increases the safety criticality of the electric power system. This elevated level of criticality can potentially be compensated for by increasing the number of paths between generators and buses that supply and deliver power to newly introduced loads. The increased number of overall components in the electric power systems raises the complexity of design as all possible configurations need to be considered. The number of configurations quickly goes beyond currently available verification and testing capabilities.

2.1.1 System Components

The electric power system schematic in Figure 2.1 includes a combination of generators, contactors, buses, and loads, transformers, and rectifier units. The following is a brief description of the components referenced in the primary power distribution single-line diagram [62].

Generators: AC generators can operate at either high voltages, which can connect to the high-voltage AC buses, or low voltages, which feed directly to the low-voltage buses.

Buses: High-voltage and low-voltage AC and DC buses deliver power to a number of sub-buses, loads, or power conversion equipment. Depending on the power availability and quality requirements on the loads, these buses can be classified as essential or non-essential. For example, essential buses supply loads that should always remain powered, such as the flight actuation subsystem, while non-essential buses have loads that may be shed in the case of a fault or failure, such as cabin lighting.

Contactors: Contactors are high-power electronic switches that connect the flow of power from sources to buses and loads. Depending on the power status of generators and buses, contactors can reconfigure, i.e., switch between open and closed. Contactors provide the actuation for reconfiguration of the topology of the electric power system, hence, changing the paths through which power is delivered from generators to loads depending on the contingencies.

Transformer Rectifier Units: Rectifier Units (RUs) convert three-phase AC power to DC power. Transformer Rectifier Units (XFMRs) combine a rectifier unit and a step-down transformer to additionally lower the voltage.

Batteries: Batteries are used as an electrical storage medium independent of primary generation sources. They provide short-term power during emergency conditions while alternative sources are being brought online.

RAM Air Turbine: The RAM Air Turbine (RAT) is a part of the emergency power system, and is a special purpose generator that becomes active with the loss of a number of main generators.

2.1.2 System Description

The following provides a brief description of the electric power system topology in Figure 2.1.

At the top of the diagram are six AC generators: two low-voltage, two high-voltage, and two APUs. Each engine connects to a high-voltage AC generator and a low voltage AC emergency generator. The high-voltage APU-mounted generators, hereafter referred to as auxiliary generators can also serve as backup power sources if a main generator fails.

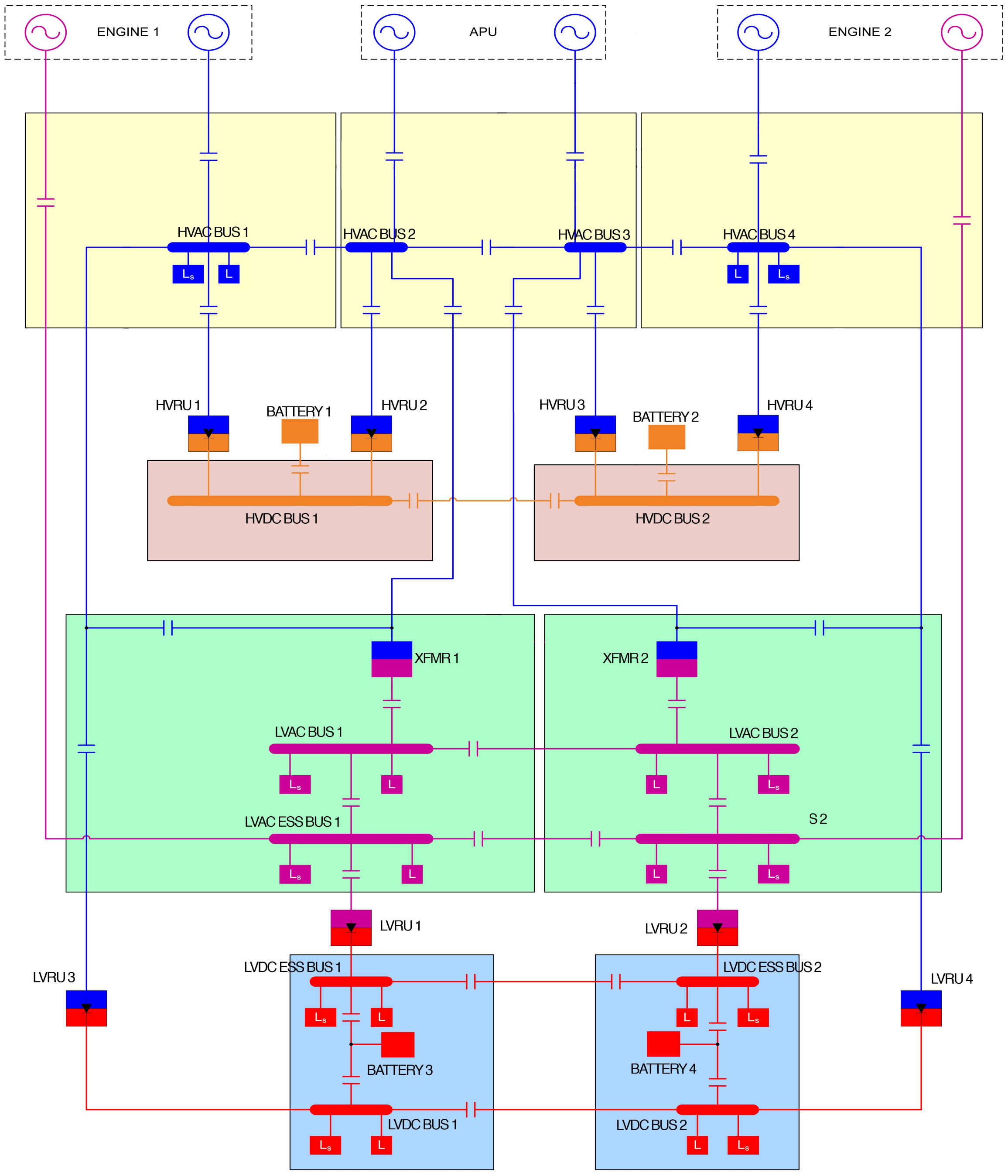


Figure 2.1: Single line diagram of an electric power system adapted from a Honeywell, Inc. patent [60]. Two high-voltage generators, two APUs, and two low-voltage generators serve as power sources for the aircraft. Depending on the configuration of contactors, power can be routed from sources to buses through the contactors, rectifier units, and transformers. Buses are connected to subsystem loads. Batteries can be used to provide emergency backup power to DC buses.

The three distinct panels directly below the generators contain the high-voltage AC distribution system. Each panel represents the physical separation of components within the aircraft. We denote components that can connect or disconnect from each other through the opening or closing of contactors as selectively connected (i.e., connected through a contactor). The four high-voltage AC buses can be selectively connected to all HVAC generators and auxiliary generators as well as each other by way of contactors (represented by $\dashv\vdash$).

Selectively connected to the four high-voltage AC buses are four high-voltage rectifier units (HVRUs) that transform AC power to DC power. HVRU 1 and HVRU 2 are directly connected to high-voltage DC Bus 1; HVRU 3 and HVRU 4 are directly connected high-voltage DC Bus 2. Each high-voltage DC bus also has a battery source which can also be selectively connected.

High-voltage AC Bus 2 and Bus 3 are also selectively connected to a set of transformers (labeled as XFMR on the single-line diagram) that convert high-voltage AC power to low-voltage AC power. The low-voltage AC system is depicted in the two panels in Figure 2.1 just below the high-voltage AC panels. These two transformers are connected to a set of four low-voltage AC buses. LVAC ESS Bus 1 and LVAC ESS Bus 2 are essential, meaning that they connect to loads which must always be powered. These essential buses are also selectively connected to the two low-voltage AC emergency generators in the case of a failure from the HVAC side.

The low-voltage AC essential buses are directly connected to low-voltage rectifier units (labeled as LVRU on the single-line diagram) converting low-voltage AC to low-voltage DC, as shown in the two bottom panels in Figure 2.1. There are four low-voltage DC buses, as well as two batteries which may also be selectively connected. Power can also be routed from the high-voltage AC buses through transformers to LVDC Main Bus 1 and LVDC Main Bus 2. Similar to the low-voltage AC case, low-voltage DC essential buses must remain powered at all times throughout the flight because of essential loads attached to the buses.

2.2 Temporal Logic

2.2.1 Linear Temporal Logic

Temporal Logic is an extension of propositional logic that incorporates notions of temporal ordering to reason about correctness over a sequence of states [9, 30, 39]. First introduced as a specification language by Pnueli [72] in the 1970s, it has since been demonstrated to be an appropriate formalism to reason about various kinds of systems, in particular in concurrent programs. The use of temporal logics to formally specify and verify behavioral properties has been seen in various applications, including embedded systems, robotics, and controls [73].

In reactive systems (i.e., systems which react to a dynamic, a priori unknown environment), correctness will depend not only on inputs and outputs of a computation, but on execution of the system as well. Temporal logic is a formalism well-suited for these types of problems in which the system must react to an adversary or environment. In this thesis we consider a version of temporal logic called linear temporal logic (LTL) that is suitable for describing certain properties of electric power systems. Other forms of temporal logic may be more or less expressive than LTL, depending on the desired behavior of the system. A brief overview of other languages is discussed in Section 2.2.2.

Before describing LTL, we begin by defining an atomic proposition, the basic building block of LTL. An atomic proposition is defined based on the variable structure of a system, as follows.

Definition 1: A system consists of a set V of variables. The domain of V, denoted $dom(V)$, is the set of valuations of V. A state of the system is an element $v \in domV$.

Definition 2: An *atomic proposition* is a statement on a valuation $v \in dom(V)$ with a unique truth value (*True* or *False*) for a given v. Let the valuation $v \in dom(V)$ be a state of the system, and p be an atomic proposition. Then $v \Vdash p$, read v satisfies p, if p is *True* at that state v. Otherwise, $v \nVdash p$.

In the electric power system domain, the set of variables includes, for instance, generator and contactor statuses. Valuations of these variables include the health values of generators. An atomic proposition could state that each generator in the system be healthy.

Alongside atomic propositions, LTL also includes Boolean connectors like negation ($\neg$), disjunction ($\vee$), conjunction ($\wedge$), material implication ($\rightarrow$), and two basic temporal modalities *next* ($\bigcirc$) and *until* ($\mathcal{U}$). By combining these operators and propositions, it is possible to specify a wide range of requirements on the desired behavior of a system and environment assumptions. Given a set π of atomic propositions, an LTL formula is defined inductively as follows:

- any atomic proposition $p \in \pi$ is an LTL formula;
- given LTL formulas φ and ψ over π, $\neg\varphi$, $\varphi \vee \psi$, $\bigcirc\varphi$ and $\varphi \,\mathcal{U}\, \psi$ are also LTL formulas.

Given a set of valuations and a set π of atomic propositions over valuations $v \in dom(V)$, LTL formulas over π are interpreted over infinite sequences of states. For example, the formula $\bigcirc\varphi$ holds for a sequence of states at the current step of the sequence if φ is true in the next step. The formula

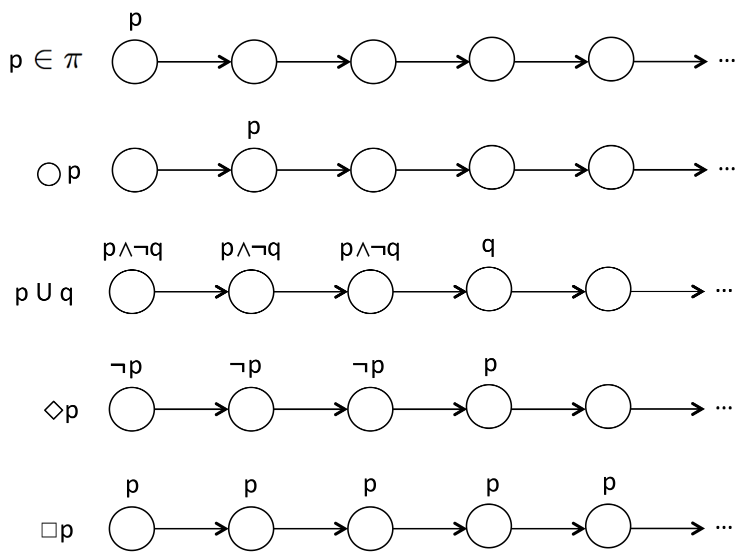


Figure 2.2: Semantics of LTL temporal modalities. Propositions are reasoned about over entire sequences of states. In the first sequence, atomic proposition p is true for the initial state, denoted by a p above the first state in the sequence. In the second sequence, p holds in the second state, or next step. In the third sequence, p is true until the step when q becomes true. In the fourth sequence, p is eventually true at some step. In the last sequence, p is true for every step. A state without a label contains an arbitrary set of propositions.

$\varphi_1 \, \mathcal{U} \, \varphi_2$ holds at the current step if at some future step φ_2 holds and φ_1 holds at all steps until that future step.

Formulas involving other operators can be derived from these basic ones. The until operator can be used to derive two further temporal modalities that are used commonly in LTL, namely *eventually* ($\Diamond$) and *always* ($\Box$). The formula $\Diamond\varphi$ states that φ will be true at some point in the future, while $\Box\varphi$ is satisfied if and only if φ is true for all points. Figure 2.2 illustrates some temporal modalities that can be expressed in LTL. On the left-hand side are LTL formulas over propositions p and q, while on the right are sequences of states.

More formally, the semantics of LTL is given as follows. Let $\sigma = v_0v_1v_2\ldots$ be an infinite sequence of valuations of variables in V, and φ and ψ be LTL formulas. We say that φ holds at position $i \geq 0$ of σ, written $v_i \models \varphi$, if and only if φ holds for the remainder of the execution σ starting at position i. Then, the satisfaction of φ by σ is inductively defined as:

- for atomic proposition p, $v_i \models p$ if and only if $v_i \Vdash p$;
- $v_i \models \neg\varphi$ if and only if $v_i \not\models \varphi$;
- $v_i \models \varphi \vee \psi$ if and only if $v_i \models \varphi$ or $v_i \models \psi$;

- $v_i \models \bigcirc\varphi$ if and only if $v_{i+1} \models \varphi$; and
- $v_i \models \varphi \,\mathcal{U}\, \psi$ if and only if $\exists\, k \geq i$ such that $v_k \models \psi$ and $v_j \models \varphi$ for all j, $i \leq k < j$.

Based on this definition, $\bigcirc\varphi$ holds at position i of σ if and only if φ holds at the next state v_{i+1}, $\square\varphi$ holds at position i if and only if φ holds at every position in σ starting at position i, and $\diamond\varphi$ holds at position i if and only if φ holds at some position $j \geq i$ in σ.

Let Σ be the collection of all sequences of valuations of V. Then, a system composed of the variables V is said to satisfy φ if $\sigma \models \varphi$ for all $\sigma \in \Sigma$. A set of models Σ satisfies φ, denoted by $\Sigma \models \varphi$, if every model in Σ satisfies φ.

Examples of LTL formulas: Given a propositional formula, common and widely used properties can be defined in terms of their corresponding LTL formulas as follows.

Safety: Safety formulas assert that a state or sequence of states will not be reached. In particular, we use a subclass of safety formula referred to as invariants throughout this paper. Invariant formula assert that a property will remain true throughout the entire execution σ for all executions $\sigma \in \Sigma$. Safety properties ensure that nothing bad will happen. A safety specification for the electric power system could take the form $\square(\neg bus\_i\_unpowered)$ where i is the bus index.

Progress: Progress formula guarantee that a property holds infinitely often in an execution σ. This property ensures that the system will make progress. For example, always eventually ensure that Bus 1 is powered can be written as: $\square \diamond gen\_i\_powered$.

Response: A response formula states that at some point in the execution following a state where a property is true, there exists a point where a second property is true. Response properties can be used to describe how systems need to react to changes in environment or operating conditions. A response property can be used to describe how the system should react to a generator failure. If a generator fails, then at some point a corresponding contactor should open: $\square((gen\_j\_not\_healthy) \rightarrow \diamond(contactor\_k\_open))$ where j, k represent indices for generators and contactors, respectively.

Remark 1 *Properties typically studied in the control and hybrid system domains are safety and stability. LTL can express a more general class of properties. Typical specifications seen with electric power systems or more-electric aircraft in general involve safety (avoid unsafe configurations) and response (if a failure occurs, then reconfigure). Progress properties are not used since systems do not typically have a "goal" state that needs to be reached, but instead consist of a set of safe operational states. We use a combination of response and modified progress formulas in order to capture timing properties.*

2.2.2 Other Temporal Logics

LTL is one form of temporal logic capable of expressing desired system behaviors. LTL is called linear due to the qualitative notion of time as path-based. Each moment of time has a unique possible successor state. LTL can state properties over all possible computations beginning from a state. It cannot, however, easily reason about *some* of the possible computations. To address such difficulties, Computation Tree Logic (CTL), was introduced by Clarke and Emerson [19]. CTL is a branching temporal logic, with a branching notion of time. At each moment there may be several different futures.

Real-time variants of temporal logic aim to express properties of systems with real-time specifications (e.g. p must be true within t seconds). While LTL and CTL can reason about ordering of events, they cannot specify the exact time an event must occur. Metric Temporal Logic (MTL) [46] and timed CTL (TCTL) [6] are thus extensions of LTL and CTL, respectively, with additional clocks and clock constraints. for LTL and CTL, there exists an implicit time bound on operators always, eventually, and until such that

$$\square \doteq \square_{[0,\infty)},$$

$$\diamond \doteq \diamond_{[0,\infty)},$$

$$\mathcal{U} \doteq \mathcal{U}_{[0,\infty)}.$$

MTL and TCTL modify the time interval from $(0,\infty]$ to $[i,j]$ for $i,j \in \mathbb{Z}$.

2.3 Reactive Synthesis

We now, equipped with LTL as a specification language, formally state the reactive synthesis problem. Let E and P be sets of environment and controlled variables, respectively. Let $s = (e,p) \in dom(E) \times dom(P)$ be a state of the system. Consider a LTL specification φ of assume-guarantee form

$$\varphi = \varphi_e \rightarrow \varphi_s, \tag{2.1}$$

where, roughly speaking, φ_e is the conjunction of LTL specifications that characterizes the assumptions on the environment and φ_s is the conjunction of LTL specifications that characterizes the system requirements.

The synthesis problem is then concerned with constructing a strategy, i.e., a partial function

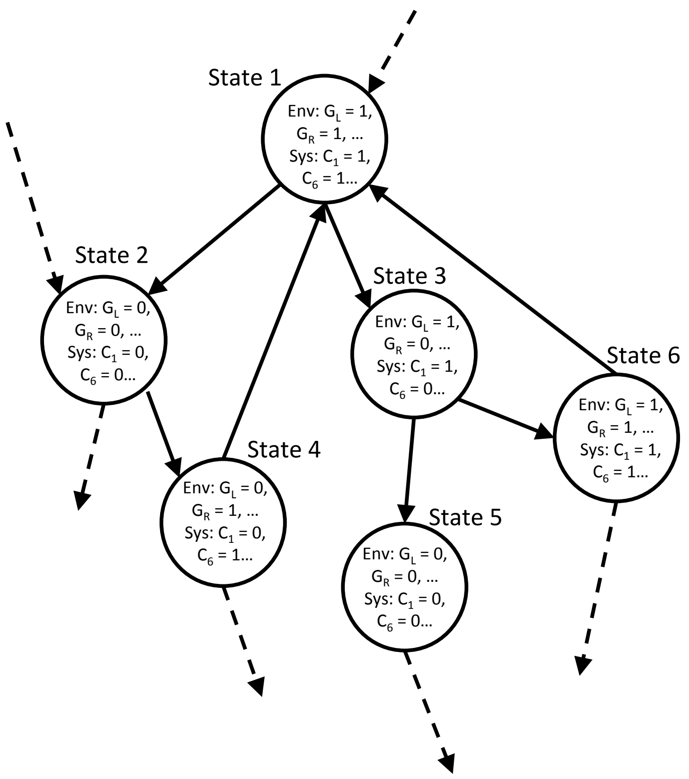

State 1
Env: $G_L = 1$, $G_R = 1$, … Sys: $C_1 = 1$, $C_6 = 1$…
State 2
Env: $G_L = 0$, $G_R = 0$, … Sys: $C_1 = 0$, $C_6 = 0$…
State 3
Env: $G_L = 1$, $G_R = 0$, … Sys: $C_1 = 1$, $C_6 = 0$…
State 4
Env: $G_L = 0$, $G_R = 1$, … Sys: $C_1 = 0$, $C_6 = 1$…
State 5
Env: $G_L = 0$, $G_R = 0$, … Sys: $C_1 = 0$, $C_6 = 0$…
State 6
Env: $G_L = 1$, $G_R = 1$, … Sys: $C_1 = 1$, $C_6 = 1$…

Figure 2.3: A portion of the resulting controller automaton for a synthesized problem. Dotted arrows represent transitions to states not depicted within the figure. Listed within each node is a valuation of environment and system variables. From state 1, an environment input determines whether the automaton moves to state 2 or state 3.

$f : (s_0 s_1 \dots s_{t-1}, e_t) \mapsto p_t$, that chooses the move of the controlled variables based on the state sequence so far and the behavior of the environment so that the system satisfies φ_s as long as the environment satisfies φ_e. The synthesis problem can be viewed as a two-player game between the environment and the controlled plant: the environment attempts to falsify the specification in (2.1) and the controlled plant tries to satisfy it. Figure 2.3 shows a portion of an example resulting automaton. Each state (node) represents a tuple of the current valuation of system and environment variables. State 1, for example, contains the initial states of both environment and system (where values are only partially listed in the figure). The system variable at the next step is determined by the environment. From state 1, if the environment determines that G_L and G_R are set both to 0, then the automaton goes to state 2, and the system variables C_1 and C_6 become 0. If the environment takes the transition from state 1 to state 3, then the system becomes $C_1 = 1$ and $C_6 = 0$.

For general LTL, it is known that the synthesis problem has a doubly exponential complexity in [73]. For a subset of LTL, namely generalized reactivity (1) (GR(1)), Piterman et al., have shown that it can be solved in polynomial time (polynomial in the number of valuations of the variables in E and P) [70]. GR(1) specifications restrict φ_e and φ_s to take the following form, for $\alpha \in \{e, s\}$,

$$\varphi_\alpha := \varphi_{\text{init}}^\alpha \wedge \bigwedge_{i \in I_1^\alpha} \Box \varphi_{1,i}^\alpha \wedge \bigwedge_{i \in I_2^\alpha} \Box \Diamond \varphi_{2,i}^\alpha, \tag{2.2}$$

where $\varphi_{\text{init}}^\alpha$ is a propositional formula characterizing the initial conditions; $\varphi_{1,i}^\alpha$ are transition relations characterizing safe, allowable moves and propositional formulas characterizing invariants; and $\varphi_{2,i}^\alpha$ are propositional formulas characterizing states that should be attained infinitely often. Many interesting temporal logic specifications can be expressed or easily transformed into GR(1) specifications. See [15, 70, 88, 89] for a more precise treatment of GR(1) synthesis and case studies in which GR(1) synthesis has been used for applications including hardware synthesis, motion planning for autonomous vehicles, and vehicle management systems.

Given a GR(1) specification, the digital design synthesis tool implemented in JTLV (a framework for developing temporal verification algorithm) [74] generates a finite automaton that represents a switching strategy for the system. The temporal logic planning (`TuLiP`) toolbox, a collection of python-based code for automatic synthesis of correct-by-construction embedded control software provides an interface to JTLV [90]. For examples discussed in this thesis, we primarily use `TuLiP`.

Additional two-player temporal logic game solvers include Anzu [42], Lily [40], Acacia [31], and Unbeast [29]. Anzu implements a GR(1) game solver symbolically. Lily accepts arbitrary LTL specifications and partially alleviates the resulting high computational cost through optimizations of the intermediate steps in the implementation [41]. Acacia and Unbeast focus on the concept of bounded synthesis from [82] and [27], respectively. See [28] for a detailed comparison of these tools. Finally, for temporal logic specifications in the form of safety formulas, it may be possible to obtain performance improvements by exploring solvers that are optimized to fragments (potentially more restrictive than GR(1)) of LTL, e.g., see [86].

2.4 Distributed Synthesis

In centralized control protocols the controller has access to measurements of all controlled and environment variables, and is able to determine the evolution of all controlled variables in order to satisfy a set of specifications. Because of their scale and complexity, control architectures for electric power systems on more-electric aircraft will likely have distributed structures. Reasons for migrating to distributed control architectures include:

Hardware challenges: A centralized controller onboard an aircraft requires wiring from a central processing unit to all components. The total length of wire can significantly increase the weight of the aircraft. Local controllers allows for shorter wires and increased efficiency due to this reduction in weight.

Increased resilience to failure: By distributing the implementation of the controller, the electric power system can be more robust to failures, i.e., if one portion of the electric power system malfunctions, the other sections are unaffected and can still be fully operational.

Reduction of computational complexity: With an increased number of electric components, the combination of configurations the controller must account for quickly becomes intractable for current verification and synthesis tools as well as testing. A distributed controller design correctly decomposes the design task into smaller subproblems each of which may be easier to cope with.

Advantages from the distribution of the control design come with increased importance of reasoning about the interfaces between the controlled subsystems. There is relatively extensive literature on compositional reasoning [31, 54, 64]. Here, we follow the exposition from recent work in [67]. Figure 2.4 illustrates the decomposition of global specifications into local specifications. For ease of presentation, consider the case where the system SYS is decomposed into two subsystems SYS_1 and SYS_2. For $i = 1, 2$, let E_i and P_i be the environment variables and controlled variables for SYS_i such that $P_1 \cup P_2 = P$ and $P_1 \cap P_2 = \emptyset$. Let φ_{e_1} and φ_{e_2} be LTL formulas containing variables in E_1 and E_2, respectively. Similarly, let φ_{s_1} and φ_{s_2} be LTL formulas in terms of $E_1 \cup P_1$ and $E_2 \cup P_2$, respectively. If the following conditions hold

1. any execution of the environment that satisfies φ_e also satisfies $(\varphi_{e1} \wedge \varphi_{e2})$,
2. any execution of the system that satisfies $(\varphi_{s1} \wedge \varphi_{s2})$ also satisfies φ_s, and
3. there exist two control protocols that realize the local specifications $(\varphi_{e1} \rightarrow \varphi_{s1})$ and $(\varphi_{e2} \rightarrow \varphi_{s2})$,

then, by a result in [67], implementing these two control protocols together leads to a system where the global specification $\varphi_e \rightarrow \varphi_s$ is met.

Two factors should be taken into account when choosing local environment and controlled variables E_1, E_2, P_1, and P_2 and the local specifications. The first is the size of the state space involved in the local synthesis problems. If the possible valuations of variables involved in local specifications are substantially less than the possible valuations of the variables in the global specification, then

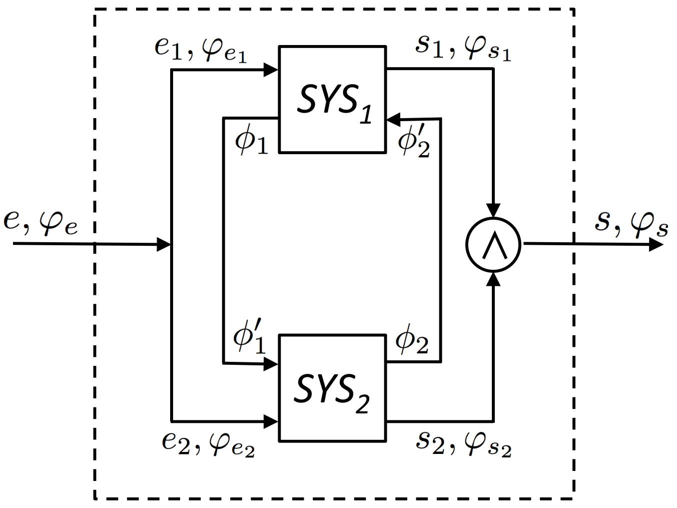


Figure 2.4: A schematic for the decomposition of global specifications into distributed controllers for two subsystems. The overall environment assumptions φ_e and system guarantees φ_s are distributed into the two subsystems SYS_1 and SYS_2. Each subsystem has its own local environment assumptions and system guarantees. In addition, SYS$_1$ has an extra set of local guarantees ϕ_1 that interact with SYS_2 as environment assumptions ϕ'_1, while SYS_2 guarantees contained in ϕ_2 act as environment assumptions ϕ'_2 for SYS_1.

distributed synthesis would be computationally more efficient than the centralized one (assuming the lengths of LTL formulas for the global and the local speciÞcations are of the same order). The second factor is the conservatism of the distributed synthesis. It is possible that even if the centralized problem is realizable, the local distributed synthesis may be unrealizable. Subsystems may need to interact with each other through shared variables (either information or physical values) in order to become realizable. As seen in Figure 2.4, subsystem SYS_1 provides additional guarantees ϕ_1 to subsystem SYS_2, evaluated as an environment assumption and denoted as ϕ'_1. The same interaction applies to the interface between SYS_2, which sends its own local guarantees ϕ_2 to SYS_1. If the following local specifications (and interface refinements) hold:

$$\phi'_2 \wedge \varphi_{e_1} \rightarrow \varphi_{s_1} \wedge \phi_1, \tag{2.3}$$

$$\phi'_1 \wedge \varphi_{e_2} \rightarrow \varphi_{s_2} \wedge \phi_2. \tag{2.4}$$

Then the global specification $\varphi_e \rightarrow \varphi_s$ is realizable. Indeed, let sets of executions be defined as

$$\sigma_e = \{\sigma \mid \sigma \models \varphi_e\}; \quad \varphi_{e'} = \{\sigma | \sigma \models (\varphi_{e1} \wedge \varphi_{e2})\};$$

$$\sigma_s = \{\sigma \mid \sigma \models \varphi_s\}; \quad \varphi_{s'} = \{\sigma | \sigma \models (\varphi_{s1} \wedge \varphi_{s2})\}.$$

Condition 1 implies that $\Sigma_{e'} \supseteq \Sigma_e$, whereas condition 2 implies that $\Sigma_{s'} \subseteq \Sigma_s$. Local variables and specifications should be chosen so that conditions 1 and 2 are satisfied. Moreover, the conservatism can be reduced by choosing φ_{e_j} and φ_{s_j} such that $\Sigma_{e'}$ is as “small" as possible, and the set $\Sigma_{s'}$ is as “large" as possible in the sense of set inclusion.

Chapter 3

Synthesis of Reactive Control Protocols with Timing

3.1 Overview

Controllers for an electric power system must be designed so that the system satisfies all safety and reliability properties and requirements. These requirements are usually text-based lists, oftentimes ambiguous in intent or inconsistent with each other. The process of verifying the correctness of a system with respect to specifications is expensive, both in terms of cost and time. In the following chapter, we "specify and synthesize" a solution to the design problem instead of "design then verify." In this approach, we begin by converting text-based system specifications for an electric power system into a mathematical formalism using a temporal logic specification language. From the set of system specifications, we then automatically synthesize centralized and distributed controllers, and examine design tradeoffs between different control architectures.

One of the challenges in automatically synthesizing controllers is its computational complexity. For a certain class of properties, a fragment of LTL known as Generalized Reactivity (1), a discrete planner can be automatically computed in polynomial time (with respect to the size of the state space) [70]. Applications of synthesis tools, however, are limited to small problems due to the state space explosion issue. To address this challenge, we utilize previous work on the compositional design of correct-by-construction, distributed protocols for an electric power system [67, 68]. Distribution of the design and implementation of the electric power system will reduce the computational complexity, as well as allow for the design of flexible control architectures in terms of modularity, fault-tolerance, and integrability [51]. The drawbacks to distributed architectures are in the coordination between subsystems and ensuring that overall system requirements are satisfied. Distributing system requirements introduces the notion of incompleteness in specifications (i.e., the lack of a guarantee subsystem requirements satisfy global specifications.) In addition, distributed controllers can be overly conservative (e.g., more generators need to be utilized in order to guarantee power to buses).

This chapter is structured as follows: Section 3.2 outlines the standard high-level specifications for an electric power system. Formalized LTL specifications are presented in Section 3.3. Section 3.4 addresses how actuation delays are captured. Sections 3.5 and 3.6 present the setup and results for a case study topology for both central and distributed controllers. Section 3.6.3 presents some benchmarks based on timing delays.

3.2 Specifications for Aircraft Electric Power Systems

Given a topology of an electric power system like that of the single-line diagram in Figure 2.1, the main design problem becomes determining all correct configurations of contactors for all flight conditions and faults that can occur in the system. For a configuration to be "correct" means that it satisfies system requirements, also referred to as specifications. We now discuss a few sample specifications relevant to the problems found in Figure 2.1.

Specifications are generally expressed in terms of safety, performance, and reliability properties. A few common ones considered in the typical electric power system control protocol design problem are listed below.

Safety: Safety specifications constrain the way each bus can be powered and the length of time it can tolerate power shortages. Increasing the number of generators operating at the same time increases the amount of power available to the electric power system. In order for AC generators to work in parallel with each other, however, they need to match their respective frequencies, and phase voltages. A mismatch in these properties can lead to loss of availability and even damage of the generator or distribution system. To avoid such difficulties of synchronization, we disallow any paralleling of AC sources, i.e., no bus should be powered by multiple AC generators at the same time.

Essential loads, such as flight critical actuators, are connected to essential AC and DC buses. These loads should never be unpowered for more than 50 msec. The 50 msec specification is a number used in industry standards in most aircraft power requirement documents. This "gap" time is short enough to ensure that load profiles are undisturbed (for safety of the aircraft), but is long enough for contactors to open or close and still avoid paralleling of sources.

The system is reconfigured through a series of changes in the contactor states. The time it

Table 3.1: Source Priority Table for HVAC Buses

Priority	Bus 1	Bus 2	Bus 3	Bus 4
1	G_2	G_3	G_4	G_5
2	G_5	G_2	G_5	G_2
3	G_3	G_5	G_2	G_4
4	G_4	G_4	G_3	G_3

takes for contactors to switch configurations will vary due to physical hardware constraints. Typical opening times can range between 10-20 msec, while closure times are between 15-25 msec [62]. Such delays need to be considered due to timing constraints on the buses and non-paralleling of sources.

Remark 2 *Specifications for DC components in the electric power system are the same as those described by the AC specifications except for two simplifications: (1) The non-paralleling of AC sources specification may be ignored, and (2) no DC bus may ever be unpowered.*

Performance: Performance specifications rank desired system configurations. A generator priority list is assigned to each bus specifying the order of sources each bus should be powered . If the first priority generator is unavailable, then it will be powered from the second priority generator, and so on. A hypothetical prioritization list is shown in Table 3.1 for HVAC Bus 1. Because G_2 is the first priority on the list, if the left high-voltage generator from Figure 2.1 is healthy, then HVAC BUS 1 receives power from that generator. If G_2 is unhealthy, then HVAC BUS 1 should receive power its second priority G_5, and so forth. These source priority tables are usually created manually, or borrowed from legacy systems. Thus, there is no guarantee on feasibility of all configurations or may not cover all possible conditions. Moreover, as there is no explicit priority between buses connected to generators, priority tables are oftentimes be contradictory. The end goal is to replace tables, which are designed with an implicit metric, with an explicit ”cost function” or metric written as a formal specification.

Reliability: Reliability specifications describe the bounds on probability of failures within the system. Every component comes with a reliability level. A level ϵ of reliability, for example, indicates that one failure will occur every $\frac{1}{\epsilon}$ hours of operation. Given multiple component failures, systems should be designed to tolerate any combination of component faults that has a joint probability of more than a certain pre-specified level. Practically, these reliability specifications determine the combination of simultaneous faults that need to be accounted for by the control protocol. An electric power system should still be able to satisfy its safety specifications given any combination of faults that lead to the pre-specified level. In the design procedure proposed in subsequent sections, reliability specifications are implicitly accounted for through the environment assumptions by limiting the number of generator faults that are allowed to occur at each step. If each component has a known failure rate, then no combination of failures can exceed a rate of, 10^{-9}, for example.

3.3 Formal Specifications For Aircraft Electric Power Systems

Given the topology in Figure 2.1, the following list details the temporal logic specifications that typically exist in the synthesis of control protocols for electric power systems.

Environment Assumptions: Let $\mathcal{G}$ represent the set of all generators in the electric power system topology. Let the Boolean variable g denote the health status of generator $G \in \mathcal{G}$. That is, the lowercase symbol represents the health status of generator denoted by the corresponding uppercase symbol. We use a similar convention between upper and lowercase symbols in the remainder of the paper. The environment assumption states that at least one generator must be healthy, i.e., have a status of 1, at any given time. This is written as

$$\square \left\{ \bigvee_{G \in \mathcal{G}} (g = 1) \right\}. \tag{3.1}$$

Unhealthy Generators: An unhealthy generator connected to the system could create a short-circuit failure, generate excess torque, cause overheating, or possible fires. We require any contactor adjoining a generator to open when that generator becomes unhealthy. Let $\mathcal{C}$ represent the set of all contactors in the electric power system. For $G \in \mathcal{G}$, let $\mathcal{C}_G \subseteq \mathcal{G}$ be the contactors directly neighboring G. In Figure 3.1, for example, the sets $\mathcal{C}_{G_1}$ and $\mathcal{C}_{G_2}$ consist of contactors C_1 and C_2, respectively. For a contactor C, let c be its status (for example, 0 represents an open contactor, 1 a closed contactor). Furthermore, the Boolean variable $\tilde{c}$ denotes the controller command (intent) for contactor C. Note the difference between status of contactor, denoted by c and intent of contactor $\tilde{c}$. Once the intent $\tilde{c}$ gets set, that command then gets executed, i.e., status c follows $\tilde{c}$ at a possibly later time step.

If a generator becomes unhealthy, then the contactors connecting to it should be commanded open, i.e., take the value of 0. The specification for disconnecting an unhealthy generator can be written as

$$\bigwedge_{G \in \mathcal{G}} \square \left\{ (g = 0) \rightarrow \bigwedge_{C \in \mathcal{C}_G} (\tilde{c} = 0) \right\}. \tag{3.2}$$

No Paralleling of AC Sources: One way to avoid paralleling AC sources is to explicitly enumerate and eliminate all configurations in which buses can be powered from multiple sources.

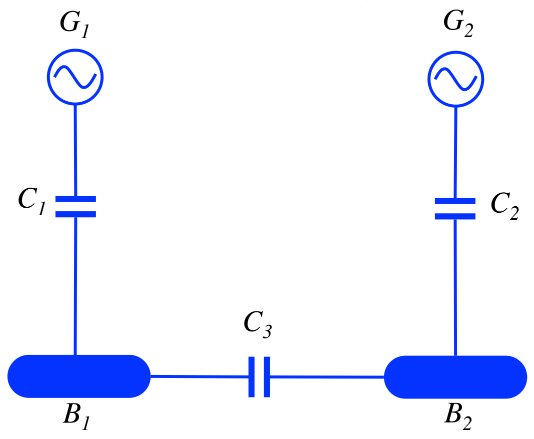


Figure 3.1: A single-line diagram with two generators, two buses, and three contactors. Paralleling of AC sources can occur if all three contactors C_1, C_2, and C_3 are all closed.

In the example shown in Figure 3.1, paralleling could occur if contactors C_1, C_2, and C_3 were all closed at the same time. A specification would then be to never allow all contactors along a path to close at the same time if that path could connect two AC sources. This "global" approach requires enumerating all possible paths between pairs of AC sources, with the number of paths and components increasing as the topology becomes more complex.

We take a "localized" view on specifications that no AC bus can be simultaneously powered from multiple sources. Instead of examining entire paths connecting generators to buses, we focus on the source of power coming into or flowing out of each bus. We first introduce the notion of power flow direction in contactors, and then examine the flow direction at each bus.

Power flow direction is defined for contactors directly connecting two buses. Contactors connecting generators to buses are assumed to only allow power to flow in one direction from generator to bus. (Note that while this assumption is valid for this problem formulation, in reality the contactor must respond in a manner to avoid backfeeding power into a generator.) Let the set $\mathcal{C}_B \subset \mathcal{C}$ be the set of all contactors that directly connect two AC buses. Let each bus connected to a contactor in $\mathcal{C}_B$ represent a "side" or direction from which power can flow into or out of, and denote them as direction **1** and direction **-1**. In Figure 3.1, for example, contactor C_3 directly connects buses B_1 and B_2, which are located on side **1** and **-1** of C_3, respectively. Consider contactor $C \in \mathcal{C}_B$. The variable $\tilde{c}$ is the intended status of the contactor, and can take values of $\{-1, 0, 1\}$ corresponding to a closed contactor with power flowing into side **-1**, an open contactor, and a closed contactor with power flowing into side **1**, respectively. Note that the status of contactors connecting generators is Boolean, while the status of contactors connecting two AC buses can take three values.

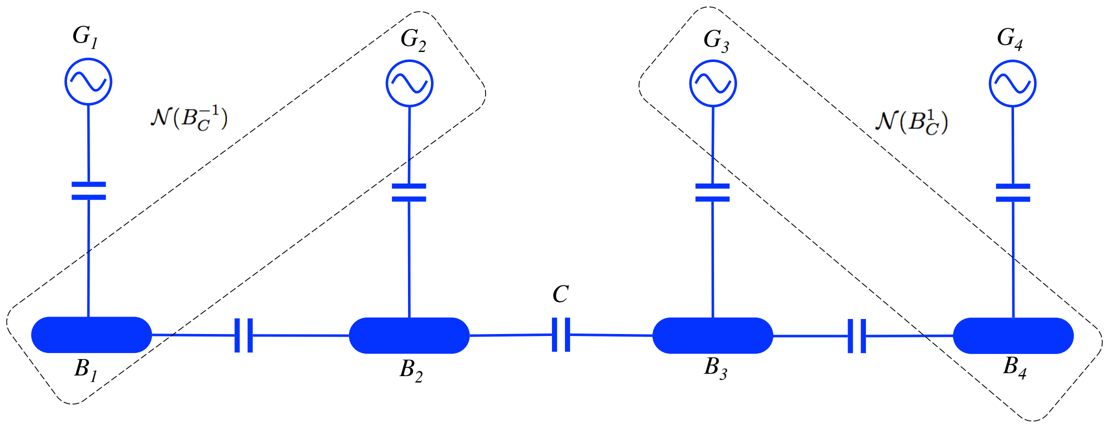


Figure 3.2: A single-line diagram depicting contactor C and its connecting two buses B_2 and B_3, as well as neighboring nodes in $\mathcal{N}(B_C^{-1})$ and $\mathcal{N}(B_C^{1})$.

For $C \in \mathcal{C}_B$, let B_C^1 denote the bus on side **1** of contactor C, and B_C^{-1} the bus on side **-1** of contactor C. The set $\mathcal{N}(B_C^1)$ contains all nodes, defined as either a bus or a generator, that are directly connected to the bus on side **1** of contactor C. Similarly, $\mathcal{N}(B_C^{-1})$ is the set of nearest nodes connected to the bus on side **-1** of contactor C. Sets $\mathcal{N}(B_C^1)$ and $\mathcal{N}(B_C^{-1})$ do not include any contactors. For any bus B, let the Boolean variable b represent its power status (0 for unpowered, 1 for powered). Consider contactor C in Figure 3.2, where $B_C^1 = B_3$, $B_C^{-1} = B_2$, $\mathcal{N}(B_C^1) = \{G_3, B_4\}$, and $\mathcal{N}(B_C^{-1}) = \{G_2, B_1\}$.

The direction of power flow through a contactor is defined by identifying the status of buses directly connected to a contactor, and neighboring components $\mathcal{N}(B)$ of those buses. For each component $X \in \mathcal{N}(B_C^1)$ or $X \in \mathcal{N}(B_C^{-1})$, x is the status. For contactors $C \in \mathcal{C}_b$, if no node in $\mathcal{N}(B_C^1)$ is powered or healthy (depending on whether the node is a bus or generator, respectively), then C cannot direct power from side **1** to side **-1** (i.e., $\tilde{c}$ cannot be 1). Alternatively, if no node in $\mathcal{N}(B_C^{-1})$ is powered or healthy, then C cannot direct power from side **-1** to side **1** (i.e., $\tilde{c}$ should not be -1). Specifications for contactor directionality can be written as the following.

If the bus on side **1** of contactor C is unpowered and none of its neighboring nodes are powered, then its states should be set to -1 (cannot direct power from side **1**):

$$\bigwedge_{C \in \mathcal{C}_b} \square \left[\neg \left(\left(b_C^1 = 1\right) \wedge \bigvee_{X \in \mathcal{N}(B_C^1)} (X = 1) \right) \rightarrow \neg\,(c = -1) \right]. \tag{3.3}$$

If the bus on side **-1** of contactor C is unpowered and none of its neighboring nodes are powered,

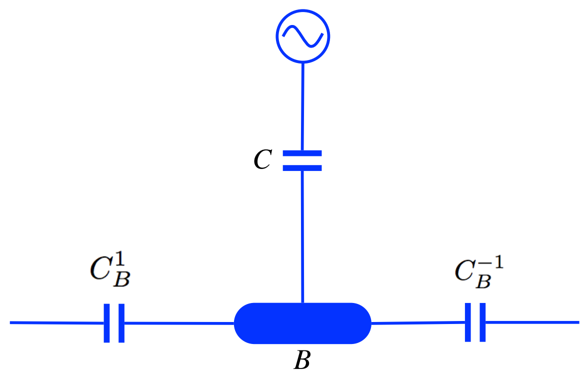


Figure 3.3: A portion of the single-line diagram from Figure 2.1. Non-paralleling specifications are written from the "local" viewpoint of each bus. Bus B is on side **-1** of contactor C_B^{-1}, and on side **1** of contactor C_B^1. No combination of two contactors can be connected (and directing power into a bus) at the same time.

then its states should not be set to 1 (cannot direct power from side **-1**):

$$\bigwedge_{C \in \mathcal{C}_b} \Box \left[\neg \left((b_C^{-1} = 1) \wedge \bigvee_{X \in \mathcal{N}(B_C^{-1})} (X = 1) \right) \rightarrow \neg (c = 1) \right]. \tag{3.4}$$

Once contactor directionality is established, specifications for non-paralleling of AC sources can be examined at the "local" level by considering each individual AC bus. Let $\mathcal{B}_{AC}$ be the set of AC buses. We now consider every combination of contactors for which power may flow into the same bus. Consider again the set $\mathcal{C}_G \subset \mathcal{C}$ to be the set of all contactors connecting bus to a neighboring generator. In Figure 2.1, each bus has, at most, three contactors through which power can flow into the bus. The following specifications are written for this case, and may be generalized for any number of contactors through which power can flow into a bus. For each bus $B \in \mathcal{B}_{AC}$, let each contactor $C \notin \mathcal{C}_G$ connected to B represent a "side" or direction of B. In typical configurations only two directions are needed, though this method can be generalized for more sides. For bus B that is on side **1** of a contactor, denote that contactor as C_B^1. Denote contactor C_B^{-1} as the contactor for which bus B is on the **-1** side. We disallow any cases where power can flow into the bus through multiple paths. These specifications can be written as

$$\begin{gathered}\bigwedge_{B\in\mathcal{B}_{AC}} \square\neg \bigvee_{G\in\mathcal{N}(B),C\in\mathcal{C}_G} \left[(c=1)\wedge(c_B^1=1)\right],\\ \bigwedge_{B\in\mathcal{B}_{AC}} \square\neg \bigvee_{G\in\mathcal{N}(B),C\in\mathcal{C}_G} \left[(c=1)\wedge(c_B^{-1}=-1)\right],\\ \bigwedge_{B\in\mathcal{B}_{AC}} \square\neg \left[(c_B^1=1)\wedge(c_B^{-1}=-1)\right].\end{gathered} \tag{3.5}$$

Power Status of Buses: A bus can only be powered if a neighboring generator is healthy or a neighboring bus is powered, and the contactor connecting to that bus is closed. If no neighboring node is healthy or powered, or the contactor is open, then the bus will be unpowered. Let $\mathcal{B}$ be the set of all AC and DC buses. Consider generators $G\in\mathcal{N}(B)$ to be the neighboring generators of bus B. For all generator-contactor pairs directly neighboring a bus, the specification can be written as

$$\bigwedge_{B\in\mathcal{B}} \square\left\{\left[\bigvee_{C\in\mathcal{C}_G,G\in\mathcal{N}(B)} ((c=1)\ \wedge\ (g=1))\right]\rightarrow(b=1)\right\}. \tag{3.6}$$

We then examine all neighboring bus/contactor pairs connected to bus B. Let $B^*\in\mathcal{N}(B)$ be a neighbor bus to B, where $\mathcal{N}^1(B)\subset\mathcal{N}(B)$, and $\mathcal{N}^{-1}(B)\subset\mathcal{N}(B)$. Bus B is on side **1** of components in $\mathcal{N}^1(B)$, and side **-1** of $\mathcal{N}^{-1}(B)$. A bus may be powered if one of the following holds:

Bus B is powered if it is on side **1** of a contactor and neighboring bus pair, the contactor is closed with power flowing in the direction of side **1** and the neighboring bus is powered. Then,

$$\bigwedge_{B\in\mathcal{B}} \square\left\{\left(\bigvee_{B^*\in\mathcal{N}^1(B)} (b^*=1)\ \wedge\ (c_B^1=1)\right)\rightarrow(b=1)\right\}. \tag{3.7}$$

Bus B is powered if on side **-1** of the contactor and bus pair, the contactor is closed with power flowing in the direction of side **-1**, and the neighboring bus is powered. This is written as

$$\bigwedge_{B\in\mathcal{B}} \square\left\{\left(\bigvee_{B^*\in\mathcal{N}^{-1}(B)} (b^*=1)\ \wedge\ (c_B^{-1}=-1)\right)\rightarrow(b=1)\right\}. \tag{3.8}$$

If none of the above three conditions hold, bus B will be unpowered.

Safety Criticality of Buses: Certain buses within the distribution system will be connected to safety-critical loads, e.g., flight actuators or de-icers, and need to remain powered. Due to non-paralleling specifications, however, these buses also need to be able to stay unpowered for short lengths of time in order to reconfigure contactors without violating specifications. Let $\mathcal{B}_s$ be the set of all safety-critical buses. Denote the allowable length of time a bus can remain unpowered as T. For example, typical values for T fall in the 50 msec range [62]. LTL reasons about temporal ordering, but does not explicitly address the notion of real-time. Time in this formulation is implemented through an additional clock variable θ_B associated with bus B, and where each "tick" of the clock represents δt time. The "tick" of the clock δt represents both the time it takes for a contactor to open or close (e.g., 10 msec), and the controller sampling time. Thus θ_B can takes values from $\{0, \delta t, 2\delta t, \ldots, \frac{T}{\delta t}\}$. For each safety-critical bus in $B \in \mathcal{B}_s$, these specifications can be written as the following.

If bus B is unpowered, then in the next step, clock variable θ_B will increment by 1 unit, which is written as

$$\square \left\{(b = 0) \rightarrow (\bigcirc \theta_B = \theta_B + \delta t)\right\}. \tag{3.9}$$

If bus B is powered, then in the next step, clock variable x_B is reset to 0. This is written as

$$\square \left\{(b = 1) \rightarrow (\bigcirc \theta_B = 0)\right\}. \tag{3.10}$$

Clock variable x_B will never be greater than the maximum allowable unpowered time $\frac{T}{\delta t}$. This is implemented by

$$\square \left\{\theta_B \leq \frac{T}{\delta t}\right\}. \tag{3.11}$$

3.4 Capturing Actuation Delays

LTL can be used to specify "real-time" properties for synchronous systems in which all processes (i.e., components) proceed in a lock-step manner. The next operator has a "time" measure so that, for a given property φ, $\bigcirc\varphi$ signifies at the next time instant φ is true. To specify a property occurring at some point in the future, multiple next operators can be used, such that $\bigcirc^k \varphi \triangleq \bigcirc \bigcirc \ldots \bigcirc \varphi$ asserts that property φ holds k time instants in the future. As an alternative to multiple next operators, the "timed" specifications in the electric power system uses a clock variable to define an equivalent property.

For simplicity, we can assume ideal contactors that can be instantaneously controlled. It is possible, however, to capture delays in contactor opening and closing times, as well as the communication delays between the controller and the contactors. To this effect, one can introduce a controlled variable $\tilde{c}$ to represent the controller intent for contactor C and treat the contactor as an environment variable. The uncertain delay between the controller intent and contactor state can be handled by the use of an additional clock variable x_C for each contactor C, where each "tick" of the clock represents δ time. If the contactor intent is open and the contactor state is closed, the contactor opens within $[T_{o_{min}}, T_{o_{max}}]$ units of time unless a close command is issued before it opens. If the contactor intent is closed and the contactor state is open, the contactor closes within $[T_{c_{min}}, T_{c_{max}}]$ units of time unless an open command is issued before it closes. Once the contactor intent is set, if the contactor state does not match the intent, at the next step clock x_C will increase by δ. If contactor state and intent match, then at the next step clock x_C resets to zero:

$$\Box\{(\bigcirc c = \tilde{c}) \rightarrow (\bigcirc x_C = 0)\}.$$

When the control command is the same as the contactor state, the contactor state remains the same, i.e.,

$$\Box\{(\tilde{c} = c) \rightarrow (\bigcirc c = c)\}.$$

Finally, the assumption capturing the contactor closing behavior in relation to the controller input intent is given by

$$\begin{aligned}\Box\left\{(\tilde{c} = 1 \wedge c = 0 \wedge (x_C < T_{c_{min}})) \rightarrow (\bigcirc c = 0 \wedge \bigcirc x_C = x_C + \delta)\right\},\\ \Box\left\{(\tilde{c} = 1 \wedge c = 0 \wedge (x_C \geq T_{c_{min}})) \rightarrow (\bigcirc c = 1 \vee \bigcirc x_C = x_C + \delta)\right\},\\ \Box(x_C \leq T_{c_{max}}).\end{aligned}$$

The contactor opening behavior can be formally captured in a similar manner. The formulas mentioned in this remark enter to the control synthesis problem as new environment assumptions when delays are taken into account.

3.5 Case Study

We address the problem of primary distribution in an electric power system by examining a simplified version of the single-line diagram. Figure 3.4 shows the portion of the single-line diagram considered for the problem formulation used in the rest of this chapter. This topology consists of high-voltage

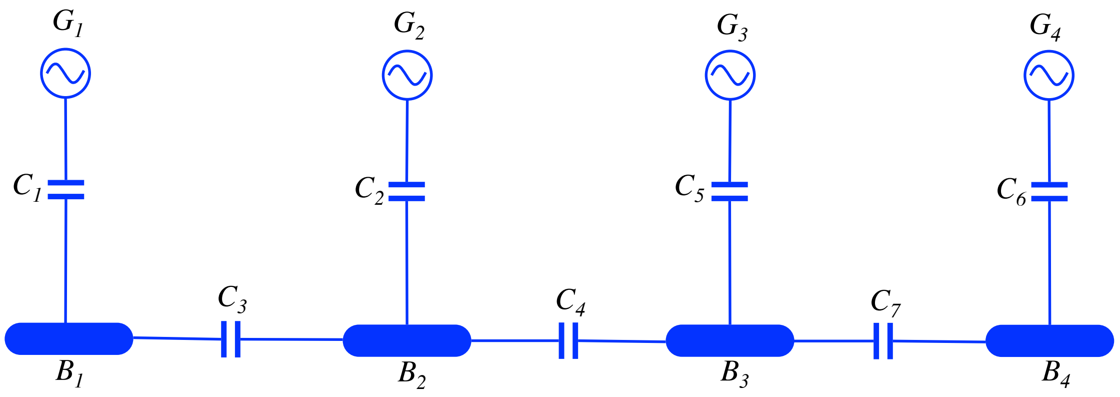


Figure 3.4: Simplified diagram of the single-line diagram used in the centralized problem. Four power sources connect to four buses through a series of seven contactors.

AC components: four generators connect to four buses via seven contactors.

3.5.1 Variables

Variables used in this formulation, and shown in Figure 3.4, are classified as environment, controlled, or dependent.

Environment Variables: Consider G_1 and G_4 to be standard high-voltage AC generators, while G_2 and G_3 are backup generators connected to the APU. The health statuses of the all four sources g_1, g_2, g_3, and g_4 can each take values of healthy (1) and unhealthy (0). Again, we distinguish component variables and status variables by upper and lower cases, e.g., the first generator is represented by G_1, while its health status is denoted by g_1.

Controlled Variables: The statuses c_1, c_2, c_5, c_6 of contactors connecting generators to buses can each take values of open (0) or closed (1). A closed contactor will allow power to pass through, while an open one does not. The statuses (c_3, c_4, c_7) of contactors located between buses can take three values. A value of 0 denotes an open contactor. A value of **-1** or **1** signifies a contactor is closed and that power is flowing from side **-1** or **1**, respectively.

Dependent Variables: The power statuses (b_1, b_2, b_3, b_4) of buses can be either powered (1) or unpowered (0) depending on the status of neighboring contactors and generators.

3.5.2 Specifications

Given the topology in Figure 3.4, the specifications described in Section 3.3 reduce to the following specifications used in the synthesis problem for the simplified single-line diagram.

Environment Assumption: The assumption that at least one power source is always healthy from (3.1) becomes

$$\square\left\{(g_1 = 1) \ \vee \ (g_2 = 1) \ \vee \ (g_3 = 1) \ \vee \ (g_4 = 1)\right\}. \tag{3.12}$$

No Paralleling of AC Sources: In Figure 3.4, an instance of paralleling may occur if G_1 and G_2 are both healthy, and contactors C_1, C_2, and C_3 are all closed. Consider, for example, power flow direction for contactor C_3. In Figure 3.4, we define bus B_1 as the bus on side **-1** of C_3, and bus B_2 on side **1** of C_3. B_1 corresponds to $B_{C_3}^{-1}$ from notation used in Section 3.3, while $B_2 = B_{C_3}^{-1}$. Then, the neighbor nodes of B_1 is G_1, i.e., $\mathcal{N}(B_1) = \{G_1\}$, and $\mathcal{N}(B_2) = \{G_2, B_3\}$. Equations (3.3) and (3.4) can be reduced to the following.

If generator G_1 is unhealthy and bus B_1 is unpowered, then contactor C_3 cannot direct power from side **-1** to side **1**, i.e., it cannot take a value of 1, and the intent variable $\tilde{c}_3$ should be assigned accordingly. This is written as

$$\square\left\{\neg\left((g_1 = 1) \wedge (b_1 = 1)\right) \rightarrow \neg(\tilde{c}_3 = 1)\right\}. \tag{3.13}$$

If generator G_2 is healthy and B_2 is unpowered, or if B_3 and B_2 are unpowered, then C_3 cannot direct power from side **1** to side **-1**, i.e., take a value of -1, and the intent variable $\tilde{c}_3$ should be assigned accordingly. This can be written as

$$\square\Big\{(\neg((g_2 = 1) \wedge (b_3 = 1)) \vee (\neg((b_2 = 1) \wedge (b_3 = 1))) \rightarrow \neg(\tilde{c}_3 = -1)\Big\}. \tag{3.14}$$

A similar argument is made for contactor statuses c_4 and c_7.

Given direction of flow through contactors, we can examine each bus and eliminate any configuration of contactors which may allow for paralleling of sources. Consider bus B_2, which we define to be on side **1** of C_3 and on side **-1** of C_4. Following the notation in Section 3.3, contactor $C_2 \in \mathcal{C}_{B_2}$, $C_3 = C_{B_2}^{1}$, and $C_4 = C_{B_2}^{-1}$. Then, equation (3.5) reduces to the following specifications for bus B_2

$$\begin{aligned} &\square\left\{\neg((c_2 = 1) \ \wedge \ (c_3 = 1))\right\}, \\ &\square\left\{\neg((c_2 = 1) \ \wedge \ (c_4 = -1))\right\}, \\ &\square\left\{\neg((c_3 = 1) \ \wedge \ (c_4 = -1))\right\}. \end{aligned} \tag{3.15}$$

Specifications for buses B_1, B_3, and B_4 are applied similarly.

Power Status of Buses: Consider bus B_2, located on side **-1** of contactor C_4, and on side **1** of contactor C_3. Equations (3.6) and (3.8) reduce to the following.

For generator $G_2 \in \mathcal{N}(B_2)$ and $C_2 \in C_{G_2}$, if G_2 is healthy and contactor C_2 is closed, then B_2 will be powered. This is written as

$$\Box\{((g_2 = 1) \ \wedge \ (c_2 = 1)) \rightarrow (b_2 = 1)\}. \tag{3.16}$$

For $C_3 = C^1_{B_2}$ and $B_1 \in \mathcal{N}^1(B_2)$, if bus B_1 is powered and contactor C_1 is closed with power flowing into side **1**, then B_2 will be powered. This is written as

$$\Box\{((b_1 = 1) \ \wedge \ (c_3 = 1)) \rightarrow (b_2 = 1)\}. \tag{3.17}$$

For $C_4 = C^{-1}_{B_2}$ and $B_3 \in \mathcal{N}^{-1}(B_2)$, if bus B_3 is powered and contactor C_4 is closed with power flowing into side **-1**, then B_2 will be powered. This is written as

$$\Box\{((b_3 = 1) \ \wedge \ (c_4 = -1)) \rightarrow (b_2 = 1)\}. \tag{3.18}$$

If none of the previous properties holds, then B_2 will be unpowered, written as

$$\begin{aligned}\Box\{(\neg((g_2 = 1) \ \wedge \ (c_2 = 1)) \vee ((b_1 = 1) \ \wedge \ (c_3 = 1)) \\ \vee((b_3 = 1) \wedge \ (c_4 = -1))) \rightarrow (b_2 = 0)\}.\end{aligned} \tag{3.19}$$

A similar set of specifications is applied for bus statuses B_1, B_3, and B_4.

Safety Criticality of Buses: In this problem, we consider buses B_1 and B_4 to be safety-critical buses, and can be unpowered for no longer than five time steps. Each "tick" of the clock variable θ_{B_1} and θ_{B_4} represents 10 msec. A safety specification for bus B_1 is of the following form:

If B_1 is unpowered, then at the next time step clock θ_{B_1} increments by one "tick" such that

$$\Box\{(b_1 = 0) \rightarrow (\bigcirc\theta_{B_1} = \theta_{B_1} + 1)\}. \tag{3.20}$$

If B_1 is powered, then at the next time step reset clock θ_{B_1} to zero. This is written as

$$\Box\{(b_1 = 1) \rightarrow (\bigcirc\theta_{B_1} = 0)\}. \tag{3.21}$$

To ensure that B_1 is never unpowered for more than 5 steps (e.g., 50 msec), the specification becomes

$$\Box\{\theta_{B_1} \leq 5\}. \tag{3.22}$$

Unhealthy Generators: When a generator becomes unhealthy the controller will open its nearest contactor connecting the generator to a bus. In Figure 3.4, the set of neighboring contactors to generators are $\mathcal{N}(G_1) = C_1$, $\mathcal{N}(G_2) = C_2$, $\mathcal{N}(G_3) = C_5$, and $\mathcal{N}(G_4) = C_6$. If, for example, generator G_3 becomes unhealthy, its neighboring contactor status intent should be set to open (0). This specification can be written as

$$\square \{(g_3 = 0) \to (\tilde{c}_5 = 0)\}. \tag{3.23}$$

3.6 Results

Consider a system model $\mathbb{S}$ with a set of variables $V = S \cup E$. Environment variables E includes generators $G_1 - G_4$, and system variables S consist of contactors $C_1 - C_7$ and buses $B_1 - B_4$. Specification φ consists of φ_e and φ_s such that

$$\varphi = (\varphi_e \implies \varphi_s). \tag{3.24}$$

Given environment assumption φ_e from Eq. (3.12), and φ_s as the conjunction of all specifications from Eqs. (3.13)-(3.23), we synthesize a control protocol such that Eq. (3.24) holds. The output of the synthesis procedure includes a discrete planner represented as a finite-state atuomaton. States are pairs of system and environment states. If the system follows the transitions in the automaton, the system will satisfy its requirements under all allowable environment actions.

3.6.1 Centralized Controller Design

We now present the results for the centralized case of the electric power system design problem with variables and specifications discussed in the previous section. Figure 3.5 shows the simplified single-line diagram used in problem formulation overlaid with a sample simulation run. The horizontal axis of each graph in the figure represents the step of the simulation, starting at step 0 and ending with step 5.

The four graphs in row 1 correspond to the statuses of the environment variables. These values are arbitrarily input, subject to the restrictions placed on the environment. At each step, generator statuses can switch between healthy and unhealthy as long as at least one source remains healthy. Graphs in rows 2 and 3 correspond to the contactor statuses generated from the synthesized control protocol. Because power can only flow from a generator, the graphs for the contactors shown in

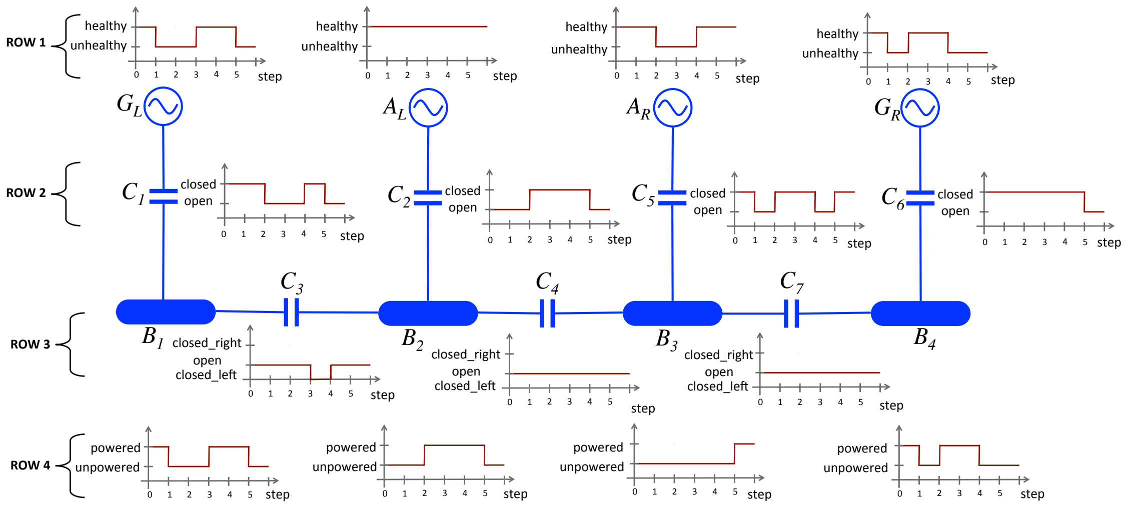


Figure 3.5: A simulation result for a centralized controller for the electric power system. The horizontal axis represents the simulation step. Row 1 shows the environment inputs for generator healths. Based on these values, the controller values for contactors are set to either open or closed, as seen in Row 2. Additionally, Row 3 shows the direction of power flow through contactors C_3, C_4, and C_7. Row 4 shows the power status for all four buses.

row 2 can only take values of open or closed. Graphs in row 3, however, can take three values corresponding to open or closed (with a direction). Graphs in row 4 correspond to the buses, and the vertical axis represents the power status of eachbus. Because buses are dependent variables, these values are determined by the environment variables as well as the contactor configurations.

To better understand the results shown in Figure 3.5 let us examine the simulation graphs for a single step, namely step 2. Generator G_1 is unhealthy and contactor status C_1 is open. Generator G_2 is healthy, and C_2 is closed. Bus B_2 is powered because it is connected to G_2, and B_1 is unpowered because both neighboring contactors C_1 and C_3 are open. Meanwhile, generator G_4 is healthy and C_6 is closed. Therefore, bus B_4 is powered. Note, however, that C_5 remains closed even though the right auxiliary generator is unhealthy. In the previous step, G_3 was healthy, and its intent to open $\tilde{c}_5$ in step 2 does not get implemented until step 4. In order to ensure non-paralleling of sources, contactor C_7 must remain open at step 2 because C_5 is closed, even though no power is flowing from generator G_3. As a result, bus B_3 is unpowered.

For safety-critical buses B_1 and B_4, their statuses are never unpowered for more than two time steps throughout the entire simulation sequence. This specification is not imposed on the middle two buses, however, and and thus B_3 can remain unpowered for five steps without violating any system requirements. In addition, at no time in the simulation run are AC sources paralleled. Consider, for example, power flowing to bus B_1. When contactor C_1 is closed (steps 0, 1, and 4), C_3 is always open.

The synthesis process produces a control protocol in the form of a finite state automaton. The resulting automaton for the electric power system centralized controller takes roughly one minute to solve on a MacBook Pro with a 2 GHz Intel Core Duo processor, and has 200 states. Within each automaton state is a list of successor states, which represent possible configurations for the system depending on the behavior of the environment states. Once the environment acts, then the system responds and the automaton steps to its next state. From State 0, for example, the automaton can move to State 1 if all generators and APUs become unhealthy, or move to State 2 if the right APU remains healthy but the other three power sources become unhealthy. Note that State 1 has no successor states because its environment violates the assumption that at least one power source remain healthy at all times. Thus as long as the environment satisfies its assumption, then the system will satisfy its specifications. We can also synthesize a centralized case where the total number of contactors allowed to switch at each particular time step is limited by hardware constraints. Consider contactors $C1$, $C2$, $C5$, and $C6$ to be controlled separately because they are connected to generators. The remaining three contactors, however, are physically controlled by a single hardware that is only capable of switching two contactors at one time. This problem, with the environment assumption (one source is always healthy), no longer becomes realizable as it violates the safety requirements for buses. If the environment assumption is relaxed, i.e. at least two sources must always remain powered, then the problem once again becomes realizable. A similar approach holds for the case when only one of the three middle contactors can be switched at one time.

3.6.2 Distributed Control Architecture

In this section we describe the results for a distributed control structure based on the refinement technique discussed in Section 2.4. More specifically, we decompose the centralized electric power system topology into two smaller subsystems and synthesize two local controllers. When implemented together, these controllers are guaranteed to be correct with respect to the global specification. The physical decomposition of the electric power system is shown in Figure 3.6. Let SYS_1 represent subsystem on the left, and SYS_2 the subsystem on the right. The environment and system variables for the two subsystems are denoted by e_1, s_1, e_2 and s_2, respectively.

We now present results for two types of distributed control architectures: master/slave and bi-directional.

Master/Slave Control Architecture: For a master/slave architecture, power flow between the decomposed systems is controlled by one side, and unidirectional only. For the decomposition

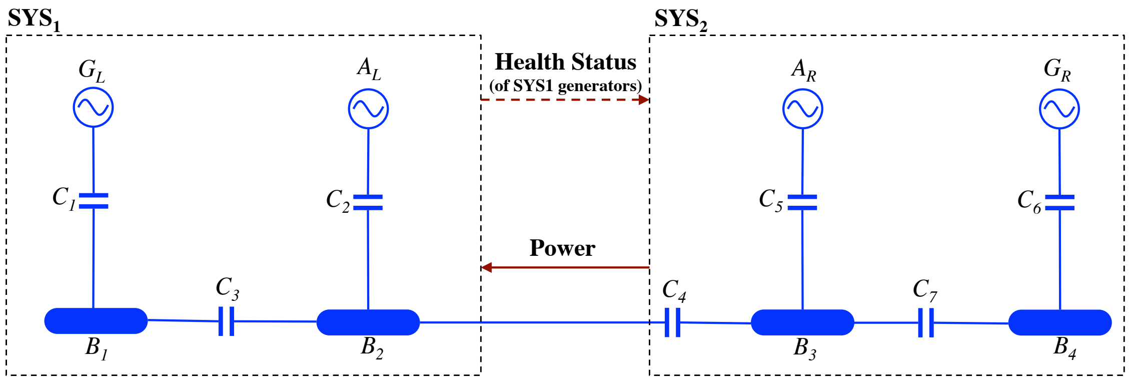


Figure 3.6: A distributed controller decomposition for the electric power system. Components enclosed within the dashed rectangles are controlled by their own respective controllers. The dashed arrow represents information flow, in the form of a health status variable, directed from SYS_1 to SYS_2. The solid arrow represents the physical transfer of power from SYS_2 to SYS_1.

shown in Figure 3.6, subsystem SYS_2 is the "master" and can control the supply of power that can flow via contactor C_4. Subsystem SYS_1 is the "slave" and can only receive power when SYS_2 provides it. We decompose the global environment assumption, in which at least one power source must remain healthy at each step, such that

$$\varphi_{e_2} = \square(g_3 = 1 \vee g_4 = 1),$$

$$\varphi_{e_1} = \square(true).$$

The specification for φ_{e_1} states that there are no restrictions on the behavior of φ_{e_1}. The assumption placed on φ_{e_2} ensures that for any execution $\sigma \in \Sigma$, the controller for SYS_2 is able to supply power to SYS_1 at any step. Health status information for g_1 and g_2 are sent to the SYS_2 via a health status variable H_1. The variable is set to 0 if neither source is healthy, and is set of 1 if either g_1 or g_2 is healthy so that φ_{e_2} can assume knowledge about the health status of the left side.

In order for the master/slave distributed synthesis problem to become realizable, additional assumptions and guarantees (i.e., interface refinements) need to be implemented. It is not enough for generators G_3 and G_4 to be able to generate power at all steps. The controller for SYS_2 must also be able to guarantee that power can be delivered to SYS_1. Thus, we introduce ϕ_2 as a guarantee for controller SYS_2, and denote ϕ_2' as an assumption for controller SYS_1. Because the master subsystem controls the flow of power, a single-sided refinement is sufficient for the design problem to be realizable, and we can set $\phi_1 = true$. The additional specification ϕ_2 imposes conditions on contactor status c_4 and bus status b_3 (the components nearest to the interface of SYS_2 and SYS_1).

These specifications are of the following form: Bus B_3 is never unpowered for a pre-specified period of time T. Essentially, B_3 becomes a safety-critical bus, and we introduce a variable t_3 that is used as a counter to monitor the power status

$$\square\{(b_3 = 0) \rightarrow (\bigcirc t_3 = t_3 + 1)\} \ \wedge \ \square\{(b_3 = 1) \rightarrow (\bigcirc t_3 = 0)\} \ \wedge \ \square\{t_3 \leq T\}.$$

If health status $H_1 = 0$, i.e., both G_1 and G_2 are unhealthy, then, whenever B_3 is powered, C_4 will close

$$\square\{((H_1 = 0) \wedge (B_3 = 1)) \rightarrow (\tilde{c_4} = -1)\}.$$

A similar modification is made for the case when power flows from SYS_1 to SYS_2 (and SYS_2 still remains master). In both of the cases discussed in the master/slave architecture, all other specifications remain the same as those discussed from Section 3.3 and decomposed with their respective components. Simulation results are comparable to those for the centralized controller, shown in Figure 3.5, and thus omitted.

Decentralized Control Architecture: Consider again the physical decomposition shown in Figure 3.6, where power is allowed to flow from either subsystem to the other. The physical actuation of contactor C_4 is still controlled by the right side. The environment variables for SYS_1 include G_1, G_2, and C_4, while environment variables for SYS_2 contain G_3, G_4, B_2, and H_1. Note that this differs from the master/slave control architecture with the necessary addition of B_2 as an environment variable to allow for power to flow in two directions.

The case where there is power flow between SYS_1 and SYS_2 corresponds to an interconnection where part of the output of each system acts as an environment variable for the other, i.e., both ϕ_1 and ϕ_2 are non-trivial. In order to ensure that the interconnection is well-posed, i.e., the interconnected system avoids deadlock, environment variables should be partitioned into external and feedback parts. For subsystem SYS_1, external environment variables are g_1 and g_2, while the feedback environment is contactor C_4. In order for the system to be well-posed, decisions made by the controller for SYS_1 at step t must use the value of C_4 at the previous step $t-1$. A deadlock situation can occur between subsystems if this time shift is not accounted for, where each subsystem waits on an action from the other subsystem before it can make a move. See [67] for further discussion.

Due to the issue of well-posedness in the decentralized controller architecture, additional specifications are introduced in order to make the problem realizable. In order to successfully synthesize controllers for each subsystem, the following guarantees/assumptions are imposed.

- For SYS_2, if neither G_3 nor G_4 is healthy, then bus B_2 is powered. This is written as

$$\phi_r = \square\{g_3 = 1 \vee g_4 = 1 \vee b_2 = 1\}.$$

- For SYS_1, if neither G_1 nor G_2 is healthy, then power will be delivered through C_4. This is written as

$$\phi_l = \square\{g_1 = 0 \wedge g_2 = 0 \rightarrow (c_4 = -1)\}.$$

Because power must be able to be delivered to both subsystems, safety-critical buses are moved to those buses nearest the interface, i.e., to B_2 and B_3. In order to enforce well-posedness, specifications for the controller for SYS_1 involving C_4 are defined with additional next operators to implement a shift in time step. For the decentralized synthesis problem to be realizable, contactor delays are thus omitted in this problem formulation in order avoid conflicting specifications.

There are advantages and disadvantages in synthesizing controllers for a centralized versus distributed architectures. A centralized controller has complete knowledge of all components' statuses. It can anticipate the behavior of the entire environment, and thus control protocols can be less conservative (e.g. longer delays in contactor closing/opening times). For large-scale systems, though, a less-conservative controller comes at the cost of computational complexity. Distributed synthesis can be solved using less memory (due to the smaller number of components) and are thus more scalable to larger problems. However, due to lack of full information between subsystems, additional refinements are required at the interfaces. These refinements involve a more conservative contactor and bus configuration, (e.g, buses at the interface need to be powered more often). This is easily implementable for a master/slave architecture in which only a single-sided refinement is necessary. For the bi-directional distributed case in which refinements φ_1 and φ_2 are needed, well-posedness conditions further restrict the system. Contactor delays are no longer possible, and additional specifications are imposed on all components along the interfaces.

3.6.3 Timing Benchmarks

In this section we consider some timing benchmarks for the electric power system. For the topology in Figure 3.4, Table 3.2 lists the automaton size as well as total synthesis time while varying the number of clocks, as well as the discretization of clock "ticks." The first column indicates the number of clocks, or counters, used in the synthesis problem. Zero clocks refers to the the untimed synthesis problem in which all buses must always be powered. One clock refers to one essential bus that can never be unpowered for more than x ticks. The second column thus indicates the total time in which the essential bus can be unpowered. The higher the number, the more clock ticks must be incorporated into the synthesis problem. The third and fourth columns refer to the total automaton size (i.e., number of states) generated, as well as the total computation time (in seconds), respectively. All benchmark problems using < 2 GB memory.

Table 3.2: Synthesized Automaton Size

No. of Clocks	Clock "Ticks"	Aut. Size	Time [sec]
0	0	16	1
1	1	32	1.5
1	3	64	1.7
1	5	96	1.7
1	10	176	2.8
1	20	336	3.1
2	1	79	2
2	3	96	2
2	5	224	2.1
2	10	384	2.5
2	20	704	2.5
3	1	478	3.5
3	3	2858	7
3	5	7180	160
3	10	45492	1084
3	20	88604	4796
4	1	1798	7.2
4	3	22008	308
4	5	93386	4778

While GR(1) fragments of LTL can be synthesized in polynomial time to the number of states, the number of states grows exponentially with the number of clocks implemented. For small-sized problems, the difference in synthesis time is negligible. Once, however, the number of clocks used increases to 3, computation time jumps several orders of magnitude.

One thing to note is that the automaton size and times listed in Table 3.2 are worst-case scenarios. Using the specifications listed in Section 3.3, automaton size and time for 2, 3, or 4 clocks are identical to the one clock problem. This is because the first synthesis algorithm chooses the first feasible control protocol in which all buses are either powered or unpowered simultaneously. This is a feasible solution because in this formulation, there are no contactor delays. Thus, contactors can be commanded to open or close immediately. In order to calculate the numbers listed in Table 3.2, we included additional specifications requiring that the power status of buses must always eventually differ from each other. In other words, there must be transitions between states in which not all buses can all be simultaneously powered or unpowered.

3.7 Conclusions

This chapter demonstrates how text-based specifications can be converted into a temporal logic specification language using a representative single-line diagram as an example. Given a set topology for an electric power system, as seen in the single-line diagram from Figure 2.1 and a set of system requirements formalized in linear temporal logic, we automatically synthesize a control protocol for an electric power system on a more-electric aircraft. The resulting controller allows generators and APUs to connect and disconnect to buses through the closing and opening of contactors. The health status of each generator/APU is uncontrollable, and thus considered an environment action. The controller reacts to changes in the environment and is guaranteed, by construction, to satisfy the desired properties even in the presence generator failures. We synthesized a centralized controller where statuses of all components (generators, contactors, and buses) are known. We also created distributed and decentralized controllers by refining the overall system specifications. This refinement involves additional assumptions and guarantees between subsystem interfaces (i.e., specifications on the components that interact with other subsystems). For a distributed controller, we implemented a master/slave architecture where one subsystem has full authority for routing power to the other subsystem. In the decentralized controller design, we allow power exchange between two subsystems to flow in both directions, again refining the interface specifications.

The distributed and decentralized control protocols take less computational time to synthesize due to fewer components within each subsystems, and thus smaller state spaces. They are, however, more conservative than a centralized controller in terms of length of time non-essential buses are powered. Buses closer to the interfaces between subsystems are now powered for longer lengths of time in order to anticipate power requests from the other subsystem. From the basis of the work in this chapter, there are a number of potential directions for both practical and theoretical future work. We conclude the paper with a non-exhaustive list:

The number of components and specifications in the full scale electric power system represented in the single-line diagram creates a problem that is too computationally complex for current synthesis tools. There are two ways to address this challenge. The first is the method presented in this chapter via distributed controllers. The decomposition of overall system specifications into subsystem specifications, including interface assumptions and guarantees, is currently generated in an ad hoc manner. Future work will focus on automating the process of specification decomposition. The second approach to addressing the full scale problem may be in the use of linear temporal logic as a specification language. The specifications inherent in the electric power system problem concern safety requirements only (i.e., requirements only need be written with the temporal operator "always.") Thus, it does not utilize the full expressivity of LTL. It might be possible to solve larger scale problems by exploiting the case that specifications only deal with safety. Timing, and network transients, can be abstracted away to solve a series of static problems (See Section 4.4.1 for details on solving the untimed problem).

The timing specifications, (e.g., safety and contactor open/closing times) in the electric power system problem are addressed with the use of clocks by way of an additional counter variable. This discretization of time further adds to the difficulties arising from state space explosion. We are currently examining the use of timed verification and synthesis tools, in particular, UPPAAL-TIGA [11]. The efficiency of these timed verification tools, however, is still dependent on the number of clocks used in the model.

One open issue not addressed is what level of abstraction is needed for modeling, design, and specifications of an electric power system. Control of the power quality from generators is considered at a continuous level of abstraction. Load management and load shedding are considered at a discrete low-level of abstraction. Both of these problems, although at different levels of abstraction, should be interfaced with the primary distribution problem discussed in this chapter.

Chapter 4

Specification and Domain-Specific Languages

4.1 Overview

The development of a domain-specific language provides an easy interface between industry engineers knowledgeable in aircraft systems and the methods/tools used by computer scientists and software engineers. In this chapter we describe a domain-specific language for aircraft electric power systems as well as an automatic specification generator available within `TuLiP`. The language combines tools already in existence: visual programs for single-line diagrams, which engineers are familiar with, and primitives, which provide a more formal structure to specifications.

The rest of the chapter is structured as follows: Section 4.2 explains the input files and underlying graph structure used to convert a toplogy into specifications, while Section 4.3 explains how the specifications can be represented in a domain-specific language by a set of “primitives." Section 4.4 describes the specification conversion tool *AES2specgen* and provides some problem complexity benchmarks. Section 4.5 introduces an extension to the domain-specific language within an engineering framework of a sequence diagram. Section 4.6 discusses how to specify requirements using timed temporal logics.

4.2 Input Files

Figure 4.1 provides a flow diagram for the automatic specification generation procedure. Three sets of inputs must be provided from the information given by the diagram (connectivity) and components (attributes). First, the single-line diagram, a visual representation, can be converted

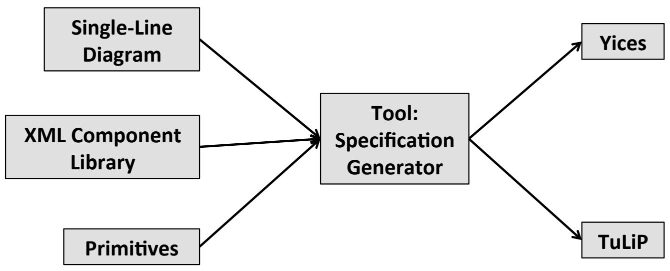


Figure 4.1: Architecture for the specification generator. The problem description includes three inputs: a single-line diagram, a component library, and a set of primitive specifications. The output is a set of formal specifications compatible with Yices (a SAT solver) or `TuLiP` (a reactive synthesis tool).

```
<contactor>           <bus>
      <failure>            <failure>
         1e-3                 1e-3
      </failure>           </failure>
      <opentime>           <essential>
         15                   true
      </opentime>          </essential>
      <closetime>     </bus>
         20
      </closetime>
</contactor>
```

Figure 4.2: A sample XML component library file for contactor and bus components that have attributes of *opentime*, *closetime*, and *essential*.

into a graph data structure, where contactors are edges, and all other components represent nodes[1]. Let $G = (V, E)$ be a graph of the electric power system, with $V = \{v_1, v_2, \ldots, v_n\}$ containing all components consisting of generators, buses, and rectifier units. Loads, transformers and batteries are not implemented in our current formulation but can be easily integrated. The set of edges $E = \{e_1, e_2, \ldots, e_m\}$ then contains all contactors (as well as solid wire links between components). The adjacency matrix $\mathbf{A}_{ij}$ is a square adjacency matrix whose diagonal entries are zeros, and whose non-diagonal entries are ones or zeros depending on whether a contactor (or solid link) exists between vertices.

The second set of information is an XML file containing component attributes. Consider a simple case in which the XML file contains a listing for each type of component: contactor, generator, rectifier unit, and bus. Figure 4.2 depicts a example of an XML file for contactor and bus components. Each component has an attribute of *name* and *failure probability*, i.e., the probability each component has of failing over a certain number of operational hours. A failure probability of 10^{-3}, for example, means that the component may fail once over the course of 10^3 operating hours. Buses have an

[1]Graphical tools exist which can convert visual diagrams into XML code. We begin with the assumption that such a conversion has been implemented and the XML file is parsed into an adjacency matrix.

additional Boolean attribute of *essential*, as well as an attribute *time* which states how long the bus may be unpowered for during a flight. In addition, contactors have attributes *opentime* and *closetime*, denoting the time it takes to physically open or close the contactor.

The third input is a set of primitives used to represent the high-level requirements that specify the desired behavior of the system. The use of primitives is described in the next section.

4.3 Specifications and Primitives

Given the topology of an electric power system and component attributes, the main design problem is determining all correct configuration of contactors for all flight conditions and faults that may occur. As previously discussed in Section 3.2, we again reference some common or standard specifications relevant to the electric power system problem, and describe how these specifications may be written using a set of primitives.

Environment Assumptions: The overall system safety level determines the possible combinations of failures which may occur. Consider the case where generators and rectifier units are environment variables, i.e., uncontrolled. Because each component has an individual failure probability, we can determine how many components may fail at a single instance (while satisfying the system safety rating), and produce a set of valid environment assumptions. Let $\mathcal{G}$ and $\mathcal{R}$ be the sets of all generators and rectifier units, respectively. In the environment primitive (in which only generators and rectifier units are uncontrolled), the first input is a system safety level, followed by all subsets of components that are uncontrolled. This can be written as $\mathbf{env}(10^{-x}, \mathcal{G}_e, \mathcal{R}_e)$, where x is the failure rating, $\mathcal{G}_e \subseteq \mathcal{G}$ and $\mathcal{R}_e \subseteq \mathcal{R}$.

No-paralleling of AC sources: One common specification may be that no two asynchronous AC sources can power a bus simultaneously. A non-paralleling primitive thus has inputs of any subset of $\mathcal{G}$. This can be written as $\mathbf{noparallel}(\mathcal{G}_p)$, where $\mathcal{G}_p \subseteq \mathcal{G}$.

Essential buses: Essential buses supply power to safety-critical subsystems and loads, and thus must be powered at all times. Let the set of all buses be $\mathcal{B}$. An essential bus primitive can input any subset of $\mathcal{B}$. This is written as $\mathbf{essbus}(\mathcal{B}_e)$, where $\mathcal{B}_e \subseteq \mathcal{B}$.

Bus unpowered time: Non-essential buses supply power to loads and subsystems which can tolerate loss of power for up to a certain period of time. This time information is captured from the component library, which contains the maximum unpowered time a bus may be able to tolerate. Thus the primitive may be written $\mathbf{buspower}(\mathcal{B}_s)$, where $\mathcal{B}_s \subseteq \mathcal{B}$, and $\mathcal{B}_e \cap \mathcal{B}_s = \emptyset$.

Disconnect with unhealthy: When certain components (generators or rectifier units) become unhealthy, they must be disconnected from the system for safety reasons, i.e., the contactor connecting that component to other buses or components, needs to open. A disconnect primitive can take as input the union of subsets of $\mathcal{G}$ and $\mathcal{R}$. This primitive is written as **disconnect**$(\mathcal{G}_d \cup \mathcal{R}_d)$, where $\mathcal{G}_d \subseteq \mathcal{G}$ and $\mathcal{R}_d \subseteq \mathcal{R}$.

4.4 Tool Integration

The electric power system can be abstracted into different model views. We consider the following four views: untimed, discrete variables; discrete-time, discrete variables; continuous-time, discrete variables; and continuous-time, continuous variables. A domain-specific language can facilitate consistency between these views by providing a unifying framework for constituent elements. The following section discusses how the design problem can be automatically synthesized within the model view of discrete variables with no time or discrete-time. Our tool, which converts the above primitives into a set of specifications, is written using Python, with the additional use of the software package NetworkX to study the underlying graph structure. The sourcecode is included in `TuLiP` version 0.4a (and above) under tools/AES directory.[2]

4.4.1 Untimed: SAT Solver (Yices)

Consider the case in which timing specifications are ignored. Generators and rectifier units can either be healthy or unhealthy, contactors may either be open or closed, and buses can either be powered or unpowered. The synthesis problem reduces to a Boolean satisfiability problem. For each set of environment scenarios, a specific configuration of contactors satisfies all system requirements. Our current tool converts the set of primitives to a format compatible with the solver Yices [26][3].

Based on the graph G derived from the single-line diagram, we automatically instantiate components, such that

$$(\textbf{define}\ g :: \textbf{bool})$$

$$(\textbf{define}\ r :: \textbf{bool})$$

$$(\textbf{define}\ b :: \textbf{bool})$$

$$(\textbf{define}\ c :: \textbf{bool})$$

[2]Click here for AES Directory

[3]To be precise, Yices is an SMT solver which can also be used as a SAT solver.

for all $g \in \mathcal{G}$, $r \in \mathcal{R}$, $b \in \mathcal{B}$, and $c \in \mathcal{C}$, where $\mathcal{C}$ is the set of all contactors.

Because the SAT solver searches for a different solution for each configuration of environment behaviors, we generate all allowable environment sets, given the system safety level, and thus generate a set of environment assertions. Let $\mathcal{P} \subseteq \mathcal{G} \times \mathcal{R}$ be the set of environment variables. Environment assumptions can be written as

$$(\textbf{assert}\ (=\ p\ \textbf{[status]}))$$

for all $p \in \mathcal{P}$, and **[status]** is either **true** or **false**, denoting a healthy or unhealthy component.

To avoid paralleling, the tool takes all pairs of generators input from the primitive and searches for all simple paths between items in each pair. For all simple paths between generator pairs, we disallow all contactors within each path to be closed at the same time. Consider, for example, In Figure 4.3, the set of contactors between g_1 and g_2 that constitute a live path are c_1, c_2, and c_3. For this case, the non-paralleling specification output would be

$$\neg \left\{ \bigwedge_{i=1}^{3} c_i = 1 \right\},$$

where 1 denotes a closed contactor [4].

More generally, define $\mathcal{X}_{ij}$ to be the set of all paths between two components $\mathcal{X}_i$ and $\mathcal{X}_j$. Each path $x_k \in \mathcal{X}$ consists of some number of components such that each x_k contains $\{x_k^1, \ldots, x_i^{n_k}$, for n_k components (not including $\mathcal{X}_i$ or $\mathcal{X}_j$. To disallow non-paralleling between any two generators, the specification is written as follows

$$\bigwedge_{x_k \in \mathcal{X}_{ij}} \left\{ \neg \left(\bigwedge_{c_k \in x_k} c_k = 1 \right) \right\},\ \forall \mathcal{X}_i, \mathcal{X}_j \in \mathcal{G}. \tag{4.1}$$

In order to assert that a bus must always remain powered, we first output a set of specifications which determine under what conditions a bus is powered or unpowered. We first search for all paths between each element input into the primitive, and output all path configurations that would cause the bus to be powered. This means all other buses, generators, and contactors in said path must be powered, healthy, and closed (respectively). If, in none of the paths, the conditions for a powered bus are met, then the bus is unpowered. Consider again the simple example from Figure 4.3. The

[4]For ease of notation, the remaining Yices specifications will be written standard propositional form, while the actual format for the tool differs slightly.

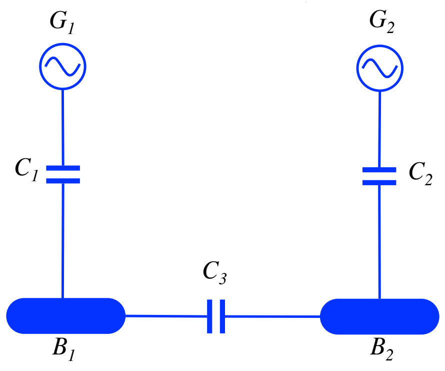


Figure 4.3: Simplified version of a the single-line diagram. Two AC generators connect to two buses via three contactors.

two output specifications for bus b_1 when it is powered would be

$$((g_1 = 1) \wedge (c_1 = 1)) \rightarrow (b_1 = 1),$$

and

$$((g_2 = 1) \wedge (c_2 = 1) \wedge (b_2 = 1) \wedge (c_3 = 1)) \rightarrow (b_1 = 1).$$

If neither of the two above conditions hold, then b_1 is unpowered. This is written as

$$\{\neg([(g_1 = 1) \wedge (c_1 = 1)] \vee [((g_2 = 1) \wedge (c_2 = 1) \wedge (b_2 = 1) \wedge (c_3 = 1))]) \rightarrow (b_1 = 0)\}.$$

More generally, consider all paths $\mathcal{X}_{ij}$ where $\mathcal{X}_i \in \mathcal{G}$ and $\mathcal{X}_j \in \mathcal{B}$. Specifications for bus power status can be written

$$\bigwedge_{\mathcal{X}_i \in \mathcal{G}, \mathcal{X}_j \in \mathcal{B}} \left\{ (\bigwedge_{x_k \in \mathcal{X}_{ij}} x_k = 1) \rightarrow (\mathcal{X}_j = 1) \right\}, \tag{4.2}$$

$$\bigvee_{\mathcal{X}_i \in \mathcal{G}, \mathcal{X}_j \in \mathcal{B}} \left\{ \neg(\bigwedge_{x_k \in \mathcal{X}_{ij}} x_k = 1) \rightarrow (\mathcal{X}_j = 0) \right\}. \tag{4.3}$$

Therefore, to assert that all buses are always powered, we then write

$$\bigwedge_{b \in \mathcal{B}} (b = 1). \tag{4.4}$$

To disconnect an unhealthy generator or rectifier unit, we search the graph for adjacent nodes,

```
(= b5 true)      (= g2 false)
(= b6 true)      (= c27 false)
(= b7 true)      (= g3 true)
(= c56 true)     (= c38 false)
(= c67 true)     …
(= c78 true)     (= r10 true)
(= c89 true)     (= c510 true)
(= c1516 true)   (= r11 true)
(= c1617 true)   (= c611 true)
```

Figure 4.4: A sample output from Yices for a single environment configuration.

and assert an implication that if a component is unhealthy, the neighboring contactor must be open (take a value of 0). This is written as

$$\bigwedge_{p\in\mathcal{P}} \left\{ (p = 0) \rightarrow (\bigwedge_{c_p} c_p = 0) \right\}. \tag{4.5}$$

for all $p \in \mathcal{P}$, and $c_p \subseteq \mathcal{C}$ is the subset of contactors connecting component p to an adjacent component.

From the above set of specifications, Yices solves a satisfiability problem and determines the configuration for all contactors, for each environment configuration. Figure 4.4 shows an portion of the output from Yices. Thus a controller from Yices is a set of contactor configurations for each environment.

4.4.2 Timed: **`TuLiP`**

The benefits of an untimed model view is reducing the synthesis problem to a satisfiability problem, in which case a SAT solver may be used, the complexity of which is less than that for synthesis algorithms. More realistic design problems in the electric power system domain require timed specifications. We therefore incorporate formats compatible with `TuLiP` as well as Yices in the translation from primitives to specifications. `TuLiP` uses a model view that includes discrete-time and discrete variables; specifications are written in linear temporal logic (LTL).

We visit the primitives described in Section 4.3, and begin by instantiating all variables (controlled and uncontrolled). Variables are again discrete and Boolean. For all environment (uncontrolled) components, instantiations are written as

$$\textbf{env\_vars}[p] = [0, 1], \tag{4.6}$$

for all $p \in \mathcal{P}$. For all controlled variables, instantiations are written as

$$\textbf{disc\_sys\_vars}[s] = [0, 1], \tag{4.7}$$

for all $s \in \mathcal{B} \cup \mathcal{C}$.

To specify the allowable environment assumptions, we again take all possible allowable sets of failures which can occur given the system failure probability. Assume the failure rate for each component is independent. Then, all combinations of failures that have a failure probability greater than the overall system level must be accounted for. The output specification, then, uses an always ($\Box$) operator alongside a string of disjunctions.

Consider a simple example with two environment variables g_1 and g_2 that can take a value of 0 (unhealthy) or 1 (healthy). Suppose the overall system safety level is 10^{-5}, and each generator has a failure probability of 10^{-3}. The probability that both generators are unhealthy becomes 10^{-6}, which is smaller than 10^{-5}. Acceptable environment behaviors include three possibilities: $g_1 = 1, g_2 = 1$; $g_1 = 1, g_2 = 0$; and $g_1 = 0, g_2 = 1$. Once the tool calculates this set of allowable environments, the `TuLiP` compatible specification output becomes

$$\textbf{assumptions} = \Box((g_1 = 1 \wedge g_2 = 1) \vee (g_1 = 1 \wedge g_2 = 0) \vee (g_1 = 0 \wedge g_2 = 1)).$$

More formally, let $\mathcal{I}$ be an index set enumerating the set of environment variables. For each environment variable p_i, $i \in \mathcal{I}$, let f_i be its probability of failure in a given time interval T. Let r be the overall reliability level the system has to achieve, that is, the probability of the overall system failure within the interval T should be less than r. Assuming independence of component failures, the overall reliability level of an aircraft determines the allowable environment assumptions by providing a bound on the number of simultaneous component failures allowed. Whenever the product of components' probability of failure (p_i) is more than the reliability level r, the control must ensure the requirements are satisfied. Denote a single configuration of the environment (i.e., an environment state) by $\mathbf{e}$. For a given subset $\mathcal{I}' \subseteq \mathcal{I}$ of the environment variables, we define $\mathbf{e}_{\mathcal{I}'} = (p_1, \ldots, p_{|\mathcal{I}|})$, where $p_i = 0$ (unhealthy) if $i \in \mathcal{I}'$; and $e_i = 1$ (healthy) otherwise. We can then enumerate all allowable environment configurations based on the required reliability level, as

$$\mathcal{E} = \left\{ \mathbf{e}_{\mathcal{I}'} | \mathcal{I}' \subseteq \mathcal{I} \text{ s.t. } \prod_{j \in \mathcal{I}'} p_j \geq r \right\}. \tag{4.8}$$

With this definitions, an environment assumption can be written in LTL as

$$\textbf{assumptions} = \Box(\mathbf{e} \in \mathcal{E}). \tag{4.9}$$

The non-paralleling specification disallows all contactors to be closed if they are within a path connecting two AC sources. In LTL, this is implemented using a never operator ($\Box\neg$). Using the example, from Figure 4.3, a non-paralleling specification would be of the form

$$\textbf{guarantees} = \Box\neg((c_1 = 1) \wedge (c_2 = 1) \wedge (c_3 = 1)).$$

We thus explicitly enumerate and disallow all bad configurations. Let $x_{i,j}$ represent the set of components along a path between generators p_i, p_j, for $p_i, p_j \in \mathcal{G}$ and $i \neq j$. We disallow configurations in which all contactors $c \in x_{i,j}$ create a live path. These specifications are written as

$$\textbf{guarantees} = \Box \bigwedge_{p_i, p_j \in \mathcal{G}} \left\{ \neg \bigwedge_{c \in x_{i,j}} (c = 1) \right\}. \tag{4.10}$$

The primitives for bus power and essential bus power first create a set of discrete properties that specify the conditions for when a bus is powered. Just as in the case using Yices, we find all paths from a bus to a generator, and list the component configurations needed for a bus to receive power. In Figure 4.3, for example, there are two properties for which bus b_1 can be powered, written as

$$\begin{aligned} \textbf{disc\_props[d1]} &= (g_1 = 1) \wedge (c_1 = 1), \\ \textbf{disc\_props[d2]} &= (g_2 = 1) \wedge (c_2 = 1) \wedge (b_2 = 1) \wedge (c_3 = 1). \end{aligned}$$

Then, specifications output when bus b_1 is powered are

$$\begin{aligned} \textbf{guarantees} &= \Box((d1) \rightarrow (b_1 = 1)), \\ \textbf{guarantees} &= \Box((d2) \rightarrow (b_1 = 1)). \end{aligned}$$

If neither proposition is true, b_1 is unpowered, written as

$$\textbf{guarantees} = \Box(\neg((d1) \vee (d1)) \rightarrow (b_1 = 0)).$$

More formally, an AC bus can only be powered if there exists a *live path* (i.e., all contactors closed along a path) that connects the bus to a healthy AC generator or a healthy APU. Similarly, a DC bus can only be powered if there exists a live path that connects it to a healthy rectifier unit, which itself is connected to a powered AC bus. Let $x_{i,b}$ denote the set of all components (i.e., contactors and buses) along a path between bus b and environment variable p_i for $i \in \mathcal{I}$, excluding b and p_i. Furthermore, let $\mathcal{G} \subseteq \mathcal{P}$ and $\mathcal{R} \subseteq \mathcal{P}$ represent the sets of generators and rectifier units. AC bus b is powered if there exists a live path between B and p_i for $p_i \in \mathcal{G}$, written as

$$\textbf{guarantees} = \Box \left\{ \bigvee_{p_i \in \mathcal{G}} \left((p_i = 1) \wedge \bigwedge_{x \in x_{i,B}} (x = 1) \right) \rightarrow (b = 1) \right\}. \tag{4.11}$$

If there exists no live path between b and a generator p_i for $p_i \in \mathcal{G}$, then b will be unpowered

$$\textbf{guarantees} = \Box \left\{ \neg \bigvee_{p_i \in \mathcal{G}} \left((p_i = 1) \wedge \bigwedge_{x \in x_{i,B}} (x = 1) \right) \rightarrow (b = 0) \right\}. \tag{4.12}$$

A similar set of specifications for DC buses holds in which environment variables p_i spans $p_i \in \mathcal{R}$.

Once these specifications are written, timing on buses can be introduced. If a bus is an essential bus, then another specification guarantees that the bus always remains powered. This is written as **guarantees** $= \Box(b = 1)$, for all $b \in \mathcal{B}_e$. For non-essential buses, we impose a maximum allowable time for which the bus may be unpowered. This value is taken from the XML component library file.

For each non-essential bus $b \in \mathcal{B}_s$, we introduce a unique counter t_k. We discretize each time step to take δ time. If a bus is unpowered, at the next step the counter will increment by δ. Counters are also bounded by a set maximum time limit. If the bus is powered, at the next step the counter will reset to 0. These specifications are output as

$$\textbf{guarantees} = \Box((b_k = 0) \rightarrow (\bigcirc(t_k) = t_k + \delta)), \tag{4.13}$$

$$\textbf{guarantees} = \Box((b_k = 1) \rightarrow (\bigcirc(t_k) = 0)), \tag{4.14}$$

for all $b_k \in \mathcal{B}_s$. Then, we limit the number of "ticks" t_k can increment to $\frac{T}{\delta}$ steps. This specification is output as

$$\textbf{guarantees} = \Box(t_k \leq \frac{T}{\delta}). \tag{4.15}$$

The final set of specifications involve removing unhealthy components from the overall system. To disconnect an unhealthy generator or rectifier unit, we use an implication. For all environment variables p_i, for $i \in \{1, \dots, n_e\}$, if any component becomes unhealthy then the contactor connecting p_i to an adjacent component must open. This is written as **guarantees** $= \square((p_i = 0) \rightarrow (\wedge_{j \in N_i}(c_{ij}) = 0))$, where N_i denotes the set of vertices adjacent to vertex i.

The final set of specifications involve disconnecting unhealthy components from the overall system. Let $\mathcal{N}(e_i)$ represent the set of contactors directly connected, or neighboring, environment variable p_i for $i \in \mathcal{I}$. We write the specifications to disconnect all unhealthy sources as

$$\textbf{guarantees} = \square \bigwedge_{i \in \mathcal{I}} \left\{ (p_i = 0) \rightarrow \bigwedge_{c \in \mathcal{N}(p_i)} (c = 0) \right\}. \tag{4.16}$$

These specifications are input into `TuLiP`, which interfaces with a digital design synthesis tool implemented in JTLV [74]. If the specification is realizable, `TuLiP` outputs a finite-state automaton that represents the control protocol. Figure 2.3 shows a portion of a sample finite-state automaton.

Remark 3 *The specifications within this section differ from the specification format used in Chapter 3. While the specifications from the previous chapter are generalizable, for the purposes of a domain-specific language we utilize the topology's underlying graph structure (connectivity) in order to formulate specifications using "live" paths. For problems of this scale, the computational time of either formulation is comparable.*

4.4.3 Benchmarks

In this section we discuss some results for several electric power system topologies using both Yices and `TuLiP`. For ease of comparison, consider the base topology shown in Figure 4.5 that includes both AC and DC components. Each vertical set of components (generator, DC bus, rectifier unit, AC bus, and two contactors) form a base unit. Units may be connected together by contactors located between AC and DC buses. We examine the results for topologies with varying numbers of units.

Table 4.1 lists the amount of time our tool takes to convert a set of primitives for a given base topology into formal specifications. Columns 2 and 3 show the size of the beginning graph, while column 4 compares the difference in times between converting specifications into a Yices or `TuLiP`-compatible format. The difference in conversion times is insignificant for smaller sized graphs. The Yices conversion takes more time due to the increase of allowable environment configurations. Because we solve a series of static problems, the tool must write a set of specifications for each of the environment scenarios. One thing to note is that the topologies we explore have many symmetries in

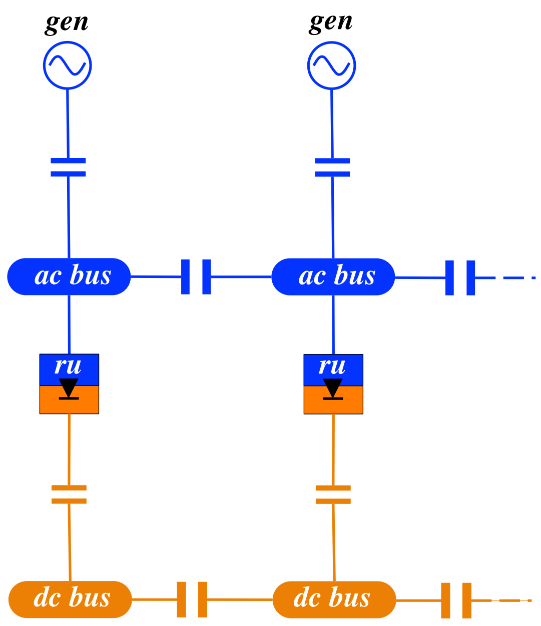


Figure 4.5: The base topology used to discuss the domain-specific language and conversion tool. Each base unit consists of a generator, DC bus, rectifier unit, and AC bus. Units are connected to each other by contactors between buses. More units are connected on the right (represented by the dotted wire/line.)

the graph. Therefore, not all environment conditions need to be enumerated, e.g., an engine failure on the left side can be treated as similar to an engine failure on the right side.

Given the set of automatically generated specifications, Table 4.2 compares the time it takes for Yices and `TuLiP` to solve/synthesize a controller for a given topology. Column 2 lists the total number of environment configurations, i.e., the number of static problems Yices must solve. Then, Column 3 shows the time for Yices to solve a single environment configuration, as well as the time it takes for `TuLiP` to solve the full synthesis problem. Columns 3 and 4 show that solving a series of satisfiability problems is much time and memory efficient than using a synthesis tool. Increasing the topology from four to five base units dramatically increases the computation time. In addition, we applied the conversion tool to the single-line diagram topology from Figure 2.1. Column 5 shows the number of states output by Yices and `TuLiP`. While the number of environment configurations is large, generation of all other primitives requires only 10 seconds. For one environment configuration, Yices takes 0.9 seconds and 39MB of memory to solve. This shows that the use of our conversion tool can be applicable to industrial-sized problems for untimed problems.

The size of the Yices controllers is the number of different environment configurations. `TuLiP` synthesized controllers with four and five base units have 256 and 1024 states, respectively. While

Table 4.1: Specification Conversion Time for Yices (Y) and `TuLiP` (T) [time in seconds]

Base Units	Nodes	Edges	Conversion Time (Y/T)
4	16	18	.13/.11
5	20	23	.25/.26
10	40	48	24/18
12	48	58	141/111
15	60	73	1634/1205

Table 4.2: Comparison of Synthesis Time for Yices (Y) and `TuLiP` (T). [time in seconds]

Base Units	Yices Env.	Time(Y/T)	Mem. (Y/T)	Output Size (Y/T)
4	25	.25/10.7	25MB/215MB	400/256
5	36	.82/1015	36MB/16GB	720/1022
10	121	205.7/–	53MB/–	4840/–
12	169	1410/–	158MB/–	8112/–
15	256	62208/–	1.2GB/–	15360/–

the use of a SAT solver is seemingly more advantageous than that of a synthesis tool, the range of problems which the SAT solver can handle is limited to those with untimed specifications. Alternatively, specifications written in linear temporal logic and synthesized using `TuLiP` can incorporate discrete-time specifications. Thus, we can automatically generate control protocols that can not only solve static configurations, but reason about how to transition between environment configurations through a series of contactor switches.

4.5 Broadening the Domain-Specific Language

The primitives discussed in Section 4.3 encompass a standard set of high-level specifications found in aircraft electric power systems. While it provides an interface to temporal logic specifications, as yet the domain-specific language does not offer much flexibility for an engineer to design a system. While domain-specific languages must be kept structured (i.e., limited to specific tasks), the formulation in the above section can be extended. In the following we describe two additions to the *AES2spec* tool.

4.5.1 Exceptions and Nominal Cases

Given a known set of environmental conditions or assumptions, such as those discussed in Section 3.2, all specifications must be satisfied in order for the system to function correctly. We consider such a flight to be operating in a "nominal." In cases, however, in which the flight were to operate under an additional "degraded" mode, then not all specifications need be satisfied. For example, a nominal condition may be that half of all generators are available and healthy. A degraded condition may be that only one generator is available to power the entire aircraft, in which case not all buses need to satisfy the condition of always being powered. Thus, both nominal and exception cases can be introduced into the domain-specific language.

4.5.1.1 Primitives

Assume that the XML component library contains information on what flight modes are possible, such that set of modes $\mathcal{M} = \{nom, m_1, \ldots, m_i\}$, where nom is the nominal flight mode, and i indexes all other "degraded" modes. The following primitives can be modified such that:

No-paralleling of AC sources: For a nominal condition in which no live path can exist between two AC sources, the primitive for non-paralleling has an input of mode $m \in \mathcal{M}$ and a subset of generators, written as **noparallel**$(m, \mathcal{G}_p)$, where $\mathcal{G}_m \subseteq \mathcal{G}$. The LTL specification is then translated by introducing the mode condition into the left side of the implication, such that

$$\bigwedge_{m \in \mathcal{M}} \left\{ \Box(m \to \neg(\bigwedge_{\mathcal{G}_m} paths)) \right\}, \tag{4.17}$$

where $paths$ represent the conjunction of all contactors located between two elements in $\mathcal{G}_m$ that could form to create a live path.

For an aircraft in nominal mode, for example, equation 4.10 becomes

$$\Box \{(m = nom) \to \neg((c_1 = 1) \wedge (c_2 = 1) \wedge (c_3 = 1))\}.$$

Essential buses: All essential buses must always be powered in a nominal condition, but may not necessarily be enforced if in another mode. The primitive can thus be modified with two inputs, mode m and subset of buses $\mathcal{B}_m \subseteq \mathcal{B}$ that must always remain powered: **essbus**$(m, \mathcal{B}_m)$. The LTL specification becomes

$$\bigwedge_{m \in \mathcal{M}} \left\{ \Box((mode = m) \to \bigwedge_{b \in \mathcal{B}_m} (b = 1))) \right\}. \tag{4.18}$$

Bus unpowered time: The amount of time a bus remains unpowered can also depend on the flight mode. This duration may be extended in degraded conditions. The primitive becomes **buspower**$(m, \mathcal{B}'_m)$ where $\mathcal{B}'_m \subseteq \mathcal{B}$. The LTL specification can be written as

$$\bigwedge_{m\in\mathcal{M}}\left\{\bigwedge_{b\in\mathcal{B}'_m}(\Box((mode=m)\wedge(b=0))\rightarrow(\bigcirc(t_b=t_b+1)))\right\}, \tag{4.19}$$

$$\bigwedge_{m\in\mathcal{M}}\left\{\bigwedge_{b\in\mathcal{B}'_m}(\Box((mode=m)\wedge(b=1))\rightarrow(\bigcirc(t_b=0)))\right\}, \tag{4.20}$$

$$\bigwedge_{m\in\mathcal{M}}\left\{\bigwedge_{b\in\mathcal{B}'_m}\Box(t_b\leq\frac{T_m}{\delta})\right\}, \tag{4.21}$$

where t_b is the clock variable for bus b and T_m is the maximum time which a bus can be unpowered in flight mode m.

4.5.2 Sequence Diagrams

The second addition to the domain-specific language is the ability to integrate scenario-based requirements. Sequence diagrams are a part of the SysML modeling language, and are used to define sequences of events. Diagrams communicate messages between ”actors” and in what particular order they must occur. Figure 4.6 depicts an example sequence diagram in which four actors interact within some aircraft system. The pilot, supervisory control, plant, and display send messages to each other along horizontal lines, while the vertical axis represents the progression of time. Solid lines represent messages that must occur, while dotted lines represent messages that may occur. The length of the vertical green boxes signifies time, but is not meant to represent an exact duration.

4.5.2.1 Live Sequence Charts

Sequence diagrams lack semantics, which makes integration with formal methods tools difficult. Kugler and Harel [49] have provided semantics for a variant of sequence diagrams, called Live Sequence Charts. With these imposed semantics, the behaviors of live sequence charts can be captured using temporal logic. In particular, two types of charts can be used: existential (sequence of events must happen at least once) and universal (sequence of events must always happen). Existential charts can be expressed in CTL, while universal charts are expressible in LTL. The live sequence chart to temporal logic conversion can be shown to generate a formula that is at most quadratic in the size of the chart.

A further extension of live sequence charts is shown in [56] that incorporates assume-guarantee scenarios. Furthermore, this extension of sequence charts extends syntax and semantics to distin-

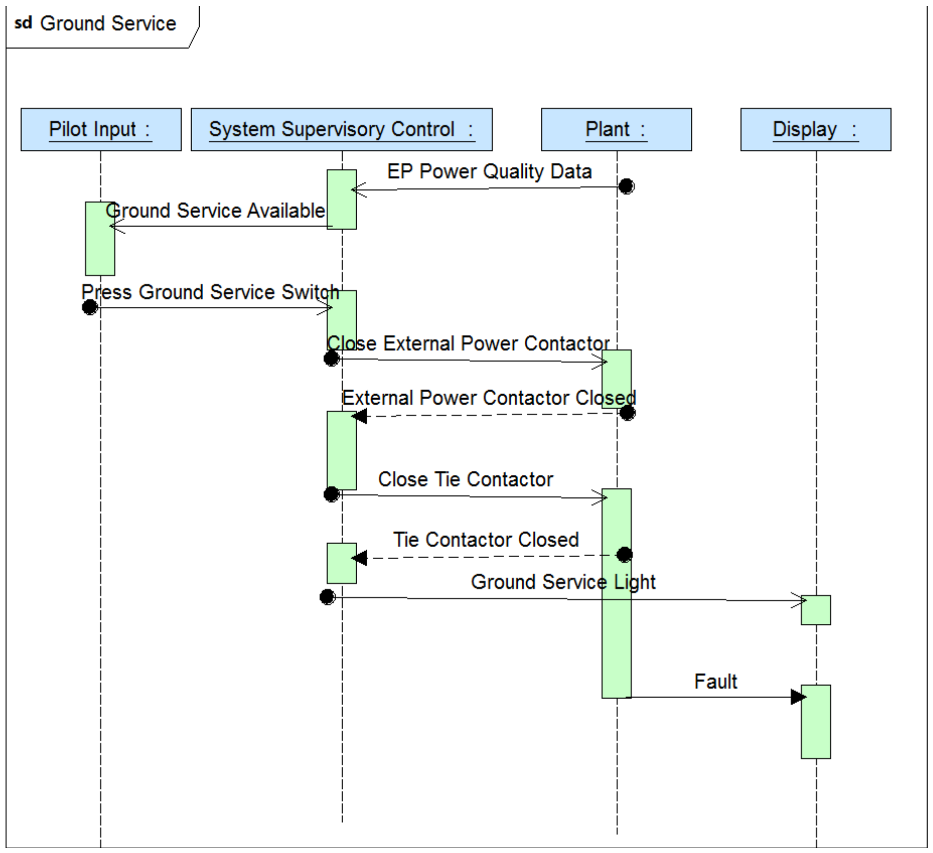


Figure 4.6: An example sequence diagram with pilot, control, plant, and display actors. Actors send messages (horizontally) along vertical lines (representing the dimension of time).

guish between system and environment entities, and supports conversion to GR(1) fragments of LTL. In this next sections we follow the exposition of [50] and [56] in describing the live sequence chart to temporal logic conversion.

4.5.2.2 Live Sequence Chart Semantics

Consider the set of live sequence charts depicted in Figure 4.7. Vertical lines (in which time progresses downward) are *instances* that represent an interacting agent. Agents are controlled either by the system or environment. *Messages* are horizontal lines that represent calls between agents. A message is a *system message* if it is sent from an instance controlled by the system, and is an *environment message* if sent from an environment instance. The chart defines a partial order on messages induced by the vertical ordering of messages sent and received along instances.

In the top left chart of Figure 4.7 (`InsertCoins`), `user` is an environment instance, while `panel` and `cashier` are system instances. Messages `insertCoin` and `incCoins` are environment and system messages, respectively.

A *system cut* represents the current state of an live sequence chart, signifying the progress of events along instances. The *minimal cut* is the state at which the chart is closed. A message is *enabled* in a cut of the chart if it appears immediately after the cut in the induced partial order. A message is *violating* in a cut if it appears in the chart but is not enabled.

Messages, depicted as either red or blue lines, can be hot or cold. A hot enabled message must eventually occur. A cold enabled message could eventually occur. A cut may be hot if at least one of the enabled system messages is hot, otherwise it is cold. The chart progresses to the next cut when an enabled message occurs. If a violating message occurs, transitions depend on the temperature of the cut. If the cut is cold, the chart closes gracefully. If a cut is hot, then this represents a violation of requirements.

Conditions may also be hot or cold, and are evaluated as soon as they are enabled. A hot enabled condition must be evaluated to be true, while a cold enabled condition may or may not be evaluated to true. The chart progresses to the next cut if a condition is evaluated to be true. If a condition is evaluated false and the condition is cold, the chart closes gracefully If the condition is hot, this represents a violation of requirements.

System messages can be either *execution* or *monitoring* (solid and dashed lines, respectively). All environment messages are monitoring. A chart is *active* if the current cut has an enabled system message.

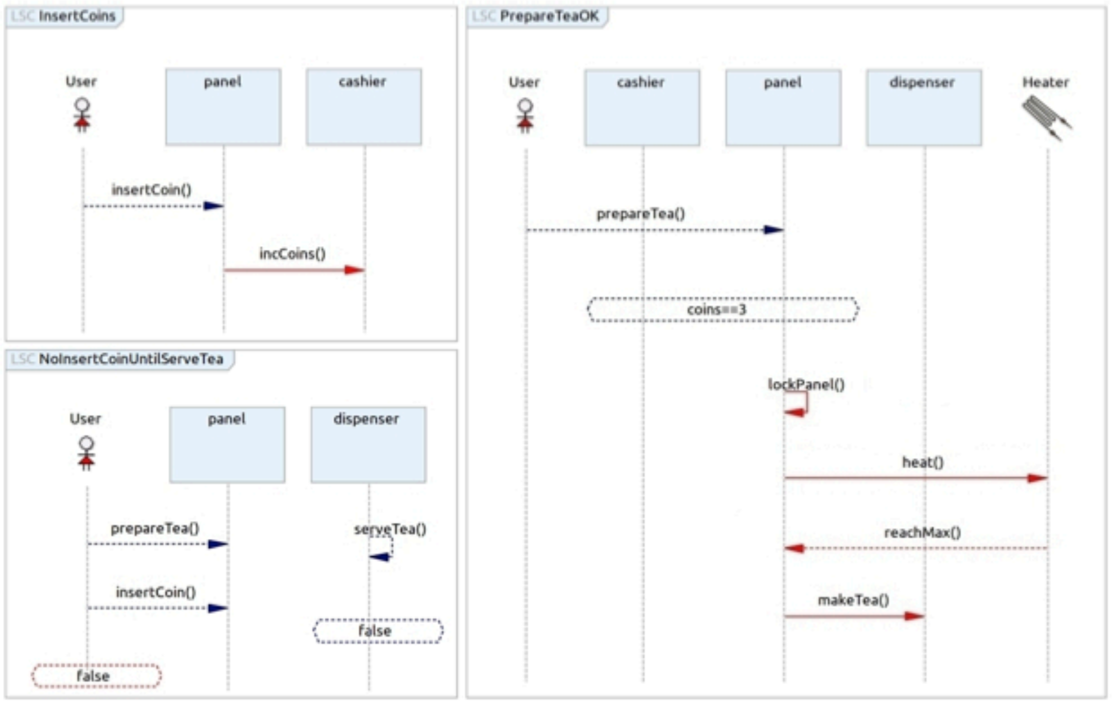


Figure 4.7: Three assume-guarantee scenarios for an example vending machine specification. System entities include panel, cashier, and dispenser. Environment entities include user and heater. Figure from [56].

4.5.3 LTL-Live Sequence Chart Semantics

Given a set of live sequence charts $\mathcal{L} = \{\mathcal{L}_1, \ldots, \mathcal{L}_n\}$, let $\mathcal{M}^s(\mathcal{L})$ and $\mathcal{M}^e(\mathcal{L})$ be defined as the set of system and environment messages that can be sent, respectively. Additionally, let the two sets be disjoint, such that $\mathcal{M}^s(\mathcal{L}) \cap \mathcal{M}^e(\mathcal{L}) = 0$. the following variables are used to define a formal model:

- m_e is an environment message variable (input) over the domain of all messages the environment can send in $\mathcal{L}$. An additional *no_op* value is included for doing nothing. For every environment message $m \in \mathcal{M}^e(\mathcal{L}) \cup \{no\_op\}$ sent, a synthesized strategy will know how to react.

- m_s is a system message variable (output) over the domain of all messages the system can send in $\mathcal{L}$. An additional *no_op* is included for doing nothing. For every state, the synthesized strategy knows which system message $m \in \mathcal{M}^s(\mathcal{L}) \cup \{\text{“}no\_op\text{”}\}$ to send.

- $\{l_1, \ldots, l_n\}$ is the set of output cut variables. Every l_i encodes a *cut automaton* for live sequence chart $\mathcal{L}_i$. The domain of l_i, denoted $dom(l_i)$ consists of all possible $\mathcal{L}_i$ cuts, including the minimal cut (denoted by *MIN*). Two additional sink values of VIO^s and VIO^e represent hot violation of system guarantees and environment assumptions.

The minimal cut value *MIN* indicates a closed chart. VIO^s indicates that the system performed a hot violation. Thus, the chart can not be satisfied. VIO^e indicates that the environment violated its assumptions, and thus the chart is vacantly satisfied. Denote $\rho_{\mathcal{L}_i}$ the transition of the cut automaton $\mathcal{L}_i$. The following are the assumptions and guarantees, in GR(1) form, for the live sequence chart.

4.5.3.1 Superstep Requirements

In order to encode assumptions in live sequence chart semantics, we include superstep requirements. A superstep is a series of system messages encapsulated between environment messages. This enforces an artificial technical step to deal with the mechanics of a game structure, and thus ties in to the GR(1) synthesis algorithm.

Guarantee 1: The system will only sends a finite number of messages, allowing the environment a fair chance to communicate.

$$\Box\Diamond(m_s = no\_op). \tag{4.22}$$

Guarantee 2: The system performs a message only if the environment is not sending a message. Thus, if the environment send a message, the system cannot send a message.

$$\Box \bigcirc (m_e \neq no\_op \rightarrow m_s = no\_op). \tag{4.23}$$

Assumption 1: The environment can only send one message at a time, giving the system a fair chance to react. If the environment cannot sent a message in the next step if it has sent a message in the last step.

$$\Box\left(m_e \neq no\_op \rightarrow \bigcirc(m_e = no\_op)\right). \tag{4.24}$$

Assumption 2: If the system sent a message in the last step, the environment cannot send a message in the next step. This guarantees that the environment will not send a message if the system is not ready to receive one.

$$\Box\left(m_s \neq no\_op \rightarrow \bigcirc(m_e = no\_op)\right). \tag{4.25}$$

The semantics for superstep requirements are not application-specific, but rather model the live sequence chart settings. In the following, we describe the GR(1) formulation for application-specific live sequence chart specifications.

4.5.3.2 Environment Assumptions

Define $Exp_i \subseteq dom(l_i)$ to be the subset of cuts that contain executable environment messages. Given a cut $c \in dom(l_i)$, furthermore define $\xi^e(c)$ to be the set of hot environment messages enabled in cut c. If $c \in dom(l_i) \backslash Exp_i$, then $\xi^e(c) = 0$.

Assumption 3: For every live sequence chart $\mathcal{L}_i \in \mathcal{L}$ and expecting cut $c \in Exp_i$, if in the last step the system was in cut c then the environment in the next step will send either *no_op* or a message from the set of hot enabled messages:

$$\bigwedge_{i=1}^{n} \bigwedge_{c \in Exp_i} \Box(l_i = c \rightarrow \bigcirc(m_e \in \{\xi^e(c) \cup no\_op\})). \tag{4.26}$$

Assumption 4: If the system is in an expecting cup, each enabled hot environment message $m \in \xi^e(c)$ must eventually be sent:

$$\bigwedge_{i=1}^{n} \bigwedge_{c \in Exp_i} \bigwedge_{m \in \xi^e(c)} \Box\Diamond(l_i = c \rightarrow (m_e = m)). \tag{4.27}$$

Assumption 5: The environment must avoid letting the system reach the sink value that indicates a hot environment violation:

$$\bigwedge_{i=1}^{n} \Box(l_i \neq VIO^e). \tag{4.28}$$

4.5.3.3 System Guarantees

For a set of live sequence charts $\{\mathcal{L}_1, \ldots, \mathcal{L}_n\}$, we define $Act_i \subseteq dom(l_i)$ to be the cuts that contain an executable message the system should perform.

Guarantee 3: For every live sequence chart $\mathcal{L}_i \in \mathcal{L}$, the system starts from a state in which cut variable l_i is the minimal cut.

$$\bigwedge_{i=1}^{n} (l_i = MIN). \tag{4.29}$$

Guarantee 4: For every $\mathcal{L}_i \in \mathcal{L}$, the system continuously preserves the transitions of the cut automaton of $\mathcal{L}_i$.

$$\bigwedge_{i=1}^{n} \Box\rho_{\mathcal{L}_i}. \tag{4.30}$$

Guarantee 5: The system will always eventually reach a state in which $\mathcal{L}_i \in \mathcal{L}$ is not active,

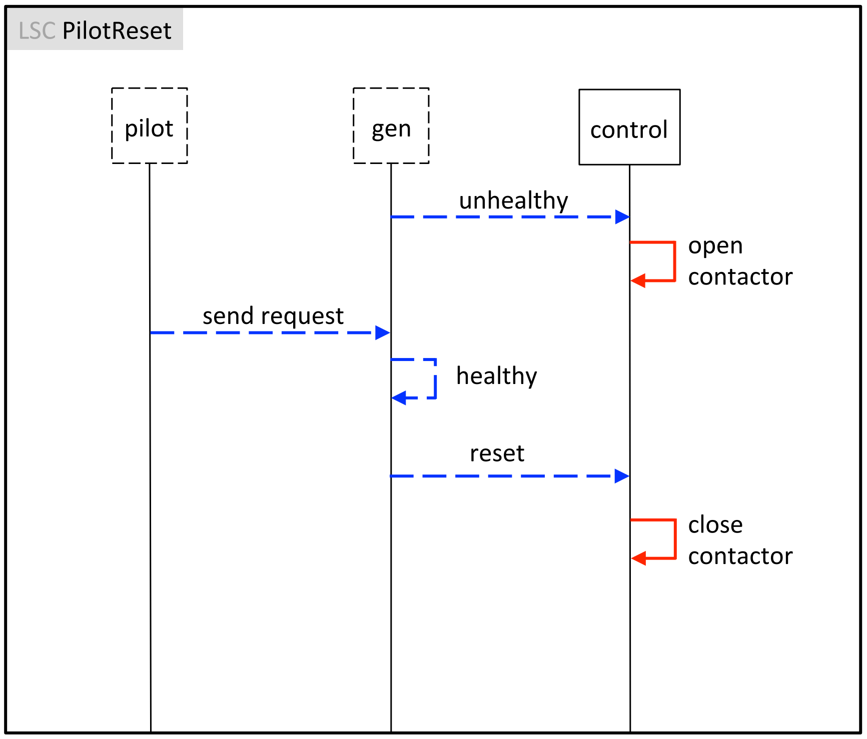


Figure 4.8: An example assume-guarantee live sequence chart with one system instance (control) and two environment instances (pilot and generator). Blue lines denote cold messages (eventually can happen), while red lines denote hot messages (must happen).

i.e., all charts visit inactive cuts in which there are no executable system messages.

$$\Box\Diamond \bigwedge_{i=1}^{n} (l_i \notin Act_i). \tag{4.31}$$

4.5.4 Live Sequence Chart Example

Figure 4.8 is a generic example assume-guarantee live sequence chart for the system topology depicted from Figure 4.5 (with 2 base topology units). If a generator becomes unhealthy (cold message), then the control must open the contactor. Then, the pilot may send a request (to turn the generator back online), the generator can turn back to healthy, and then send a reset message to the control. If the reset message is sent to the control, the contactor must close.

In addition to the specifications discussed in the previous section, the following assumptions are included in the synthesis formulation, as derived from the live sequence chart.

- The pilot environment request variable is set to an initial value of 0 (i.e., no request).

$$\bigwedge_{i\in\{0,1\}} req_i = 0,$$

  where index i represents the generator which the pilot requests to come back online.

- Generators cannot come back online unless they are requested. This is formulated as

$$\bigwedge_{i\in\{0,1\}} \square\{((req_i = 0) \wedge (g_i = 0)) \rightarrow (\bigcirc g_i = 0)\}.$$

The following are the additional guarantees generated from the live sequence chart.

- The controller reset variable is initially set to 0 (i.e., there is initially no reset flag raised).

$$\bigwedge_{i\in\{0,1\}} reset_i = 0.$$

- The reset flag is only raised if the pilot requests the generator to come back online.

$$\bigwedge_{i\in\{0,1\}} \square\{(req_i = 0) \rightarrow (reset_i = 0)\},$$

$$\bigwedge_{i\in\{0,1\}} \square\{(req_i = 1) \rightarrow (reset_i = 1)\}.$$

- If a generator is healthy (i.e., online) and the reset flag is raised, then the neighboring contactor must close.

$$\bigwedge_{i\in\{0,1\}} \square\{((g_i = 1) \wedge (reset_i = 1)) \rightarrow (c_{ij} = 1)\},$$

  where index j references a neighboring node to generator g_i.

Additional timing specifications can be incorporated into the above specifications by introducing counter variables. For instance, the reset flag may not necessarily be raised immediately once the pilot sets the request, but may occur after a given time delay. Likewise, contactor delays may be included (as discussed in Section 3.4).

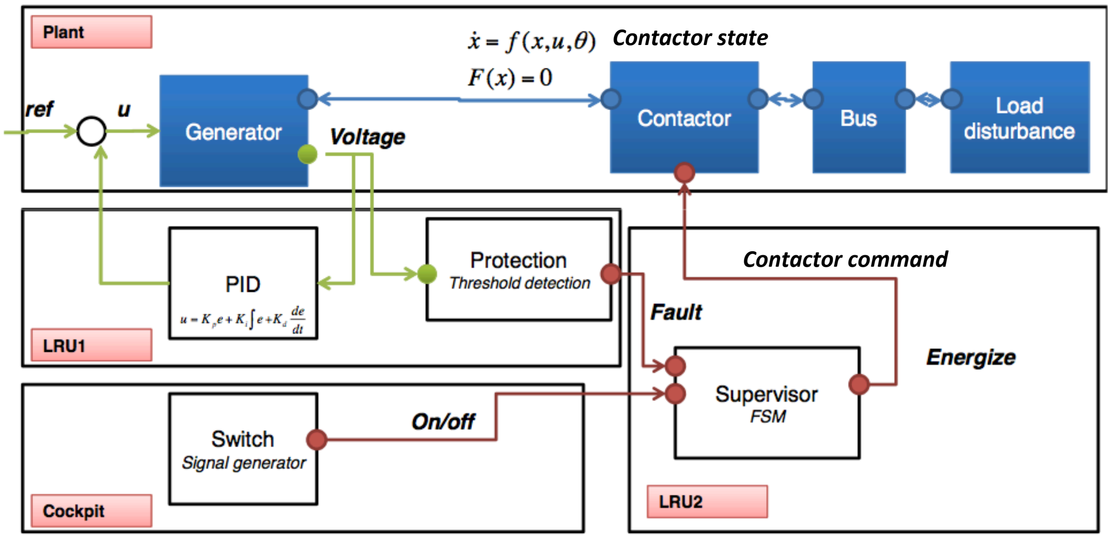


Figure 4.9: Unit test for a sample portion of an electric power system. The plant contains one generator, contactor, and bus. The cockpit (pilot) can send a switch to command the contactor position. Protection and supervisor controllers monitor the state of the system.

4.6 Timed Temporal Logics

In what follows we start with a set of text-based specifications and consider different ways of formalizing them. Depending on the model of computation one wants to reason about the specification, we adopt different state-based and event-based semantics. In particular, we use MTL with continuous semantics [46] and TCTL [6], in addition to GR(1) LTL formulations.

Consider the unit test configuration in Figure 4.9. The plant consists of a generator, contactor, and bus (subject to load disturbance). For the purposes of specification, we choose a level of abstraction high enough to ignore continuous dynamics. The generator outputs some voltage that is monitored by the Protection Line Replaceable Unit (LRU). The protector outputs a *fault* warning to the supervisor LRU if the generator voltage exceeds some threshold. The supervisor also monitors the *switch* command from the Cockpit (controlled by the pilot). Based on the states of *fault* and *switch*, the supervisor controls the state of the contactor.

The text-based specifications for the supervisor and protector are listed below.

Protector

1. If input is greater than threshold for z_1 time, fault output is true.

2. If input is less than threshold for z_2 time, fault output is false.

These specifications do not cover "if and only if"s. More precisely, it is likely possible for a fault output when there is no fault. For instance, for state-based semantics this would lead to " if the current state is $fault = false$, then the state becomes $fault = true$ if and only if input is greater than threshold for z_1 time." As a simplification, we can assume lazy controllers, i.e., if the preconditions do not hold, the controller should not take any action.

Supervisor

3. If fault occurs, open contactor within X time regardless of other inputs and leave open (no reset).

4. If switch is on, close contactor within Y1 time.

5. If switch is off, open contactor within Y2 time.

   These specifications are ambiguous in that it is not clear whether "open/close contactor" refers to the event of supervisor issuing an "open (close)" command or the event that the actual state of the contactor component becoming "open (close)". For the latter case, in order to synthesize a controller for these specifications, we need to know the relation between issuing a command and physical state change in the contactor. This can ideally be done by building a hybrid dynamical model for the contactor. For simplicity, in control synthesis, we will use the following assumptions:

6. If an open command is issued and if the contactor state is closed, the contactor opens within $[O_{min}, O_{max}]$ units of time unless a close command is issued before it opens.

7. If a close command is issued and if the contactor state is open, the contactor closes within $[C_{min}, C_{max}]$ units of time unless an open command is issued before it closes.

4.6.1 Timed Specifications

4.6.1.1 Protector

For the protector LRU, the voltage level v is an environment variable, and can take values of bt (below threshold) and at (above threshold). The system outputs a fault state variable $fault$ of f (there exists a fault) or nf (no fault). In MTL, specifications (1) - (2) can be written as:

1. $\Box\{(\Box_{[0,z_1)} v = at) \longrightarrow (\Diamond_{\{z_1\}} fault = f)\}$.

2. $\Box\{(\Box_{[0,z_2)} v = bt) \longrightarrow (\Diamond_{\{z_2\}} fault = nf)\}$.

In TCTL, we introduce an additional clock variable c on the voltage environment variable. Specifications (1) - (2) can be written in the form

1. $\forall\Box\{((v = at)\text{and}(c \geq z_1)) \longrightarrow (fault = f)\}$

2. $\forall\Box\{((v = bt)\text{and}(c \geq z_2)) \longrightarrow (fault = nf)\}$

3. $\forall\Box\{((fault = f)\text{and}(c = 0)) \longrightarrow \forall[(fault = f)W((v = at)\text{and}(c \geq z_1))]\}$

4. $\forall\Box\{((fault = nf\text{and}(c = 0)) \longrightarrow \forall[(fault = nf)W((v = bt)\text{and}(c \geq z_2))]\}$

where $\forall\Box$ indicates for all paths, and W is the operator for weak until (i.e., the condition before W does not have to be true).

4.6.1.2 Supervisor

For the Supervisor LRU, environment variables include the $fault$ variable (considered a system variable from the protector), $switch$, which can either be *on* or off, and contactor state cs that can either be *opened* or *closed*. System variables include a contactor command cc of *open* or *close*, and an additional $flag$ that can either be *normal* or *abnormal*. Specifications (3)-(7) are written as

1. (a) $\Box\{(fault = f) \longrightarrow ((\Diamond_X cs = open)\text{and}(flag = abnormal))\}$

   (b) $\Box\{((flag = abnormal)\text{and}(cs = open)) \longrightarrow (\Box cs = open)\}$

   (c) $\Box\{(flag = abnormal) \longrightarrow (\Box flag = abnormal)\}$

   (d) $\{(flag = normal)\ \mathcal{U}\ (fault = f)\}$

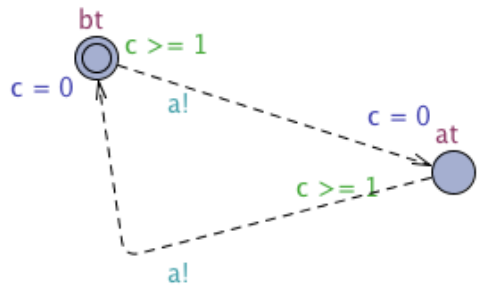
2. $\Box\,\{((flag = normal)\text{and}(switch = on)) \longrightarrow (\Diamond_{Y_1} cs = closed)\}$

3. $\Box\,\{((flag = normal)\text{and}(switch = off)) \longrightarrow (\Diamond_{Y_2} cs = open)\}$

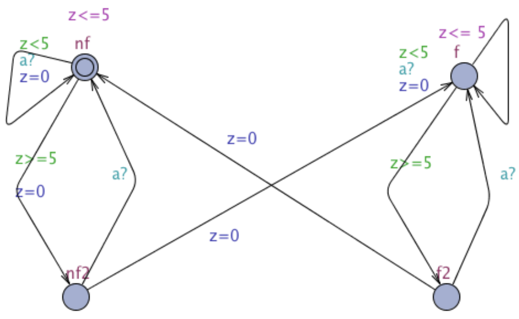
4. $\Box\{(cc = open)\text{and}(cs = closed)) \longrightarrow$
   $\{(\Diamond_{[O_{min},O_{max}]}(cs = open)\ \vee ((cs = closed)\ \mathcal{U}\ _{[0,O_{max}]}(cc = closed))\}\}$

5. $\Box\{(cc = close)\text{and}(cs = open)) \longrightarrow$
   $\{(\Diamond_{[C_{min},C_{max}]}(cs = closed)\ \vee\ ((cs = open)\ \mathcal{U}\ _{[0,C_{max}]}(cc = open))\}\}$

The TCTL version of specifications follow exactly from the above, with the replace of $\Box$ with a $\forall\Box$ operator.



(a) Voltage



(b) Fault

Figure 4.10: UPPAAL-TIGA finite-state automata for the protector LRU.

4.6.1.3 UPPAAL-TIGA

Figure 4.10 depicts the protector finite-state automata used for the timed synthesis tool UPPAAL-TIGA, which includes two *processes*: Voltage, and Fault. The Voltage process alternates between below threshold and above threshold. A local clock c counts the time the system is in each state. Dotted lines represent environment (uncontrollable) transitions, while solid lines represent controllable transitions. In addition, the message a is sent (denoted by the ! sign) from voltage, and received by the Fault process (denoted by the ? sign). In the Fault process, two pseudo-states are introduced in addition to the f and nf states. This is done because of the limitations in UPPAAL-TIGA specifications. The tool can synthesize controllers for only one specification, but not multiple ones. This could be solved with the conjunction of all specifications previously listed. However, UPPAAL-TIGA cannot process nested specifications, which occur because of the weak until operators W. The Fault process, therefore, encodes those specifications by design.

Figure 4.11 depicts the supervisor finite-state automata used in the timed synthesis problem. The three processes are fault, switch, and contactor. The fault can transition (uncontrolled) from normal to abnormal at any time. The pilot can also change the switch between on and off at any time. The contactor encodes from contactor command and contactor state. Again, the same problems arise in the supervisor LRU as in the protector LRU, namely, specifications cannot be nested.

While UPPAAL and UPPAAL-TIGA can be used to verify and synthesize timed problems, respectively, they are more useful for systems with simple specifications. For more complex specifications, even with simple unit tests used in our examples, this tool can be limited. While these requirements can be designed by hand, this bypasses the utility gained from synthesizing correct-by-construction controllers. We next discuss how to implement continuous time specifications in a discrete-time setting.

4.6.2 Discrete-Time LTL

Because of the limitations of timed synthesis tools, we additionally formulate the specifications for supervisor an protector in GR(1) LTL compatible with `TuLiP`, and synthesize controllers. To monitor time in the discrete setting, we introduce an additional clock variable x_v for the protector, and x_C for the supervisor.

4.6.2.1 Protector

In LTL, the specifications for the protector can be written as

1. Reset the clock to zero if the voltage state changes.

   $\square\{(\bigcirc v \neq v) \rightarrow (\bigcirc x_v = 0)\}$

2. If the voltage remains above threshold but the clock is less than the maximum time, then increment the clock by some amount δ.

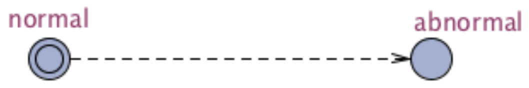
   $\square\{(v = at \wedge \bigcirc v = at \wedge (x_v < z_1)) \rightarrow (\bigcirc x_v = x_v + \delta)\}$

3. If the voltage remains above threshold and the clock is already at the maximum time, then keep the clock value the same. In other words, this sets a maximum bound on the clock value.

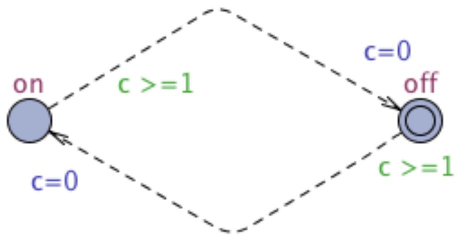
   $\square\{(v = at \wedge \bigcirc v = at \wedge (x_v = z_1)) \rightarrow (\bigcirc x_v = z_1)\}$

4. If the voltage remains below threshold but the clock is less than the maximum time, then increment the clock by some amount δ.

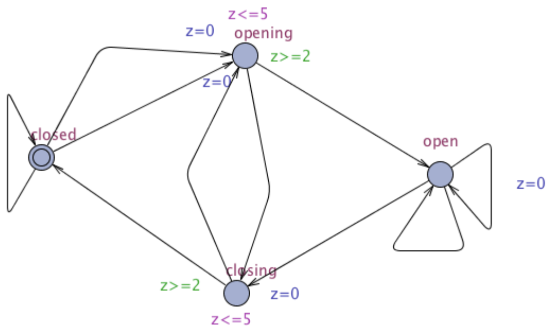
   $\square\{(v = bt \wedge \bigcirc v = bt \wedge (x_v < z_2)) \rightarrow (\bigcirc x_v = x_v + \delta)\}$



(a) Fault



(b) Switch



(c) Contactor

Figure 4.11: UPPAAL-TIGA finite-state automata for the supervisor LRU

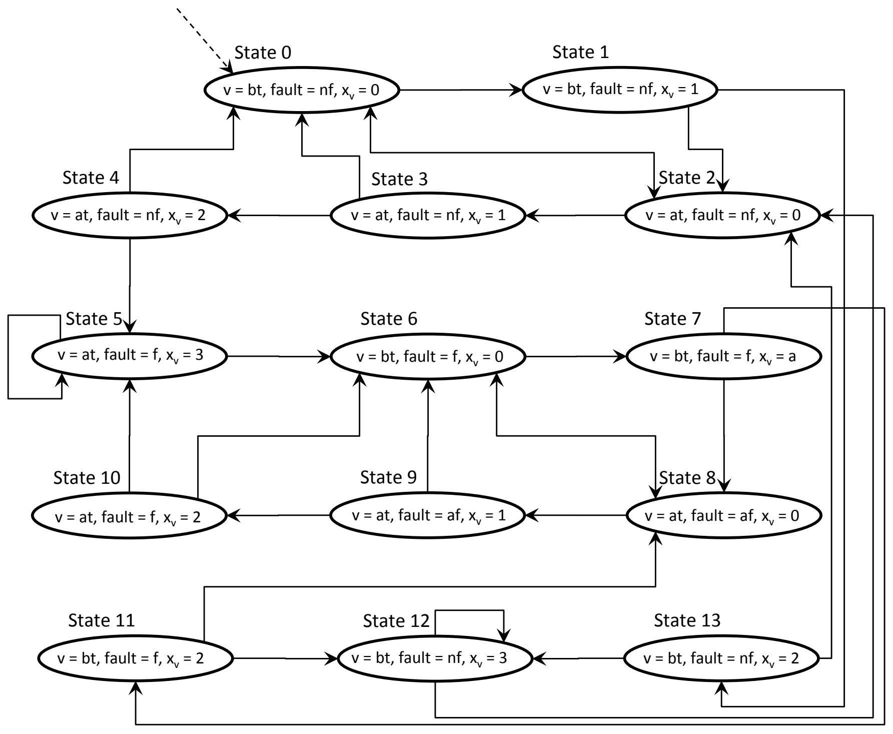

State 0
v = bt, fault = nf, x_v = 0
State 1
v = bt, fault = nf, x_v = 1
State 4
v = at, fault = nf, x_v = 2
State 3
v = at, fault = nf, x_v = 1
State 2
v = at, fault = nf, x_v = 0
State 5
v = at, fault = f, x_v = 3
State 6
v = bt, fault = f, x_v = 0
State 7
v = bt, fault = f, x_v = a
State 10
v = at, fault = f, x_v = 2
State 9
v = at, fault = af, x_v = 1
State 8
v = at, fault = af, x_v = 0
State 11
v = bt, fault = f, x_v = 2
State 12
v = bt, fault = nf, x_v = 3
State 13
v = bt, fault = nf, x_v = 2

Figure 4.12: UPPAAL-TIGA finite-state automata for the supervisor LRU

5. If the voltage remains above threshold and the clock is already at the maximum time, then keep the clock value the same.

   $\Box\{(v = bt \wedge \bigcirc v = bt \wedge (x_v = z_2)) \rightarrow (\bigcirc x_v = z_2)\}.$

6. If the voltage is above threshold for greater than z_1 length of time, then output a fault.

   $\Box\{(v = at \wedge x_v = z_1) \rightarrow (fault = f)\}.$

7. If the voltage is below threshold for greater than z_2 length of time, then output no fault.

   $\Box\{(v = bt \wedge x_v = z_2) \rightarrow (fault = nf)\}.$

8. The fault status cannot change until the voltage stays above threshold for at least z_2 time.

   $\Box\{((v = at) \wedge (\bigcirc x_v < z_2)) \rightarrow (\bigcirc fault = fault)\}$

9. The fault status cannot change until the voltage stays below threshold for z_1 time.

   $\Box\{((v = bt) \wedge (\bigcirc x_v < z_1)) \rightarrow (\bigcirc fault = fault)\}$

For time constants $z_1, z_2 = 3$, and a discrete time step $\delta = 1$, the synthesized controllers for the protector has 14 states. Figure 4.12 shows the resulting controller. The initial state (State 0) begins with a voltage below threshold, no fault, and counter x set to 0. If the voltage stays below threshold, the system transitions to State 1, and the counter increments by 1. If the voltage goes above threshold, the system transitions to State 2. If the voltage stays above the threshold for longer than two steps, fault outputs a value of f (State 5). From State 5, if the voltage crosses below threshold again, the fault output remains f until the voltage remains below threshold for 3 steps.

4.6.2.2 Supervior

In LTL, the specifications for the supervisor can be written as

Assumptions

1. If the contactor state is the same as the contactor command, then in the next step the contactor state should not change.

   $\square\{(cc = cs) \rightarrow (\bigcirc cs = cs)\}$

2. If the contactor command is set to close, then the contactor state should closed within C_{min} and C_{max} time.

   (a) $\square\{(cc = close \wedge cs = opened \wedge (x_C < C_{min})) \rightarrow (\bigcirc cs = opened \wedge \bigcirc x_C = x_C + \delta)\}$.

   (b) $\square\{(cc = close \wedge cs = opened \wedge (x_C \geq C_{min})) \rightarrow (\bigcirc cs = closed \vee \bigcirc x_C = x_C + \delta)\}$.

   (c) $\square\{(cc = close \wedge cs = opened) \rightarrow (x_C \leq C_{max})\}$.

3. If the contactor command is set to open, then the contactor state should be opened within O_{min} and O_{max} time.

   (a) $\square\{(cc = open \wedge cs = closed \wedge (x_C < O_{min})) \rightarrow (\bigcirc cs = closed \wedge \bigcirc x_C = x_C + \delta)\}$.

   (b) $\square\{(cc = open \wedge cs = closed \wedge (x_C \geq O_{min})) \rightarrow (\bigcirc cs = opened \vee \bigcirc x_C = x_C + \delta)\}$.

   (c) $\square\{(cc = open \wedge cs = closed) \rightarrow (x_C \leq O_{max})\}$.

Guarantees

1. (a) Reset the clock if in the next step the contactor state matches the command.

   $\square\{(\bigcirc cs = cc) \rightarrow (\bigcirc x_c = 0)\}$

   (b) If there is a fault, then open the contactor.

   $\square\{(fault = f) \longrightarrow ((cc = 0) \wedge (flag = abnormal))\}$

   (c) If a fault occurs, always leave the contactor open.

   $\square\{((flag = abnormal) \wedge (cc = 0)) \longrightarrow (\bigcirc cc = 0)\}$

(d) $\square\{(flag = abnormal) \longrightarrow (\bigcirc flag = abnormal)\}$

(e) Do not raise a flag unnecessarily.

$\square\{((flag = normal) \wedge (\bigcirc fault = nf)) \longrightarrow (\bigcirc flag = normal)\}$

2. If the switch command is set to *on*, then close the contactor.

   $\square\,\{((flag = normal) \wedge (switch = on)) \longrightarrow (cc = close)\}$

3. If the switch command is set to off, then open the contactor.

   $\square\,\{((flag = normal) \wedge (switch = off)) \longrightarrow (cc = open)\}$

For minimum and maximum opening and closing time constraints of $O_{min}, C_{min} = 2, O_{max}, C_{max} = 4$, and for a time step of $\delta = 1$, the resulting synthesized controller has 32 states. The size of the finite-state automaton will increase with the the size of the discretization δ as well as the values of C_{max} and O_{max}.

4.7 Conclusions

We have demonstrated techniques for synthesis of discrete-variable, untimed and discrete-time control protocols. Further extensions also include extending the domain-specific language to include user-specific requirements that may not be included in the high-level general specifications described in this chapter. Generation specification from assume-guarantee live sequence charts are performed manually. While tools exist to synthesize controllers from a given live sequence chart [57], this does not allow integration of live sequence chart specification with other system requirements. Integration of live sequence chart specifications with `TuLiP` is subject of future work. In addition, we are also exploring the use of this tool and language to distributed controller protocols. Namely, how to distribute a given topology among subsystems and generate interface specifications such that the overall system is realizable. Lastly, the problem of network effects, including transients and delays, has been mostly ignored or abstracted away within this problem formulation. Introducing specifications encompassing network effects would be an additional feature for a domain-specific language.

This chapter has also demonstrated how to formulate timed specifications within the framework of MTL and TCTL. The TCTL synthesis solver UPPAAL-TIGA is limited in the specifications it is capable of handling, while no tool for MTL synthesis (or even model checking) is currently available. This led to a formulation of specifications in LTL, in particular, within the fragment of GR(1) by the use of additional of counters. The main challenge that synthesis may be able to solve is the interaction of timing delays within the larger system. Such problems are the subject of future work.

Chapter 5

Design Space Exploration

This chapter addresses the concept of design space exploration for an aircraft electric power system. To achieve an optimal implementation that is correct by construction, we propose a methodology for electric power system design that enables independent implementability of system topology (i.e. interconnection among elements) and control protocols by using a compositional approach. In this flow, design space exploration is carried out as a sequence of refinement steps from the initial specification towards a final implementation by mapping higher-level behavioral and performance models into a set of library components at the lower level. To perform such tasks, we define convenient abstractions for system exploration and compositional synthesis of system topology (interconnection among the various components) and control. We first synthesize an electric power system topology from system requirements formalized as arithmetic constraints on Boolean variables. For the given topology, we translate the same requirements into linear temporal logic formulas (as discussed in Chapter 3), by which we create correct-by-construction controllers. To reason about different requirements in a compositional way, we use the concept of *contracts* [81] that formalize the notion of *interfaces* between models and tools in the design flow.

Section 5.0.1 provides background on contract-based design, including the notion of contracts and components, as well as Signal Temporal Logic (STL), a specification language We show the effectiveness of our approach on a proof-of-concept design based on an electric power system case study. Section 5.1 applies the design space exploration methodology on a case study. The work presented in this section is in collaboration with Pierluigi Nuzzo, Necmiye Ozay, and Alexander Donze. Finally, in Section 5.2, we present a hardware testbed for the electric power system in which to validate synthesized reactive controllers within a real-time setting.

5.0.1 Background: Contract-Based Design of Cyberphysical Systems

Inspired by recent results on assume-guarantee compositional reasoning and interface theories in the context of hybrid systems and software verification, our methodology is based on the use of assume-guarantee contracts for cyber-physical systems [12, 81]. Informally, contracts mimic the thought process of a designer, who aims at *guaranteeing* certain performance figures for the design under specific *assumptions* on its environment. The essence of contracts is, therefore, a *compositional* approach, where design and verification complexity is reduced by decomposing system-level tasks into more manageable subproblems at the component level, under a set of assumptions. System properties can then be inferred or proved based on component properties. In this respect, contract-based design can be a rigorous and effective paradigm while dealing with the complexity of modern system design, and has been successfully applied to other embedded system domains, such as automotive applications [12] and mixed-signal integrated circuits [66].

5.0.1.1 Components

We summarize the main concepts behind contract-based design starting with the notion of components. A *component* $\mathcal{M}$ can be seen as an abstraction, a hierarchical entity representing an element of a design, characterized by the following *component attributes*:

- a set of input, output and internal *variables* (including state variables); a set of configuration *parameters*, and a set of input, output and bidirectional *ports* for connections with other components;

- a set of *behaviors*, which can be implicitly represented by a dynamic *behavioral model* $\mathcal{F}(\cdot) = 0$, uniquely determining the value of the output and internal variables given the one of the input variables and configuration parameters. Behaviors are generic, and could be continuous functions that result from solving differential equations, or sequences of values or events recognized by an automata model;

- a set of *non-functional models*, i.e. maps that allow computing non-functional attributes of a component corresponding to particular valuations of its input variables and configuration parameters. Examples of non-functional maps include the *performance model*, computing a set of performance figures by solving the behavioral model, or the *reliability model*, providing the failure probability of a component.

Components can be connected together by sharing certain ports under constraints on the values of certain variables. In what follows, we use *variables* to denote both component variables and ports. Moreover, components can be hierarchically organized to represent the system at different levels of abstraction. Given a set of components at level *l*, a system can then be composed by *parallel composition* and represented as a new component at level *l+1*. At each level of abstraction, components are also capable of exposing multiple, complementary *views*, associated to different concerns (e.g. safety, performance, reliability), which can be expressed via different formalisms and analyzed by different tools.

A component may be associated to both implementations and contracts. An *implementation* M is an instantiation of a component $\mathcal{M}$ for a given set of configuration parameters. The definition of contract is presented below.

5.0.1.2 Contracts

A *contract* $\mathcal{C}$ for a component $\mathcal{M}$ is a pair of assertions (A, G), called the *assumptions* and the *guarantees*. An *assertion* B represents a specific set of behaviors over variables, which is the set satisfying B. Therefore, operations on assertions and contracts are set operations. An implementation M satisfies an assertion B whenever M and B are defined over the same set of variables and all the behaviors of M satisfy the assertion, i.e. when $M \subseteq B$. The set of all the legal *environments* for $\mathcal{C}$ collects all implementations E such that $E \subseteq A$. An implementation of a component satisfies a contract whenever it satisfies its guarantee, subject to the assumption. Formally, $M \cap A \subseteq G$, where M and $\mathcal{C}$ have the same variables. We denote such a *satisfaction* relation by writing $M \models \mathcal{C}$. Similarly, we relate a legal environment E to a contract $\mathcal{C}$ by the satisfaction relation $E \models_E \mathcal{C}$.

Any implementation M of a component such that $M \subseteq G \cup \neg A$, where $\neg A$ is the complement of A, is also an implementation for $\mathcal{C}$. In general, $M_{\mathcal{C}} = G \cup \neg A$ is the maximal implementation for $\mathcal{C}$. Two contracts $\mathcal{C}$ and $\mathcal{C}'$ with identical variables, identical assumptions, and such that $G' \cup \neg A = G \cup \neg A$, possess identical sets of implementations. Such two contracts are then *equivalent*. Therefore, any contract $\mathcal{C} = (A, G)$ is equivalent to a contract in *saturated form* (A, G'), which also satisfies $G' \supseteq \neg A$, or, equivalently, $G' \cup A = True$, the true assertion. To obtain the saturated form of a contract, it is enough to take $G' = G \cup \neg A$.

Contracts associated to different components can be combined according to different rules. Similar to parallel composition of components, *parallel composition* of contracts can be used to construct composite contracts out of simpler ones. Let $\mathcal{C}_1 = (A_1, G_1)$ and $\mathcal{C}_2 = (A_2, G_2)$ in saturated form, then the assumption and the promise of the composite $\mathcal{C}_1 \otimes \mathcal{C}_2$ can be computed as follows [12]:

$$A = (A_1 \cap A_2) \cup \neg(G_1 \cap G_2), \tag{5.1}$$

$$G = G_1 \cap G_2. \tag{5.2}$$

The composite contract must clearly satisfy the guarantees of both. Moreover, since the environment should satisfy all the assumptions, we should expect that the assumptions of each contract would also combine by conjunction. In general, however, part of the assumptions A_1 will be already satisfied by composing $\mathcal{C}_1$ with $\mathcal{C}_2$, which acts as a partial environment for $\mathcal{C}_1$. Therefore, G_2 can relax the assumptions A_1, and vice-versa, which motivates (5.1). To use (5.1) and (5.2), the behaviors related to the original contracts need to be extended to a common set of variables. Such an extension, which is also called *alphabet equalization*, can be achieved by an operation of inverse projection [12].

Even if they need to be satisfied simultaneously, multiple views of the same component do not generally compose by parallel composition. Therefore, the *conjunction* ($\wedge$) of contracts can also be defined so that if $M \models \mathcal{C}_1 \wedge \mathcal{C}_2$, then $M \models \mathcal{C}_1$ and $M \models \mathcal{C}_2$. Contract conjunction can be computed by defining a partial order on contracts, which formalizes a notion of *refinement*. We say that $\mathcal{C}$ refines $\mathcal{C}'$, written $\mathcal{C} \preceq \mathcal{C}'$ (with $\mathcal{C}$ and $\mathcal{C}'$ both in saturated form), if and only if $A \supseteq A'$ and $G \subseteq G'$. Refinement amounts to relaxing assumptions and reinforcing guarantees, therefore strengthening the contract. Clearly, if $M \models \mathcal{C}$ and $\mathcal{C} \preceq \mathcal{C}'$, then $M \models \mathcal{C}'$. On the other hand, if $E \models_E \mathcal{C}'$, then $E \models_E \mathcal{C}$. With the given ordering, we can compute the conjunction of contracts by taking the greatest lower bound of $\mathcal{C}_1$ and $\mathcal{C}_2$. For contracts in saturated form, we have

$$C_1 \wedge C_2 = (A_1 \cup A_2, G_1 \cap G_2), \tag{5.3}$$

i.e. conjunction of contracts amounts to taking the intersection of the guarantees and the union of the assumptions. Conjunction can be used to compute the overall contract for a component starting from the contracts related to multiple views (concerns, requirements) in a design.

In addition to satisfaction and refinement, *consistency* and *compatibility* are also relations involving contracts. Technically, these two notions refer to individual contracts. A contract is consistent when the set of implementations satisfying it is not empty, i.e. it is feasible to develop implementations for it. For contracts in saturated form, this amounts to verify that $G \neq \emptyset$. Let M be any implementation, i.e. $M \models \mathcal{C}$, then $\mathcal{C}$ is compatible, if there exists a legal environment E for $\mathcal{C}$, i.e. if and only if $A \neq \emptyset$. The intent is that a component satisfying contract $\mathcal{C}$ can only be used in the context of a compatible environment. In practice, however, violations of consistency and compatibility occur as a result of a parallel composition, so that we can refer to the collection of components forming a composite contract as being consistent or compatible.

5.0.1.3 Signal Temporal Logic

In addition to LTL, Signal Temporal Logic (STL) is likewise a particularly suitable for capturing system and component requirements and reasoning about the correctness of their behaviors.

LTL allows formally reasoning about the temporal behaviors of reactive systems with Boolean, discrete-time signals or sequences of events. To deal with dense-time real signals and hybrid dynamical model that mix the discrete dynamics of the controller with the continuous dynamics of the plant, several logics have been introduced over the years, such as Timed Propositional Temporal Logic [3], and Metric Temporal Logic (MTL) [46]. Signal Temporal Logic (STL) [55] has been proposed more recently as a specification language for constraints on real-valued signals in the context of analog and mixed-signal circuits. In this chapter, we refine LTL system requirements into constraints on physical variables (e.g. voltages and currents) expressed using STL constructs. Then, we monitor and process simulation traces to verify constraints satisfaction, while optimizing a set of design parameters.

For a hybrid dynamical model, we define a *signal* as a function mapping the time domain $\mathbb{T} = \mathbb{R}_{\geq 0}$ to the reals $\mathbb{R}$. A multi-dimensional signal $\boldsymbol{x}$ is then a function from $\mathbb{T}$ to $\mathbb{R}^n$ such that $\forall t \in \mathbb{T}$, $\boldsymbol{x}(t) = (x_1(t), \cdots, x_n(t))$, where $x_i(t)$ is the i-th component of vector $\boldsymbol{x}(t)$. Moreover, we assume that a hybrid system behavioral model $\mathcal{F}$ (e.g. implemented in a simulator) takes as input a signal $\boldsymbol{u}(t)$ and computes an output signal $\boldsymbol{y}(t) = \mathcal{F}(\boldsymbol{u}(t))$. The collection of output signals resulting from a simulation of the system is a *trace*, which can also be viewed as a multi-dimensional signal.

In STL, constraints on real-valued signals, or *predicates*, can be reduced to the form $\mu = f(\boldsymbol{x}) \sim \pi$, where f is a scalar-valued function over the signal $\boldsymbol{x}$, $\sim \in \{<, \leq, \geq, >, =, \neq\}$, and π is a real number. As in LTL, temporal formulas are formed using temporal operators, *always*, *eventually* and *until*. However, each temporal operator is indexed by intervals of the form (a, b), $(a, b]$, $[a, b)$, $[a, b]$, (a, ∞) or $[a, \infty)$, where each of a, b is a non-negative real-valued constant. If I is an interval, then an STL formula is written using the following grammar:

$$\varphi := True \mid \mu \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \, \mathcal{U}_I \, \varphi_2$$

The *always* and *eventually* operators are defined as special cases of the *until* operator as follows:

$\Box_I \varphi \triangleq \neg \Diamond_I \neg \varphi$, $\Diamond_I \varphi \triangleq True \mathcal{U}_I \varphi$. When the interval I is omitted, we use the default interval of $[0, +\infty)$.

The semantics of STL formulas are defined informally as follows. The signal $\boldsymbol{x}$ satisfies $\mu = f(\boldsymbol{x}) < 2$ at time t (where $t \geq 0$), written $(\boldsymbol{x}, t) \models \mu$, if $f(\boldsymbol{x}(t)) < 2$. It satisfies $\varphi = \Box_{[0,2)} (x > -1)$, written $(\boldsymbol{x}, t) \models \varphi$, if for all time $0 \leq t < 2$, $x(t) > -1$. The signal x_1 satisfies $\varphi = \Diamond_{[1,2)} x_1 > 0.4$ iff there exists time t such that $1 \leq t < 2$ and $x_1(t) > 0.4$. The two-dimensional signal $\boldsymbol{x} = (x_1, x_2)$ satisfies the formula $\varphi = (x_1 > 10) \, \mathcal{U}_{[2.3,4.5]} \, (x_2 < 1)$ iff there is some time u where $2.3 \leq u \leq 4.5$ and $x_2(u) < 1$, and for all time v in $[2.3, u)$, $x_1(u)$ is greater than 10.

We write $\boldsymbol{x} \models \varphi$ as a shorthand of $(\boldsymbol{x}, 0) \models \varphi$. Formal semantics can be found in [55].

Parametric Signal Temporal Logic (PSTL) is an extension of STL introduced in [5] to define *template formulas* containing unknown parameters. Syntactically speaking, a PSTL formula is an STL formula where numeric constants, either in the constraints given by the predicates μ or in the time intervals of the temporal operators, can be replaced by symbolic parameters. These parameters are divided into two types:

- A *scale* parameter π is a parameter appearing in predicates of the form $\mu = f(\boldsymbol{x}) \sim \pi$,
- A *time* parameter τ is a parameter appearing in an interval of a temporal operator.

An STL formula is obtained by pairing a PSTL formula with a valuation function that assigns a value to each symbolic parameter. For example, consider the PSTL formula $\varphi(\pi, \tau) = \Box_{[0,\tau]} x > \pi$, with symbolic parameters π (scale) and τ (time). The STL formula $\Box_{[0,10]} x > 1.2$ is an instance of φ obtained with the valuation $v = \{\tau \mapsto 10, \ \pi \mapsto 1.2\}$.

5.1 Design Space Exploration: Case Study

Our design flow, pictorially represented in Fig. 5.1, consists of two main steps, namely topology design and control design. The *topology design* step instantiates electric power system components and connections among them to generate an optimal topology while guaranteeing the desired reliability level. Given this topology, a reactive control logic can then be synthesized in the *control design* phase, to drive contactors while guaranteeing that loads are correctly powered. The above two steps are, however, connected. The correctness of the controller needs to be enforced in conjunction with its boundary conditions, i.e. the assumptions on the entities that are not controlled, yet interact with it. An example of such an assumption is the number of paths from generators to a load made available by the electric power system architecture to the controller. Similarly, the reliability of an architecture

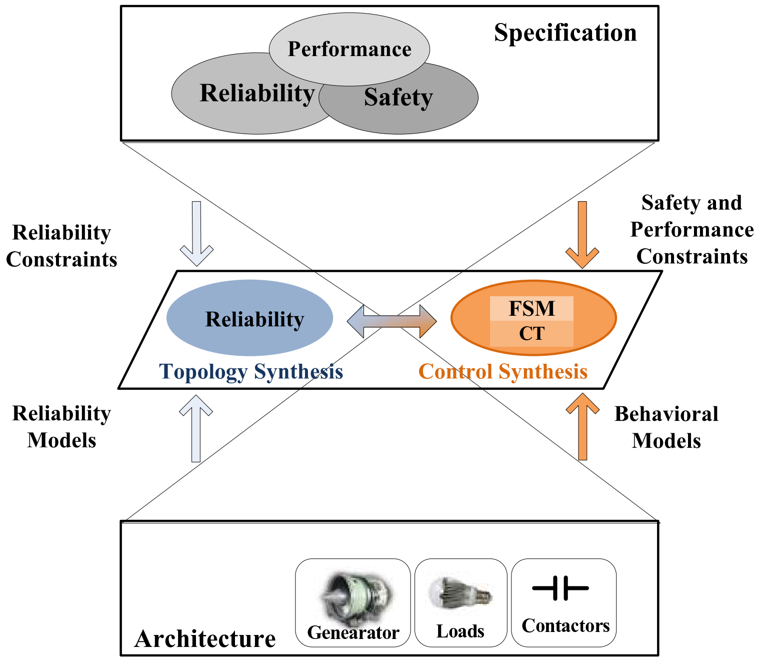


Figure 5.1: Proposed electric power system topology and control design flow.

must be assessed in conjunction with the assumptions for the controller to adequately configure its contactors and leverage the available paths. Therefore, to achieve independent implementability of architecture and controller, we address the synthesis problem in a compositional way, by using contracts to incorporate the information on the environment conditions under which each entity is expected to operate.

Our design process includes a top-down and a bottom-up phase. In the top-down phase, we associate the requirements to the different entities in the system and formulate top-down vertical contracts for them. In the bottom-up phase, we build a library of components including, for instance, generators, buses, power converters and contactors. Each component is characterized by its attributes, including multiple models or views, such as behavioral or reliability views, and finite state machine (FSM) or continuous-time models. Horizontal contracts specify legal compositions between these models. Bottom-up vertical contracts define under which conditions the models are a faithful representation of the physical elements in the system.

In what follows, we provide details on the electric power system design space exploration.

5.1.1 Electric Power System

There is currently no automated procedure for optimal synthesis of control protocols simultaneously subject to reliability, safety and real-time performance constraints. Therefore, we aim to reason about these three aspects of the design, by using specialized analysis and synthesis frameworks operating using different formalisms. Contracts specifying the interface between components and views help transfer requirements between different frameworks and verify correctness with respect to the full set of requirements. Our design space exploration is organized as follows:

a) From system requirements, we generate a set of constraints for the electric power system architecture. Safety, connectivity and power flow constraints are expressed as arithmetic constraints on Boolean variables (mixed integer-linear inequalities); reliability constraints are inequalities on real numbers involving component failure probabilities. Such constraints encode both the guarantees offered by the architecture as well as the assumptions on the underlying control protocol (horizontal contracts between the plant and its controller). The trade-off between redundancy and cost can then be explored and an electric power system topology is synthesized to minimize the total component cost while satisfying the constraints above. The synthesized topology serves as a specification for the subsequent control design step.

b) The original high-level electric power system specifications are translated into LTL formulas for

Table 5.1: Load Requirements

Component	Requirement (W)
LL1	3000
LL2	4000
RL1	2000
RL2	3000

Table 5.2: Generator Power Ratings

Component	Capability (W)
LG1	7000
LG2	3000
RG1	5000
RG2	4000
APU	10000

the topology generated in a). Using the results from Chapters 3-4, a reactive control protocol is then synthesized from LTL constructs and made available as one (or more) state machines, satisfying safety specifications by construction. However, no notion of the architectural and real-time constraints (e.g. timing) related to the physical plant and the hardware implementation of the control algorithm are available at this level. In this work, timing constraints are handled at a lower abstraction level, as detailed below.

c) The architecture in a) and the controller in b) are executed using continuous-time behavioral models to check for their compatibility (horizontal contracts) and assess satisfaction of all the requirements at a lower abstraction level. LTL requirements are refined into STL formulas. Simulation traces are monitored to verify and optimize the controller. As an example, an optimal clock period can be selected in the presence of delays in the switches and under the assumption of a synchronous implementation. The resulting architecture and controller pair is then returned as the final design.

5.1.2 Topology Synthesis

We illustrate our methodology on the proof-of-concept design of the primary power distribution of an electric power system, involving the configuration of contactors to deliver power to high-voltage AC and DC buses and loads.

The topology synthesis algorithm has been implemented in MATLAB and leverages CPLEX [1] to solve the MILP at each iteration. We present the result obtained for an electric power system topology template $\mathcal{T}$ consisting of two generators, two AC buses, two rectifiers, two DC buses and two loads on each side. Tables 5.1 and 5.2 report the load power requirements and the generator power ratings in our example; Table 5.3 shows the component costs, while the failure probabilities are reported in Table 5.4.

Figures 5.2 and 5.3 show the topologies obtained after running the synthesis algorithm when a set of strategies to increase reliability are sequentially implemented after every MILP iteration. By

Table 5.3: Component Costs

Component	Cost
Generator	Generator power/10
APU	APU power/10
AC-Bus	200
Rectifier	200
DC-Bus	200
Contactor	100

Table 5.4: Component failure probabilities

Component	Failure Probability
Generator/APU	10^{-5}
AC/DC bus	0
Rectifier	2×10^{-4}
contactor, load	0

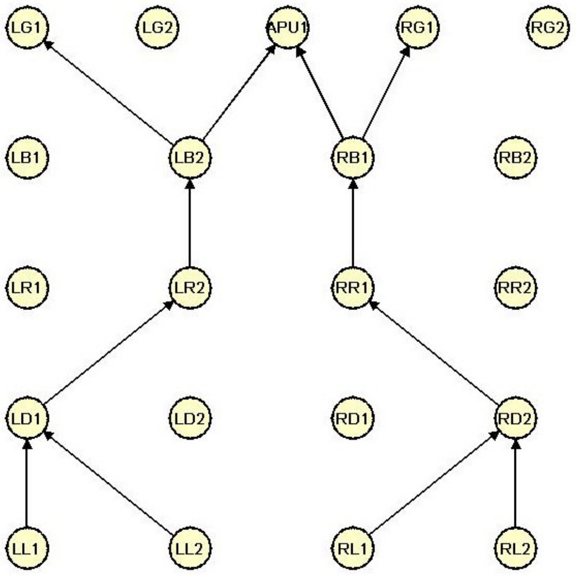
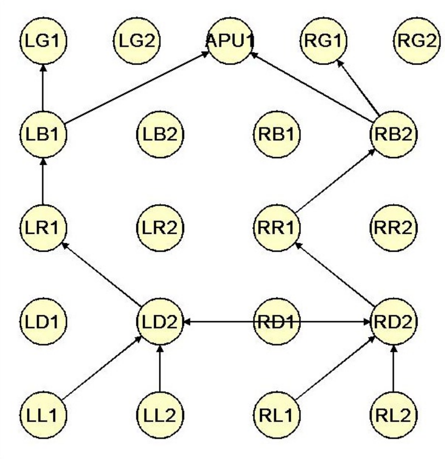
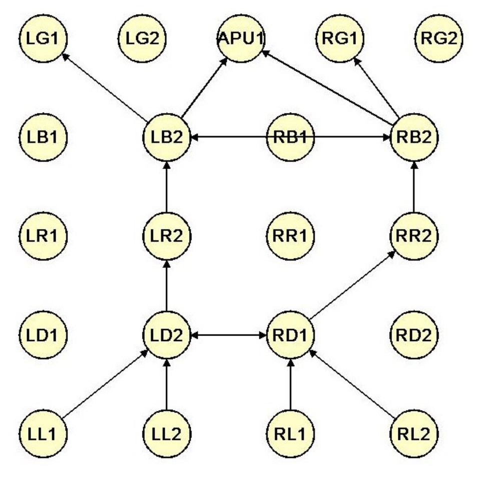


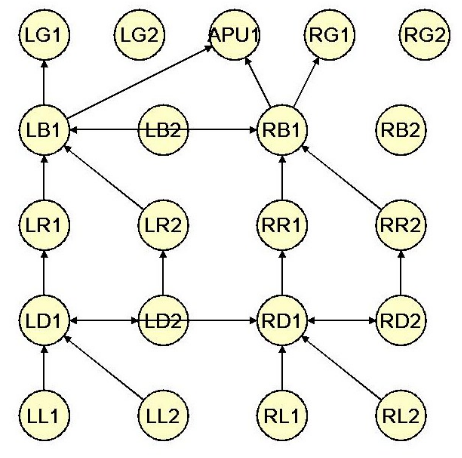
Figure 5.2: Directed graph representation of an electric power system architecture. Unconnected nodes represent virtual components.



(a)



(b)



(c)

Figure 5.3: Topologies 2 (a), 3 (b) and 4 (c).

Table 5.5: Failure Probability at Load $LL1$ for Topologies 1-4

Topology	Failure Probability
1	2.3×10^{-4}
2	4.5×10^{-8}
3	4.5×10^{-8}
4	1×10^{-12}

solving the MILP including only connectivity and power flow constraints, we obtain the topology in Fig. 5.2, the simplest possible architecture, which only provides a single path from a load to a generator (or APU) on each side. Such a topology presents the lowest reliability level at its loads, as shown in Table 5.5.

In Fig. 5.3 a) and b) horizontal connections are added between the DC buses and AC buses of the left and right hand sides of the system. Since increasing the number of components is expensive, the algorithm first tries to increase reliability by just adding connections among existing components at the only cost of additional contactors. Additional components (e.g. buses and rectifiers) are finally used in Fig. 5.3 c). In Table 5.5, we report the reliability (failure probability) at load $LL1$, as computed for the topologies in Fig. 5.2 and 5.3.

In a typical run, the number of necessary paths to increase reliability is estimated at the first MILP step and convergence to the final topology occurs in no more than two iterations.

5.1.3 Control Synthesis

For each of the four topologies in Figures 5.2-5.3, we formalize a set of environment assumptions and system specifications to synthesize a control protocol. For the purpose of brevity, we present the variables and formal specifications, written in LTL, for the topology depicted in Fig. 5.3 b) only.

Environment Variables: Generators $LG1, LG2, APU1$ and rectifier units $LR2$ and $RR2$ are uncontrolled variables that can switch between healthy (1) and unhealthy (0).

Controlled Variables: Contactors $C_{i,j}$[1] (depicted only as wires in Fig. 5.3) are variables that are set to open (0) or closed (1).

Dependent Variables: Buses are either powered (1) or unpowered (2) depending on the status of environment and controlled variables.

Environment Assumption: For an overall system reliability of 10^{-9}, no more than one generator

[1] i and j denote the name of the components contactor $C_{i,j}$ connects.

and one rectifier unit may be unhealthy at any given time. This is written as

$$\begin{aligned}&\Box((LG1 + LG2 + APU1) \geq 2)\\ &\wedge \Box((LR2 + RR2) \geq 1).\end{aligned}$$

No Paralleling of AC Sources: No combination of contactors can be closed so that a path exists between generators.

$$\begin{aligned}&\Box\neg((C_{LG1,LB2} = 1) \wedge (C_{APU1,LB2} = 1))\\ &\wedge \Box\neg((C_{APU1,RB2} = 1) \wedge (C_{RG1,RB2} = 1)).\end{aligned}$$

Power Status of Buses: A bus can only be powered if there exists a path (in which a contactor is closed) between a bus and a generator. In Fig. 5.3 b), bus $LB2$ is powered if either generator $LG1$ or $APU1$ is powered, and the contactor between generator and bus is closed.

$$\begin{aligned}&\Box((LG1 = 1) \wedge (C_{LG1,LB2} = 1) \rightarrow (LB2 = 1))\\ &\Box((APU1 = 1) \wedge (C_{APU1,LB2} = 1) \rightarrow (LB2 = 1)).\end{aligned}$$

If neither of these two cases is true, then $LB2$ will be unpowered. These specifications are written as

$$\begin{aligned}&\Box(\neg(((LG1 = 1) \wedge (C_{LG1,LB2} = 1)\\ &\vee((APU1 = 1) \wedge (C_{APU1,LB2} = 1))) \rightarrow (LB2 = 0)).\end{aligned}$$

Similar specifications may be written for buses $RB2, LD2$, and $RD1$.

Safety-Criticality of Buses: We consider all buses to be safety-critical, so that at no time can any bus be unpowered

$$\Box((LB2 = 1) \wedge (RB2 = 1) \wedge (LD2 = 1) \wedge (RD1 = 1)).$$

The resulting controller has 32 states with a computation time of 1.6 seconds on a Powerbook 2.2 GHz Intel Core Processor.

5.1.4 Distributed Synthesis

Given a global specification and a system composed of subsystems, distributed synthesis proceeds by first finding local specifications for each subsystem, and then synthesizing local controllers for these subsystems separately. If the local specifications satisfy certain conditions, it can be shown that the local controllers realizing these local specifications can be implemented together and the overall system is guaranteed to satisfy the global specification, as detailed in [67]. We describe below a special case of distributed architecture, i.e. a serial interconnection of controllers, which is used in the design in Section 5.1.4.1 to synthesize controllers for AC and DC subsystems separately.

Theorem 5.1.1 *Given*

- *a system characterized by a set $S = P \cup E$ of variables, where P and E are disjoint sets of controllable and environment variables,*
- *its two subsystems with variables $S_1 = P_1 \cup E_1$ and $S_2 = P_2 \cup E_2$, where for each $i \in \{1, 2\}$, P_i and E_i are disjoint sets of controllable and environment variables for the i^{th} subsystem, P_1 and P_2 are disjoint, and $P = P_1 \cup P_2$,*
- *a set $\mathcal{I}$ of pairs of variables representing the interconnection structure, that is, for a serial interconnection, $\mathcal{I} = \{(o_1, i_2) | o_1 \in O_1 \subseteq (P_1 \cup E_1), i_2 \in I_2 \subseteq E_2\}$, where for all $(o, i) \in \mathcal{I}$, $o = i$,*
- *a global specification $\varphi_e \to \varphi_s$, and two local specifications $\varphi_{e_1} \to \varphi_{s_1}$ and $\varphi_{e_2} \to \varphi_{s_2}$, where φ_e, φ_{e_1}, φ_{e_2}, φ_s, φ_{s_1}, and φ_{s_2} are LTL formulas containing variables only from their respective sets of environment variables E, E_1, E_2 and system variables S, S_1, S_2;*

if the following conditions hold:

1. *any behavior that satisfies φ_e also satisfies $(\varphi_{e_1} \wedge \varphi_{e_2})$,*
2. *any behavior that satisfies $(\varphi_{s_1} \wedge \varphi_{s_2})$ also satisfies φ_s,*
3. *there exist two controllers that make the local specifications $(\varphi_{e_1} \to \varphi_{s_1})$ and $(\varphi_{e_2} \to \varphi_{s_2})$ true with φ_{e_1} and φ_{e_2} both true;*

then, implementing the two controller together leads to a controller that satisfies the global specification $\varphi_e \to \varphi_s$.

Proof: The conditions on P, P_1, P_2 ensure that the two controllers are composable, i.e. they do not try to control the same output (controllable) variables. We first define the following sets of behaviors in terms of assumptions and guarantees:

$$A = \{\sigma : \sigma \models \varphi_e\}; \quad A_i = \{\sigma : \sigma \models \varphi_{e_i}\};$$
$$G = \{\sigma : \sigma \models \varphi_s\}; \quad G_i = \{\sigma : \sigma \models \varphi_{s_i}\}.$$

Let $\mathcal{S} = (A, G \cup \neg A)$ be the contract for the global specification and $\mathcal{S}_1 = (A_1, G_1 \cup \neg A_1)$, $\mathcal{S}_2 = (A_2, G_2 \cup \neg A_2)$ be the ones for the local specifications, all in saturated form. Since any implementations of $\mathcal{S}_1$ and $\mathcal{S}_2$ are composable, contract composition using equations (5.1) and (5.2) is well defined.

We first prove that

$$\mathcal{S}_1 \otimes \mathcal{S}_2 \preceq \mathcal{S},$$

i.e., $\mathcal{S}_1 \otimes \mathcal{S}_2 = (A_{12}, G_{12})$ refines $\mathcal{S}$. By the definition of refinement, this amounts to showing that $G_{12} \subseteq G \cup \neg A$ and $A_{12} \supseteq A$. We obtain

$$\begin{aligned} G_{12} &= (G_1 \cup \neg A_1) \cap (G_2 \cup \neg A_2) \\ &= (G_1 \cap G_2) \cup (G_1 \cap \neg A_2) \cup (\neg A_1 \cap G_2) \cup (\neg A_1 \cap \neg A_2) \\ &\subseteq G \cup \neg A_2 \cup \neg A_1 \\ &= G \cup \neg(A_1 \cap A_2) \subseteq G \cup \neg A, \end{aligned} \tag{5.4}$$

where we have used that $(G_1 \cap G_2) \subseteq G$ by condition 2 in the theorem statement, and $\neg A \supseteq \neg(A_1 \cap A_2)$ (or, equivalently, $A \subseteq (A_1 \cap A_2)$) by condition 1. Moreover

$$\begin{aligned} A_{12} &= A_1 \cap A_2 \cup \neg G_{12} \\ &= A_1 \cap A_2 \cup \neg(G_1 \cup \neg A_1) \cup \neg(G_2 \cup \neg A_2) \\ &= A_1 \cap A_2 \cup (\neg G_1 \cap A_1) \cup (\neg G_2 \cap A_2) \\ &\supseteq A_1 \cap A_2 \supseteq A \end{aligned} \tag{5.5}$$

by condition 1. Equations (5.4) and (5.5) allow us to conclude that $\mathcal{S}_1 \otimes \mathcal{S}_2$ refines $\mathcal{S}$, hence any implementations of $\mathcal{S}_1$ and $\mathcal{S}_2$ satisfy the global specification.

However, for the composite contract to be well defined, we must also show that $\mathcal{S}_1 \otimes \mathcal{S}_2$ is compatible, i.e. there exists an environment that satisfies the composite contract or, equivalently, $A_{12} = \{\sigma | \sigma \models \varphi_{A_{12}}\}$ is not empty, where

$$\varphi_{A_{12}} = (\varphi_{e1} \wedge \varphi_{e2}) \vee (\varphi_{e1} \wedge \neg\varphi_{s1}) \vee (\varphi_{e2} \wedge \neg\varphi_{s2}). \tag{5.6}$$

By condition 3, there exist behaviors that make the second and third term of the above disjunction false and the first term true. Therefore, $\varphi_{A_{12}}$ is satisfiable and $\mathcal{S}_1 \otimes \mathcal{S}_2$ is compatible. $\square$

There are two sources of conservatism in distributed synthesis. The first one is due to the fact that local controllers have only local information. Therefore, even if there exists a centralized controller that realizes a global specification, there may not exist local controllers that do so. This is an inherent problem and can only be addressed by modifying the control architecture (e.g., by changing the mapping of controlled variables to controllers, by introducing new sensors, or by modifying the information flow between local controllers).

The second source of conservatism is rather computational. Even when local controllers that realize the global specification exist, it might be difficult to find them (e.g., see [71] for some undecidability results). We note that the conditions provided in Theorem 5.1.1 are only sufficient conditions. The choices of φ_{e_j} and φ_{s_j} for $j \in \{1, 2\}$ plays a role in the level of conservatism. Hence, when conditions 1 and 2 are satisfied but condition 3 is not satisfied, one can gradually refine the local specifications.

5.1.4.1 Results

For the single-line diagram in Fig. 5.3, the distributed control synthesis problem can be solved by splitting the topology into two subsystems $\mathcal{S}_1$ and $\mathcal{S}_2$. The sets $E_{\mathcal{S}_1}, S_{\mathcal{S}_1}$, and $E_{\mathcal{S}_2}, S_{\mathcal{S}_2}$ contain all environment and system variables for subsystems $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. $E_{\mathcal{S}_1}$ is composed of generators $LG1$, $APU1$ and $RG1$. $S_{\mathcal{S}_1}$ contains AC buses $LB2, RB2$, and contactors $C_{LG1,LB2}, C_{APU1,LB2}, C_{RG1,RB2}$, $C_{LB2,RB2}$. $E_{\mathcal{S}_2}$ is composed of rectifiers $LR2, RR2$ and AC buses $LB2, RB2$, while $S_{\mathcal{S}_2}$ contains DC buses $LD2, RD1$ and contactors $C_{LR2,LD2}, C_{RR2,RD1}$, $C_{LD2,RD1}$. We assume the link between AC buses and rectifier units is a solid wire.

The environment assumption $\varphi_{e_{\mathcal{S}_1}}$ for subsystem $\mathcal{S}_1$ enforces that at least one generator will always remain healthy. Environment assumption $\varphi_{e_{\mathcal{S}_2}}$ enforces that at least one rectifier unit will always remain healthy. In addition, it also assumes that both AC buses will always be powered. This is an additional guarantee $\mathcal{S}_1$ must provide to $\mathcal{S}_2$ for the distributed synthesis problem to become

Table 5.6: Topology Reliability

Topology	No. Comp.	Reliability
1	9	2×10^{-4}
2	9	4×10^{-8}
3	9	4×10^{-8}
4	13	2×10^{-14}

realizable. All other specifications remain the same as the centralized control problem.

The synthesized controllers for $\mathcal{S}_1$ and $\mathcal{S}_2$ contains 4 and 8 states, respectively. Each controller has a computation time of approximately 0.5 seconds on a Powerbook 2.2 GHz Intel Core Processor.

5.1.4.2 Reliability Results

Consider again an electric power system topology in which generators, APUs, and rectifier units may fail with a probability of $10^{-5}, 10^{-5}$, and 2×10^{-4}, respectively. The environment assumption is designed based on the overall aircraft reliability level. For different topologies, however, the synthesis problem may still be realizable for higher reliability levels. If the synthesis problem is realizable for all possible failure conditions that occur with probability greater than 10^{-x} per hour (which can then be converted to allowable environment behaviors), then the controlled system's failure rate would be less than 10^{-x} per hour. For the topologies shown in Fig. 5.2 and 5.3, we perform a line-search on x to determine the highest reliability level for which the synthesis problem is still realizable, with the specification that all DC loads must always be powered. Table 5.6 lists these levels alongside the number of components within each topology. Note that reliability level increases as the number of edges in the topology increase, which corresponds to the number of paths that exist to continuously power DC loads.

5.1.5 Real-Time Performance

Continuous-time models are implemented in Simulink, by exploiting the SimPowerSystems extension. As an example, the continuous-time model of a generator consists of a mechanical engine (turbine), a three-phase synchronous machine, in addition to the GCU, driving the field voltage of the generator. A bottom-up vertical contract specifies the range of voltage and frequencies for which this reduced-order model is an accurate representation of the actual electro-mechanical component. In addition to timing properties, our power network model allows measuring current and voltage levels at the different circuit loads. It can be discretized to speed up simulations and can seamlessly interface also with StateFlow models for the controller.

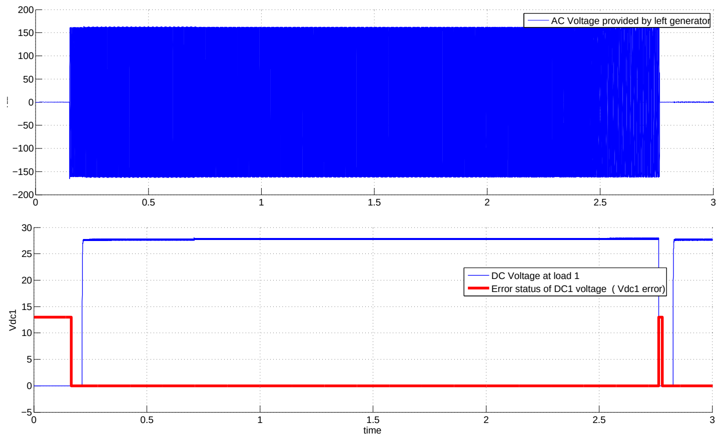


Figure 5.4: Real-time requirement violation due to a generator fault.

Figure 5.4 shows the simulated voltage V_{LD2} of bus $LD2$ in topology 3 (Fig. 5.3 b)) as a function of time, when the left generator $LG1$ fails, $T_{clk} = 20$ ms and $T_d = 40$ ms. The waveforms at the top and bottom of the figure are the voltage signals at the $LB2$ (AC) and $LD2$ (DC) buses, respectively. Both the AC and DC voltages decay to zero because of the fault. The red waveform at the bottom of the figure is interpreted as a Boolean signal, which can be high (one) or low (zero). The requirement violation suggests that the controller clock frequency should be increased to make the controlled system more "reactive" to a left generator fault.

The T_{clk} versus T_d design space is explored in Fig. 5.5 and 5.6 by leveraging a Monte Carlo based sampling scheme. The latter plot represents the amount of elapsed time τ_e^* while the DC bus voltage is out of range, i.e. for how long the requirement on the DC bus is violated. Such a violation period is then compared with the hard threshold $t_{max} = 50$ ms in Fig. 5.5, thus providing the designer with a "safe" region (marked in green in Fig. 5.5) for selecting the controller clock as a function of the contactor delay. As an example, for $T_d = 20$ ms the maximum BPCU reaction time allowed for safe operation is 45 ms.

5.2 Hardware Testbed

While in the previous section we verified controllers in simulation using *Breach* and Signal Temporal Logic, in this section we report on our simulation models and a hardware testbed for validating reactive controllers synthesized using `TuLiP` [90] and SIMULINK [83]. in order to investigate the validity of the assumptions made in controller synthesis. The particular distributed synthesis method adopted in this section follows exposition in [67] and [68] as well as from Section 5.1.4. The work

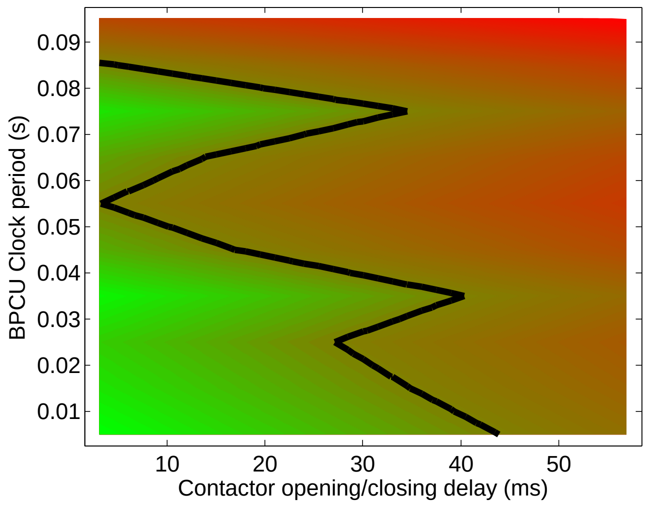


Figure 5.5: Safe clock versus contactor delay region.

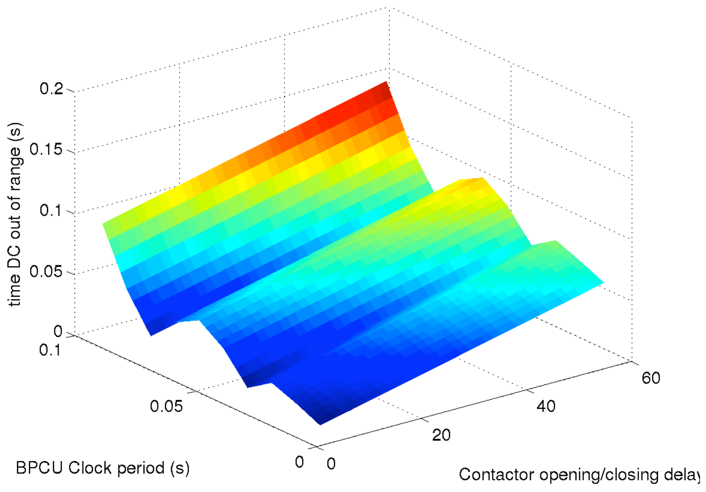


Figure 5.6: Duration of requirement violation.

described in the rest of this chapter was performed jointly with Robert Rogersten, Necmiye Ozay, and Ufuk Topcu.

University-scale testbeds for research on correct-by-construction controller synthesis are fairly limited. An advanced diagnostics and prognostics testbed is described in [75]. Some applications of this testbed to the electric power systems of spacecraft and aircraft are detailed in [58]. However, the experiments focused on diagnostic queries of the system, while our work is focused on the implementation of correct-by-construction control protocols for fault-tolerant operations. A robotics testbed implementing correct-by-construction controllers is described in [52].

The safety requirements used in simulation models and the hardware testbed follow the description in Section 4.3 and stipulate that the alternating current generators should never be paralleled and that the duration for which the bus is not powered should never exceed a certain limit. They also include the environment-related assumption that at least a subset of the generators and rectifier units must be working at all times. The simulation models were built with the physical modeling software SimPowerSystems, an extension of Simulink [83]. In order to validate the controller on the experimental hardware platform, we synthesized and tested it using `TuLiP` and SimPowerSystems, respectively. Thereafter, we investigated the validity of the assumptions used for controller synthesis on the experimental hardware platform.

An aircraft electric power system uses different voltage levels, which can broadly be divided into four categories: high-voltage AC, high-voltage DC, low-voltage AC, and low-voltage DC. The topology in Figure 5.7 is of specific interest because it is representative of some of the key features of aircraft electric power systems in simplified settings. Therefore, the hardware testbed was built based on the above mentioned topology.

5.2.1 Testbed Specifications

Consider the single-line diagram in Figure 5.7 in which environment variables are health statuses of generators and rectifier units, and controlled variables are the state of contactors. Consider also two different controller implementations: a *centralized logic* that runs the system with a single automaton and a *distributed logic* that has two different automata, one for the AC subsystem and one for the DC subsystem, running sequentially.

For the centralized logic, the specifications follow from equations (4.9) - (4.16). In particular, the environment assumptions maintains that at least one generator and one rectifier unit must always

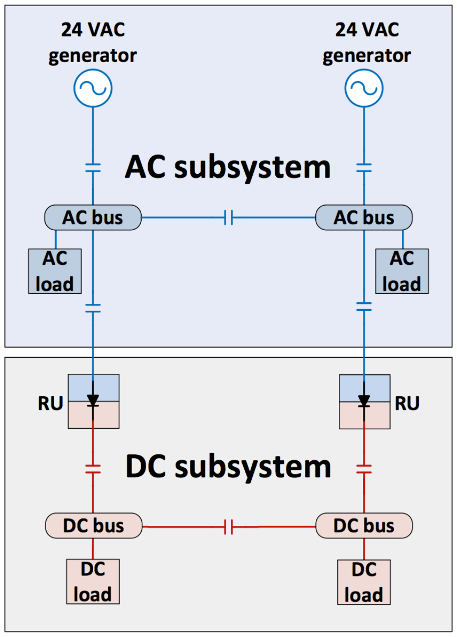


Figure 5.7: Single-line diagram of the power system testbed. Contactors are represented by double bars. The AC and DC sides of the system are separated by rectifier units (RU).

be healthy, written as

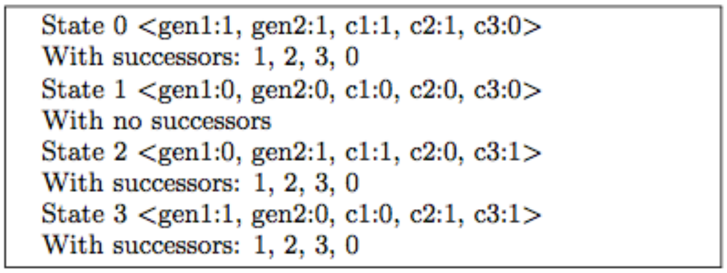
$$\Box\{((g_1 = 1) \vee (g_2 = 1)) \wedge ((r_1 = 1) \vee (r_2 = 1))\}. \tag{5.7}$$

To synthesize distributed logic, we separate the system into two subsystems, seen in Figure 5.7. The AC subsystem contains all AC components (generators, AC contactors, AC buses, and loads). The DC subsystem contains all rectifier units, DC contactors, buses, and loads. All specifications from the centralized case decompose and carry over to the distributed case. However, in order to ensure that the overall specification is realizable, we impose additional restrictions on the components located at the interface between subsystems. The rectifier units contain capacitors that can be chosen so that they create a delay T_{RU}, in which the DC buses stays powered even after that an AC bus gets unpowered.

If $T_{RU} > T$ the additional interface refinement comes in the form of a guarantee specification that all DC buses b_i, for $i \in \{1, 2\}$ will always be powered $\Box(b_i = 1)$, provided that both rectifier units stay healthy, i.e.,

$$\Box\{(r_1 = 1) \wedge (r_2 = 1)\}.$$

This guarantee is written as an environment for the DC subsystem. With this refinement, both subsystems can be synthesized independently, and the overall system specifications are satisfied



```
State 0 <gen1:1, gen2:1, c1:1, c2:1, c3:0>
With successors: 1, 2, 3, 0
State 1 <gen1:0, gen2:0, c1:0, c2:0, c3:0>
With no successors
State 2 <gen1:0, gen2:1, c1:1, c2:0, c3:1>
With successors: 1, 2, 3, 0
State 3 <gen1:1, gen2:0, c1:0, c2:1, c3:1>
With successors: 1, 2, 3, 0
```

Figure 5.8: Sample of a `TuLiP` output in two-generator and three-contactor case. The generator status variables are gen1 and gen2, and the contactor status variables are c1, c2, and c3. Each state has successors, which define where the controller can transit depending on current state. In addition, no-successor states exist.

when they are implemented together. We assume that the time a generator remains healthy is not arbitrarily short so that the AC bus powered time (i.e., the time between two intervals when AC bus is unpowered) is large enough to keep the capacitors on rectifier units charged.

5.2.2 Implementing Formal Specifications

`TuLiP` generates finite-state automata in the form of a text file that enumerates the possible states of the system and how the transitions could be carried out according to the current state. It also generates a text file that specifies environment variables (e.g., generators and rectifier units) and system variables (e.g., contactors). In order to implement the control logic in SimPowerSystems, we automatically translate these files into a Matlab-compatible script. A preliminary solution uses a Python script for this translation. A Python script generating the Matlab code is released with `TuLiP` version 0.3c under the tools directory[2].

Figure 5.8 shows an example four-state `TuLiP` generated controller for the two-generator and three-contactor case. A few lines of the auto-generated code that corresponds to this controller is shown in Figure 5.9. The auto-generated code can be inserted in SimPowerSystems as a Matlab function block. It can also be connected to the board with the code shown in Figure 5.10.

5.2.3 Design and Implementation

The single-line diagram in Figure 5.7 is a simplified notation for representing a three-phase power system. However, as described in Section 5.2.4, the power supply for the hardware testbed is not three-phase. In order to represent the installations of the sensors, circuit protection devices, and

[2] `http://tulip-control.sf.net`

```
function [c1, c2, c3] = mscript(gen1, gen2)
global state;
switch (state)
case 0:
  if gen1 == 1 and gen2 == 1 then
    state = 0; c1 = 1; c2 = 1; c3 = 0;
  else if gen1 == 0 and gen2 == 0 then
    state = 1; c1 = 0; c2 = 0; c3 = 0;
    ...
  end if
case 1:
  ...
end switch
```

Figure 5.9: Sample code generated using TuLiP controller shown in Figure 5.8.

```
global state;
while 1 do
  gen1 = readgen1();
  gen2 = readgen2();
  [c1, c2, c3] = mscript(gen1, gen2);
  writeboard(c1, c2, c3);
end while
```

Figure 5.10: Code that implements the control software running on hardware model.

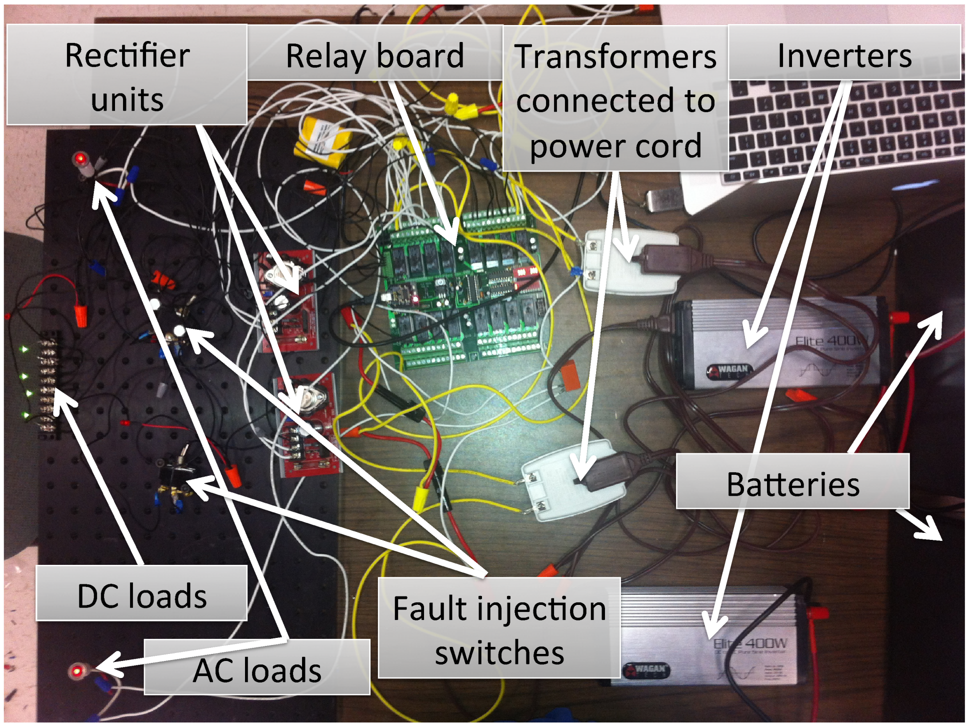
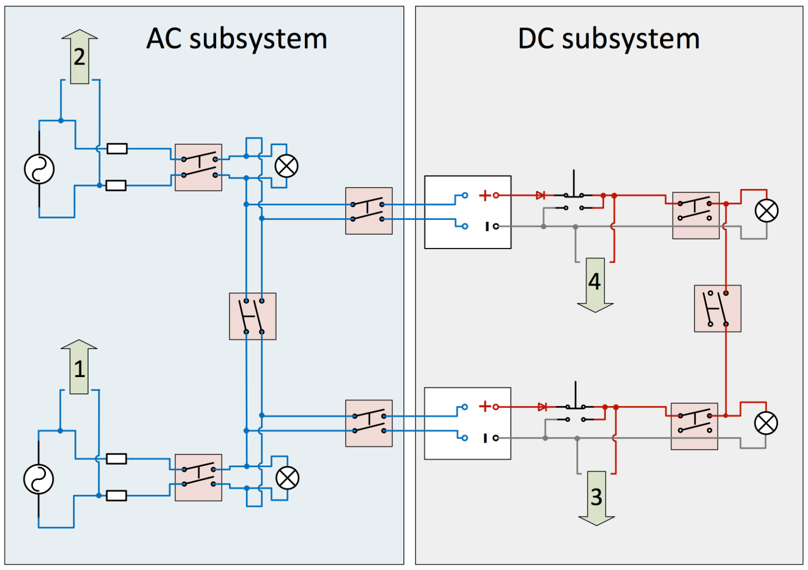


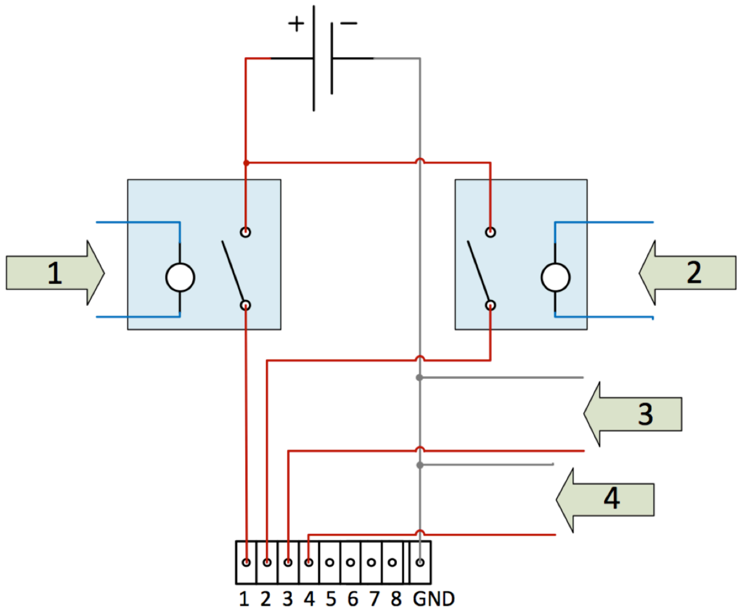
Figure 5.11: Hardware setup corresponding to the single-line diagram shown in Figure 2.1.

fault injection switches, we present a detailed schematic of the testbed in Figure 5.12. Descriptions of the components shown in Figure 5.12 are given in Figure 5.13.

The hardware testbed has two different voltage levels: 24 VAC and 2.5 VDC. The DC section is connected to the AC section by rectifier units. Aircraft contactors are designed to switch three-phase electric power with relatively high currents. Relays are generally used for switching lower currents. These operate in a similar fashion to contactors but are lighter, simpler, and less expensive. Therefore, it was more convenient to handle the switching in the hardware model with relays. It was possible to connect the control logic to the relays with the use of a relay board, which is a set of computer-controlled relays that can communicate with programming languages supporting serial communications, e.g., MATLAB. Analog-to-digital (A/D) connections on the relay board are used to monitor the system conditions. A photo of the setup is shown in Figure 5.11. The transformers in Figure 5.11 are connected to power cords; these can be unplugged to simulate a generator failure. The rectifier units are connected to a switch, which can be used to generate a fault on the DC subsystem. Next, we describe how we monitor and sense the status of generators and rectifier units.



(a) Circuit Schematic.



(b) Sensing configuration

Figure 5.12: Circuit schematic of the hardware testbed, which corresponds to the single-line diagram shown in Figure 2.1. The numbered arrows in (**a**) denote voltage sensing connections to the corresponding numbered arrows in (**b**).

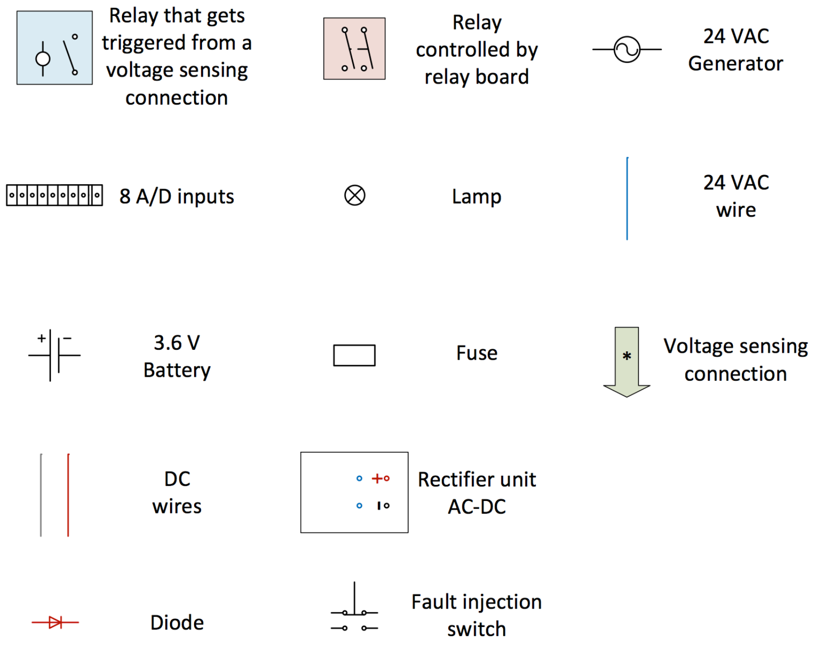


Figure 5.13: Description of the components used in Figure 5.12.

5.2.4 Generation and Circuit Protection

Each generation unit consists of a 12 V battery connected to an inverter that generates 120 VAC; that is then transformed down to 24 VAC to ensure safety. If the controller violates one of the safety requirements and connects these two sources in parallel, it would result in a short-circuit and cause excessive currents in the fuses installed next to the generators, shown in Figure 5.12(a). This observation makes it possible to monitor the correctness of the controllers at run time.

5.2.5 Sensing

The relay board needs to react consistently to faults injected into the system; this requirement implies that sensor placement, functionality, accuracy, and time delay play crucial roles in design. Two types of faults can be injected in the system, namely, rectifier unit failures and generator failures. Voltage sensing for generator failures is handled using additional relays. These relays close a 3.6 V circuit to a battery when triggered by the voltage from the transformers. If a fault occurs and a generator does not work properly, the 3.6 V circuit opens and the system reacts accordingly. The voltage sensors of the rectifier units are directly connected to the A/D ports of the relay board because the voltage can be tuned to the appropriate value using an adjustable output on the rectifier units. Figure 5.12(b) illustrates the sensing configuration on the testbed.

	T_c [ms]	T'_c [ms]
Mean	303.7	187.5
Max	333.3	234.1
Min	282.5	166.6

Table 5.7: Control cycle time, both when relay configuration changes, i.e., T_c and without any change, i.e., T'_c. The values with and without change were calculated from 20 and 250 measurements, respectively.

5.3 Experiments

We next describe the characteristics of the hardware testbed and show some preliminary test runs with different control architectures.

5.3.1 Testbed Characteristics

The first step before the implementation and testing of different controllers is characterizing the timing properties of the hardware testbed. Every relay has a time delay between the time a command is sent by the computer and the time an action (i.e., relay opening or closing) is taken, this is referred to as the *relay delay time*, T_d. Furthermore, the system has delays resulting from *control cycle times*, T_c and T'_c, defined as

$$\begin{aligned} T_c &= T_r + T_I + T_w \\ T'_c &= T_r + T_I, \end{aligned} \tag{5.8}$$

where T_r is the time it takes to read the health statuses from all of the four environment variables, T_I is the time it takes to run the logic (the time can be interpreted as the time taken to run the code shown in Figure 5.9), and T_w is the time it takes to write information to the board (see Figure 5.10). Writing information to the board is not needed in every iteration (for instance, if the system state remains the same), therefore the control cycle time also include T'_c.

The control cycle times T_c and T'_c are listed in Table 5.7. The relay delay time can be found from the board specifications and shall be less than 20 ms.

An important safety requirement in an aircraft is that a bus should never lose power for more than a certain duration, e.g., typically 50 ms. In the hardware testbed, the time for which the bus is unpowered depends on the control cycle times and the relay delay time, and because the control cycle times exceed 50 ms, we cannot use the typically specified time for which an aircraft can be unpowered. Therefore, it was necessary to adopt a suitable limit. As illustrated with two environment variables in Figure 5.10 the relay board read the health status from each environment variable in a specified order. It is therefore necessary to include a part of T'_c from the previous control cycle in this limit. The time T_I in equation (5.8) is negligible compared to T_r and T_w, the time taken to read the health status from one environment variable can therefore be approximated as $T_c'/4$. A reasonable value of an acceptable unpowered time for the hardware testbed can be

$$T \approx \max(T_d) + \max(T_c) + \frac{4-n}{4}\max(T_c'), \tag{5.9}$$

where $n \in \{1,2,3,4\}$ is the number which denotes the order of when the environment variable that is faulty is read in the code.

5.3.2 Controller Tests

Two controllers were tested, one with distributed logic and one with centralized logic. The controller with centralized logic had a 16-state automaton synthesized as explained in Section 5.2.1. The controller with distributed logic had two four-state automata that run on each subsystem. Both of these automata were synthesized in a similar fashion to the 16-state controller.

If the environment-related assumption is violated, the controller may end up in a state with no outgoing transitions, referred to as the *no-successor* state. The environment-related assumptions for the testbed are expressed in equation (5.7) of Section 5.2.1. A violation of equation (5.7) results in the controller entering a no-successor state, which happens when both generators or both rectifier units are faulty. If a centralized controller senses that both rectifier units are faulty, the whole system stops working because a no-successor state has been reached. This is not the case when distributed logic is used, because the AC system continues working even if the DC environment assumption is violated and the DC part reaches a no-successor state. The distributed logic implementation has two different automata that represent the logic, one for each subsystem, with coupling between them. However, the distributed logic is centralized in that it consists of single control software running on a single computer and communicating with the hardware through a single channel.

Figure 5.14 shows the voltage measurement for the centralized 16-state controller. The measurement was taken on the AC bus when the generator, which health status is read at second place ($n = 2$ in equation (5.9)) of the four environment variables in the code, was switched off and then on again. The generator was switched off at $t = 2.83\,\text{s}$, at which point the bus becomes unpowered. The second vertical line from the left indicates when the controller reacts and power up the bus using the other generator, which happens at $t = 3.1\,\text{s}$. The generator was switched on again at $t = 3.73\,\text{s}$; this was accompanied by a discernible change in the sine curve. Once a generator is switched on again after a fault, the time for which the bus is without power is not noticeable because the controller

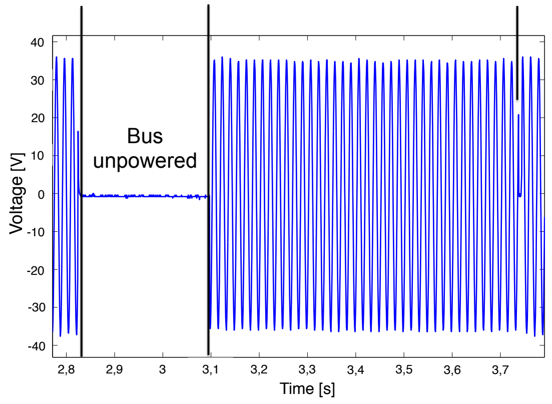


Figure 5.14: Bus voltage measurement when a generator is switched off and then turned back on. The first vertical line indicates the fault, the second vertical line is when the controller reacts, and the third line is when the generator is turned back on.

	Bus-unpowered time [ms]
Mean	333.9
Max	414.9
Min	232.7

Table 5.8: Time for which bus is unpowered after a fault is injected. These values are calculated using measurements from 10 fault injections.

sends simultaneous commands to two relays.

The measured bus-unpowered times are listed in Table 5.8, which show a maximum value of $T_{max} = 414.9\,\text{ms}$. An acceptable unpowered time when $n = 2$ and $\max(T_d) = 20\,\text{ms}$ can be calculated with equation (5.9). It follows that $T \approx \max(T_d) + \max(T_c) + \frac{1}{2}\max(T_c') = 470.35\,\text{ms}$ and hence, $T_{max} < T$. We used a digital storage oscilloscope (Rigol DS1052E 50 MHz) for measurements. The measurement data are imported into MATLAB for analysis and to estimate the unpowered times.

5.4 Conclusions

We have applied a rigorous platform-based methodology to the design of an aircraft electric power system. Our flow consists of three main phases: topology synthesis, control synthesis, and simulation-based design space exploration and verification. To express system requirements, we adopt different formalisms supported by specialized synthesis and analysis frameworks. To generate the system topology, we cast a mixed integer-linear program that minimizes the overall cost while satisfying a set of connectivity, power flow and reliability requirements, expressed in terms of linear arithmetic constraints on Boolean variables and probabilistic constraints. To generate a correct-by-construction controller for a given topology, we leverage results from reactive synthesis from linear temporal logic specifications. We then refine these LTL specifications into signal-temporal logic constructs to assess the real-time system performance and explore the design space at a lower abstraction level, based on high fidelity behavioral models. Our compositional approach uses contracts to guarantee independent implementability of system topology and control, since both topology synthesis and control synthesis rely on a consistent set of models and design constraints.

As a future work, we will extend our control synthesis algorithms to support richer formal languages (e.g., timed logic, branching logic), continuous-time specifications and continuous dynamics (e.g., transients, network and communication delays). Moreover, we plan to investigate techniques for automatic generation of local contracts for the synthesis of distributed and hierarchical control architectures.

As an extension, we plan to build more representative SIMULINK models that match the hardware characteristics (e.g., voltage ranges, timing delays), and synthesize controllers that are consistent with the timing characterization of the hardware. SIMULINK has embedded controller simulation add-ons with sensor communication networks that can make simulation models more realistic. Future work can test out "truly" distributed controllers in simulation, which we were unable to perform in hardware due to a single relay board. A simulation model that takes into account the implementation platform would better reflect software challenges.

Chapter 6

Dynamic State Estimation

This chapter explores the design problem of state estimation based on sensor placement for a given electric power system topology. Section 6.1 provides a brief overview of the state estimation problem for electric power systems. Sections 6.2 introduces the problem setup and mathematical concepts and notation used throughout the rest of the chapter. Section 6.3 proposes a mathematical strategy to determine the state of the system from sensor measurements and gives a worst-case performance bound for the strategy. Sections 6.4 and 6.5 present the problem implementation and shows results for example circuits (i.e., topologies). This chapter is joint work with Quentin Maillet and Necmiye Ozay.

6.1 Overview

Previous work in electric power system state estimation has focused on static, centralized estimation problems with continuous states. We perform discrete state estimation using active control of switches within the electric power system in a distributed control architecture. The system reconfigures itself through a set of controllable contactors (i.e., electrically controlled switches). Once reconfigured, new sensor measurements are taken to gain more information about the unknown state. We adaptively sequence switching actions by use of a greedy strategy that maximizes the one-step expected uncertainty reduction. By exploiting recent results in adaptive submodularity [34, 47], we provide theoretical bounds for the worst-case performance of the greedy strategy for a uniform probability distribution along states. Such dynamic state estimation techniques have been proposed in the context of Markov jump linear systems [14], information gathering in robotics [61, 84], active hypothesis testing [65], and active learning [37]. To the best of our knowledge, these ideas have not been applied before in electric power system state estimation and fault diagnosis problems.

A critical assumption in recent work [67, 88, 92] is that the high-level reactive control protocol has an accurate knowledge of the system states, including fault states so that it can reroute the power accordingly. An expensive, hence undesirable, solution to achieve accurate state estimates is to equip the system with a large number of sensors. The more sensors present in the system, makes maintenance more difficult, as well as adds more weight to the aircraft. Software, however, is cheaper and more amenable to change than hardware. The goal of this chapter is to obtain high-accuracy state estimates with a limited number of sensors by utilizing software-based dynamic estimation strategies. We are particularly interested in detecting and localizing faults in the system. It is common to use discrete models for fault diagnosis [80]. Therefore, continuous values of voltage and current, as well as health statuses of components in the system are discretized before performing state estimation. A discrete framework is also well-suited for combining the proposed estimation strategy with control synthesis results as discussed in Chapter 3.

6.2 Problem Setup

6.2.1 General Problem Description

Consider an aircraft electric power system topology, which can be represented by a graph data structure $G = (\mathcal{N}, \mathcal{E})$. Let Figure 6.1 be the representative circuit topology. The set of nodes $\mathcal{N} = \{n_1, \ldots, n_{n_n}\}$ in the graph representation contains the following components: generators ($\mathcal{G}$), rectifier units ($\mathcal{R}$), and voltage sensors ($\mathcal{S}$). The set of edges $\mathcal{E} = \{e_1, \ldots, e_{n_e}\}$ contains all contactors (and solid wire links) between components. The status of contactors in $\mathcal{E}$ can either be *open* or *closed*. A node corresponding to a rectifier unit has no outgoing edges on the AC side and no incoming edges on the DC side to reflect the fact that they contain a diode (i.e., power is unidirectional). The rest of the edges in the graph are bidirectional.

Elements in the set of generators $\mathcal{G} \subseteq \mathcal{N}$ and rectifier units $\mathcal{R} \subseteq \mathcal{N}$ are uncontrollable, and can take values of

1. *Unhealthy* (i.e., the component is online but outputting a voltage not in admissible range);
2. *Healthy* (i.e., the component is online and outputting the correct voltage);
3. *Offline* (i.e., no power output, open circuit).

Sensor measurements read from $\mathcal{S} \subseteq \mathcal{N}$, then, will depend on the status of generators, rectifier units, and contactors. We define a *live path* between two components if there exists a simple path

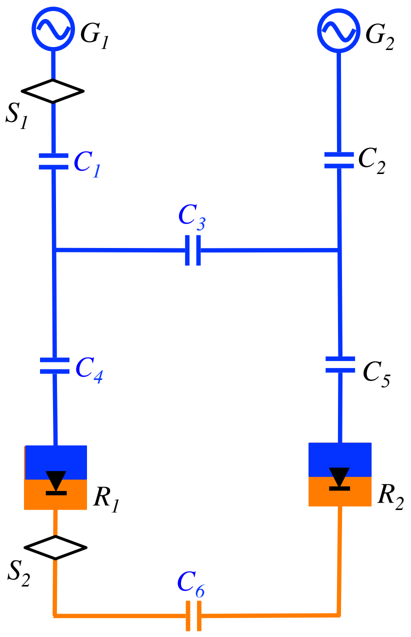


Figure 6.1: A single-line diagram of a simple circuit with AC and DC components.

in the graph G that connects the two nodes corresponding to these components, there is no offline component along the path (including end nodes), and the contactors along this path are all closed.

The reading from a sensor s can then take the following values

1. *Improper voltage*: if there is a live path between sensor s and some generator $g \in \mathcal{G}$ (not offline by definition of live path), but either g or rectifier $r \in \mathcal{R}$ along such a path is unhealthy;

2. *Admissible voltage*: for all $g \in \mathcal{G}$ that have a live path to s, both g and all rectifier units along such paths are healthy;

3. *No voltage*: there exists no live path between sensor s and any generator $g \in \mathcal{G}$.

The state x of the system is defined as a valuation on all components $n \in \mathcal{G} \cup \mathcal{R}$ and uncontrollable contactors $e \in \mathcal{C}' \subseteq \mathcal{C}$. We define Ω as the set of all states, i.e all the different valuations of the components. The state x is unknown and hence modeled as a random variable X that can only be determined by sensor measurements that are mapped back to a set of possible states in which the circuit may be.

On top of the circuit and sensing topology is a distributed control architecture with a dynamic state estimation mechanism. We assume that one of the embedded controllers is responsible for dynamic state estimation, hereafter referred to as the fault detection controller. The fault detection controller is able to control a subset $\mathcal{C} \setminus \mathcal{C}'$ of contactors (e.g., those labelled with blue in Fig. 6.1). The overall goal is to design a strategy the fault detection controller runs to adaptively estimate the discrete state of the circuit by taking "actions" (i.e., closing and opening controllable contactors), and then reading voltage sensor measurements.

6.2.2 Mathematical Formulation

In this section we introduce the relevant notations used throughout the rest of the paper and mathematically formulate the estimation problem.

The state X of the circuit is unknown and modeled as a random variable. Data on component types and reliability levels can be used to build a probability measure $\mathbb{P}[x]$ on Ω. At the beginning of the state estimation process the system is in the state $x_0 \in \Omega$. We assume that faults in the system are independent, and that x_0 remains fixed during the estimation process. This is a reasonable assumption because the timescale of the estimation process is meant to be much smaller than the failure rates of the components and the timescales of the other controllers in the system.

For the controllable subset of contactors, there exists a set $\mathcal{V}$ of actions v that can be performed and a set $\mathcal{Y}$ of measurements y that can be observed. For an action $v \in \mathcal{V}$, $y = \mu(v, x)$ is the unique outcome of performing action v if the system is in the state x. The actions $\{v_0, ..., v_t\}$ performed and outcomes $\{y_0, ..., y_t\}$ observed up until step t are represented by the partial realization $\psi_t = \{(v_i, y_i)\}_{i \in \{0,...,t\}}$. Given two partial realizations ψ_t and $\psi_{t'}$, we say that ψ_t is a subrealization of $\psi_{t'}$ if $\psi_t \subseteq \psi_{t'}$. At each step t, the probability measure $\mathbb{P}[x]$ can be updated by conditioning it on ψ_t to obtain $\mathbb{P}[x \mid \psi_t]$.

We are interested in an estimation process adaptively eliminating "invalid" states to get to the actual state x_0. We define $D(y, v)$, with $y = \mu(v, x_0)$, to be the set of states $x \in \Omega$ that are indistinguishable from x_0 under the action v. Formally, $D(\mu(v, x_0), v) = \{x \in \Omega \mid \mu(v, x) = \mu(v, x_0)\}$. We further extend this concept by defining $h(v_{0:t}, x_0)$, the set of states that produce the same set of outcomes $\{\mu(v_0, x_0), \ldots, \mu(v_t, x_0)\}$ as x_0 under the same set of actions $\{v_0, \ldots, v_t\}$. In the remainder of the paper, we use S_t as a shorthand for $h(v_{0:t}, x_0)$. If, at step t, we perform a new action $v' \notin \psi_t$, there exists a recursive relation between the two sets of states:

$$h(v_{0:t} \cup \{v'\}, x_0) = h(v_{0:t}, x_0) \cap D(\mu(v', x_0), v'), \tag{6.1}$$

which leads immediately to

$$S_t = \cap_{i \in \{0,...,t\}} D(\mu(v_i, x_0), v_i). \tag{6.2}$$

As only intersections are taken, the order of actions v_i does not matter.

To represent the uncertainty in the state estimate, we define an objective function $f : 2^{\mathcal{V}\times\mathcal{Y}} \times \Omega \to \mathbb{R}_+$ that maps the set of actions $A \subseteq \mathcal{V}$ under state x_0 to reward $f(A, x_0)$. A strategy π is a function from partial realizations to actions such that $\pi(\psi_t)$ is the action v_{t+1} taken by π when observing ψ_t. We denote $\tilde{\mathcal{V}}(\pi, x_0) \subseteq \mathcal{V}$ the set of all the actions performed under the strategy π, the state of the system being x_0. In the general case, $\tilde{\mathcal{V}}(\pi, x_0) \neq \mathcal{V}$.

The fault detection controller is assigned a budget $k \ll |\mathcal{V}|$, the number of steps within which the estimation process should terminate. The system is initially in the state x_0, which is fixed and unknown, and the controlled contactors are in some initial configuration v_0. Initial configuration v_0 and the corresponding measurement y_0 constitute ψ_0. Then, for $i = 1, \ldots, k$, we consider the following process:

$$v_i = \pi(\psi_{i-1}) \tag{6.3a}$$

$$y_i = \mu(v_i, x_0) \tag{6.3b}$$

$$\psi_i = \psi_{i-1} \cup (v_i, y_i). \tag{6.3c}$$

Equations (6.3a) - (6.3c) represent the decision making, measurement, and update in the estimation process, respectively.

The goal is to reduce the uncertainty of X represented by the probability distribution $\mathbb{P}[x]$ through performing k actions. To that end, the following reward function is considered:

$$f(v_{0:k}, x_0) = -\mathbb{P}[S_k] = -\sum_{x\in S_k} \mathbb{P}[x]. \tag{6.4}$$

The behavior driven by the maximization of f is to remove as much probability mass from Ω as possible in k steps. It is also worth noting that when the underlying probability distribution on Ω is uniform, f is just proportional to the size of S_k and so maximizing f is equivalent to minimizing the number of indistinguishable states.

The goal of estimation is to find the strategy that allows the "best expected estimate" for the state, i.e, the strategy π^* s.t.

$$\pi^* \in \arg\max_{\pi} \mathbb{E}[f(\tilde{\mathcal{V}}(\pi, X), X)], \tag{6.5}$$

subject to $|\tilde{\mathcal{V}}(\pi, x)| \leqslant k$ for all x, and with expectation taken with respect to $\mathbb{P}[x]$.

6.3 Strategy

In this section, we describe the algorithm used to solve the state estimation problem and give performance guarantees on the worst-case execution.

6.3.1 Greedy strategy

To determine the optimal strategy for the fault detection controller, one should plan ahead for k steps, yet complexity scales up exponentially with k. To address the problem efficiently we develop a greedy strategy that selects, at each step, the action maximizing the expected one-step gain in uncertainty reduction. The greedy strategy uses the information ψ_t gathered through the previous measurements and the probability measure $\mathbb{P}[x \mid \psi_t]$ on the set S_t. This probability is computed using a classic Bayesian update:

$$\mathbb{P}[x \mid \psi_t] = \frac{\mathbb{P}[\psi_t \mid x]\ \mathbb{P}[x]}{\mathbb{P}[\psi_t]}, \quad \forall\ x \in \Omega. \tag{6.6}$$

As the measurement process is deterministic, for a given $x \in \Omega$ we have $\mathbb{P}[x \mid \psi_t] = \mathbb{1}_{\{x \in S_t\}}$, meaning that $\mathbb{P}[x \mid \psi_t] = 1$ if x belongs to S_t, and $\mathbb{P}[x \mid \psi_t] = 0$ otherwise. From (6.6) we then get:

$$\mathbb{P}[x \mid \psi_t] = \begin{cases} \frac{\mathbb{P}[x]}{\mathbb{P}[\psi_t]} & \forall\ x \in S_t \\ 0 & elsewhere \end{cases} \tag{6.7}$$

The term $\mathbb{P}[\psi_t]$ is the same for all x. It is a normalization coefficient that can be computed using $\sum_{x \in S_t} \mathbb{P}[x \mid \psi_t] = 1$ to obtain

$$\mathbb{P}[\psi_t] = \sum_{x \in S_t} \mathbb{P}[x]. \tag{6.8}$$

At step t, the strategy consists of choosing the next action v_{t+1} that maximizes the gain in uncertainty reduction. Our measure of uncertainty comes from the value of the function f, established in Eq. (6.4), and therefore the benefit is expressed in terms of the change in f as we choose the action v. Consistent with our goal, we choose to maximize in mean the benefit at each step, the expectation taken with respect to the updated probability measure $\mathbb{P}[x \mid \psi_t]$. We obtain the greedy strategy:

$$v_{t+1} \in \arg\max_{v \in \mathcal{V}} \mathbb{E}[f(v_{0:t} \cup \{v\}, X) - f(v_{0:t}, X) \mid \psi_t]. \tag{6.9}$$

6.3.2 Performance Guarantees

Greedy strategies in general can perform arbitrarily badly [8]. However, by exploiting recent results on adaptive submodularity, we give a lower bound on the performance of the proposed strategy. For a brief overview of adaptive submodularity and related definitions, see Section 6.6. We next show that the function f defined in Eq. (6.4), is adaptive monotone and adaptive submodular (Def. 6.6.2 and 6.6.3).

Proposition 6.3.1 *The function f defined in Eq. (6.4) is adaptive monotone.*

Proof: Given an action $v \in \mathcal{V}$ and partial realization ψ_t at step t, we need to show the expected marginal benefit $\Delta(v|\psi_t)$ (see Def. 6.6.1) is nonnegative. For the cost function f, $\Delta(v|\psi_t)$ can be written as:

$$\Delta(v \mid \psi_t) = \mathbb{E}[f(v_{0:t}, X) \mid \psi_t] - \mathbb{E}[f(v_{0:t} \cup \{v\}, X) \mid \psi_t]. \tag{6.10}$$

By Eq. (6.7), we get

$$\Delta(v \mid \psi_t) = \sum_{x \in h(v_{0:t}, x_0)} \mathbb{P}[x|\psi_t]\ \phi(x), \tag{6.11}$$

with

$$\phi(x) = \sum_{\tilde{x} \in h(v_{0:t}, x)} \mathbb{P}[\tilde{x}]\ - \sum_{\tilde{x} \in h(v_{0:t} \cup \{v\}, x)} \mathbb{P}[\tilde{x}]. \tag{6.12}$$

By Eq. (6.1), $h(v_{0:t} \cup \{v\}, x)$ is a subset of $h(v_{0:t}, x)$ for every $x \in \Omega$. Thus, $\phi(x) \geqslant 0$, all the terms in the sum in Eq. (6.11) are non-negative, and $\Delta(v|\psi_t) \geqslant 0$. □

Proposition 6.3.2 *The function f defined in Eq. (6.4) is adaptive submodular.*

Proof: Given in Appendix 6.6.2. □

Theorem 6.3.3 *For any true state $x_0 \in \Omega$, the uncertainty reduction achieved in k steps by the greedy strategy given in Algorithm 1 is no worse than $(1 - 1/e)$ of what can be achieved in k steps by any other strategy, including the best possible strategy.*

Proof: Follows directly from Propositions 6.3.1 and 6.3.2 and Theorem 6.6.4 given in Section 6.6. □

6.4 Implementation

In this section, we give implementation details on the dynamic estimator employing the greedy strategy on some typical aircraft electric power system topologies. In order to reduce online computation, the inverse mapping from sensor measurements to compatible states of the circuit is conducted offline. Additionally, we propose some abstraction rules to reduce the size of the circuit as well as computation time.

6.4.1 Implementation Details

The overall estimation process is summarized in Algorithm 1.

Algorithm 1 Adaptive greedy strategy

```
Input: Probability measure $\mathbb{P}[x]$ on $\Omega$, number of actions to perform $k$. The system is in the state $x_0 \in \Omega$,
    fixed and unknown, and the controlled contactors are in some configuration $v_0$.
Output: Knowledge of the system $\psi_k$ gathered after the $k$ actions taken under the strategy $\pi_{greedy}$
 1: Take the measurement $y_0 = \mu(v_0, x_0)$.
 2: $\psi_0 = \{(v_0, y_0)\}$
 3: for t $\in \{1, \dots, k\}$ do
 4:     $v_t = \pi_{greedy}(\psi_{t-1})$
 5:     Perform action $v_t$
 6:     Take the measurement $y_t = \mu(v_t, x_0)$
 7:     $\psi_t = \psi_{t-1} \cup \{v_t, y_t\}$
 8:     $S_t = S_{t-1} \cap D(y_t, v_t)$
 9:     Compute $\mathbb{P}[x \mid \psi_t]$ (Bayesian update)
10: end for
11: return $(\psi_k, S_k, \mathbb{P}[x \mid \psi_k])$
```

In this algorithm, some items can be precomputed to improve run time. In particular, the inverse mapping from sensor measurements to compatible states does not have a closed form expression and the computation of the inverse map involves searching for paths on the graph $G = (\mathcal{N}, \mathcal{E})$ representing the circuit topology. Therefore, for all measurements $y \in \mathcal{Y}$ and all actions $v \in \mathcal{V}$, the sets $D(y, v)$ of states consistent with the action-measurement pairs (v, y) are computed offline to achieve a faster implementation. This collection is then accessed on the fly to significantly reduce the computation time as it is the most costly part of the algorithm.

Assumptions about the components and circuit can be easily incorporated in our framework. In particular, because these circuits are designed to achieve certain reliability levels, one common assumption is that at least one generator and one rectifier unit are online (delivering correct or improper voltage). These assumptions render certain states impossible, which are removed from the initial state set Ω.

6.4.2 Model Reduction Via Abstraction

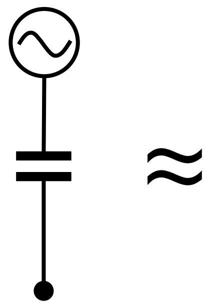
Although the greedy strategy provides an efficient way (with performance guarantees) to solve the dynamic state estimation problem, the offline computation can be very demanding. In particular, the number of possible states is exponential in the number of components whose states are being estimated. Therefore, for complex circuit topologies, the offline computation of the sets $D(y, v)$ for all $y \in \mathcal{Y}$ and all actions $v \in \mathcal{V}$ is expensive. In this section, we give a set of rules that can be recursively applied to reduce the size of the circuit by clustering certain components together into metacomponents.

Components (generators, rectifier units, contactors) are connected through their ports to form the circuit, and sensors are placed on some of these ports. The main reduction idea is that when two uncontrolled components are connected together and there is no sensor on their internal connecting port, some of the individual states of the components may become indistinguishable from what can be measured with the available sensors. Therefore, they can be treated as a single basic component, called a metacomponent, having the same global overall behavior. It is then possible to hierarchically estimate the system state, first by estimating the state of the metacomponent, and then mapping this state to possible states of individual components forming the metacomponent. When running the greedy algorithm on the reduced circuit, the probabilities of metacomponent states should be adjusted accordingly to ensure a lossless abstraction.

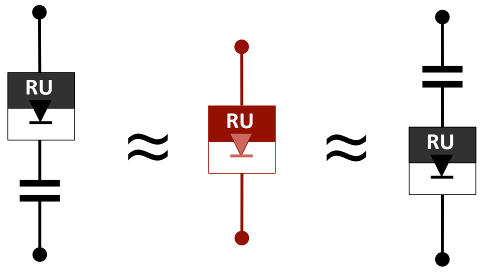
The rules we use to simplify the circuits are summarized in Fig. 6.2. Figure 6.2(a), for example, shows how the combination of generator and contactor can be abstracted into a single “generator" metacomponent. For the original combination of components, the contactor can either be open (o) or closed (c), and the generator can either be healthy (h), unhealthy (u), or offline (o). Thus, the set Ω has six possible states, represented as a tuple of contactor status and generator health: $x_1 = (c, h)$, $x_2 = (c, u)$, $x_3 = (c, o)$, $x_4 = (o, h)$, $x_5 = (o, u)$, and $x_6 = (o, o)$. The “generator" metacomponent, however, has three possible states, corresponding to healthy, unhealthy, and offline: $\tilde{x}_1 = h$, $\tilde{x}_2 = u$, and $\tilde{x}_3 = o$. These metacomponent states can be mapped back to the corresponding original components, such that $\tilde{x}_1 = \{x_1\}$, $\tilde{x}_2 = \{x_2\}$, and $\tilde{x}_3 = \{x_3, x_4, x_5, x_6\}$.

6.5 Examples

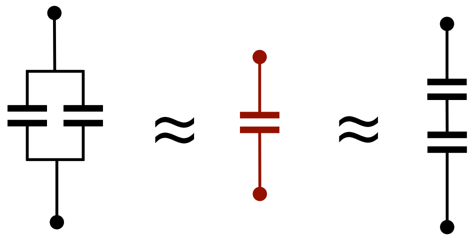
To assess the performance of the greedy strategy, we have systematically tested the greedy strategy on diverse circuits that are representative of the standard and simple circuits used in electric power



(a) Generator Metacomponent.



(b) Rectifier Unit Metacomponent.



(c) Contactor Metacomponent.

Figure 6.2: Metacomponents used for abstraction. In terms of possible external behaviors (i.e., what can be measured from the external ports), two-component circuit units (shown in black) are equivalent to the single component units (shown in red).

systems.

In many cases it is not possible to completely eliminate the uncertainty on the state of the system when there is a limited number of sensors. In order to evaluate the performance of the greedy strategy, we compare it with a brute force strategy, which exhaustively tries every action $v \in V$. At each step states inconsistent with measurements are eliminated, and the strategy terminates when no action is left. No strategy that tries $k < |V|$ actions can perform better than the brute force strategy. Although the brute force strategy is not practically applicable, as $|V|$ can be very large, it gives an upper bound on achievable performance, and can be used as a benchmark. Overall test methodology is summarized in Algorithm 2.

Algorithm 2 Test methodology

```
1: for $\phi_0 \in \Omega$ do
2:     Set the whole circuit (controlled part as well as uncontrolled part) in the state $\phi_0$
3:     Run the strategy tested (Greedy or brute force strategy)
4:     Record the computation time and the value of $f$ at the end for statistics.
5: end for
```

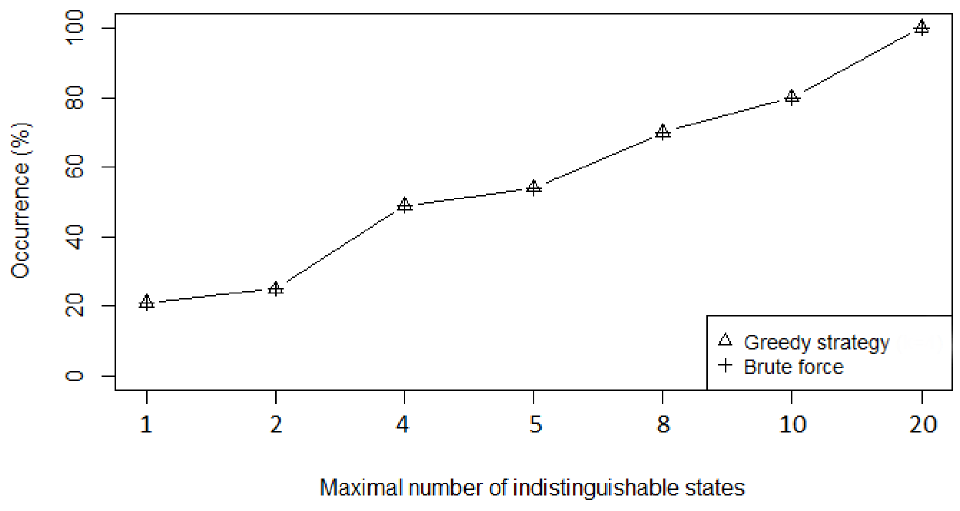


Figure 6.3: Performance comparison between greedy and brute-force strategies.

6.5.1 Small Circuit Tests

The small circuit test configuration, shown in Fig. 6.1 is comprised of 12 components. Six components are unknown $(G_1, G_2, R_1, R_2, C_2, C_5)$, 4 contactors are controlled (C_1, C_3, C_4, C_6) and two voltage sensors are available (S_1 and S_2). Taking into consideration reliability assumptions on faults, the size of the state-space generated is 1600. A more precise description of the actual hardware circuit can be found in [77]. On this particular example, with four controlled contactors, the brute force strategy performs the $|V| = 2^4 = 16$ actions. Both strategies have been run on the same MacBook Pro 2.2 GHz Intel Core. As shown in Figure 6.3, the greedy strategy with a horizon length of $k = 6$ performs as well as the brute force strategy, i.e., the value of the objective function f at the end of the 6 steps using the greedy strategy is the same as after the brute force strategy with 16 steps.

6.5.1.1 Average Execution Time

The average execution time for the greedy strategy is shown in Fig. 6.4. Online computation for the next best action executes on the order of milliseconds, whereas the offline computation for database set $\mathcal{D}$ takes 30 seconds for the small circuit.

6.5.1.2 Average Remaining States

Fig. 6.5 shows the distribution of the values of f after $k = 6$ steps for the greedy strategy. Beginning with 1600 possible states, the greedy strategy reduces the number of candidates to less than 20 states in all the cases. In 50% of the cases, there are 4 states or fewer that are still indistinguishable after

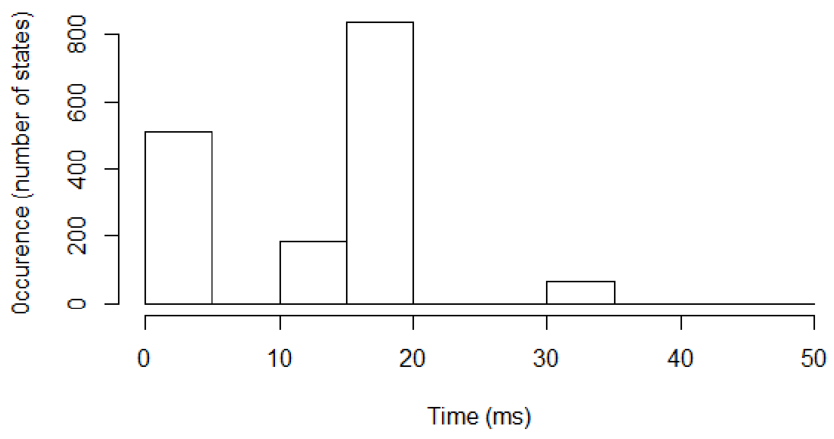


Figure 6.4: Histogram of execution time for the greedy strategy.

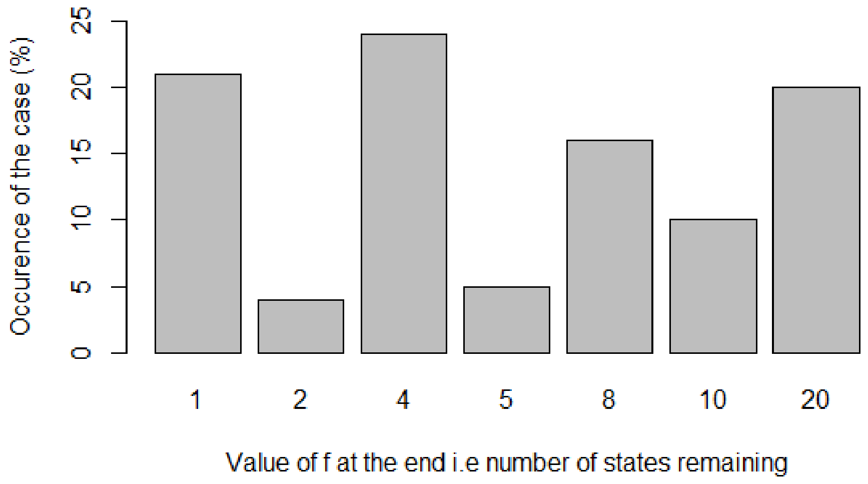


Figure 6.5: Distribution of the number of indistinguishable states with the greedy strategy

k steps. Using this metric of performance to compare greedy and brute force strategies, we can see on Fig. 6.3 that the greedy strategy performs as well as the brute force. The graph shows the same figures as Fig. 6.5, but re-shaped for an easier comparison. A point at coordinates (n, m) simply means that in $m\%$ of the cases, there are n or less indistinguishable states after the strategy (greedy or brute force) terminates.

6.5.2 Large Circuit Tests

In this section we test the greedy strategy on a larger circuit. This topology is representative of more-electric aircraft power distribution systems with multiple generators and demonstrates how abstraction can reduce the offline computation time. The circuit topology is shown in Fig. 6.6. Contactors controlled by fault detection controller are depicted in blue. Applying the lossless abstraction method established in 6.4.2 leads to a reduced circuit in which four uncontrolled contactors are eliminated. Comparing the offline computation for the full and reduced circuit, the that abstraction reduces the offline computation time by an order of magnitude (from 4000 seconds to 400 seconds).

We have also tested the greedy strategy on this circuit for a subset of Ω. Namely, we have selected a standard functioning configuration of the contactors and considered all the possible valuations of the other components, hence creating a subset of Ω. On this subset, the greedy strategy only with $k = 5$ actions again performs as well as the brute force strategy. Results for the large circuit tests are similar to those from the small circuit test, and thus figures are omitted.

6.6 Background Results in Submodularity

6.6.1 Definitions

We give some definitions and results on adaptive submodularity that follows the exposition provided in [34] and [47]. Notations used here were defined in Subsections 6.2 and 6.2.2.

Definition 6.6.1 *Given an objective function f, an action $v \in \mathcal{V}$, and a partial realization ψ_t, $\Delta(v|\psi_t)$ is the conditional expected marginal benefit of v conditioned on having observed ψ_t, defined as*

$$\Delta(v|\psi_t) \doteq \mathbb{E}[f(v_{0:t} \cup \{v\}, X) - f(v_{0:t}, X)|\psi_t],$$

and the expectation taken with respect to $\mathbb{P}[x|\psi_t]$.

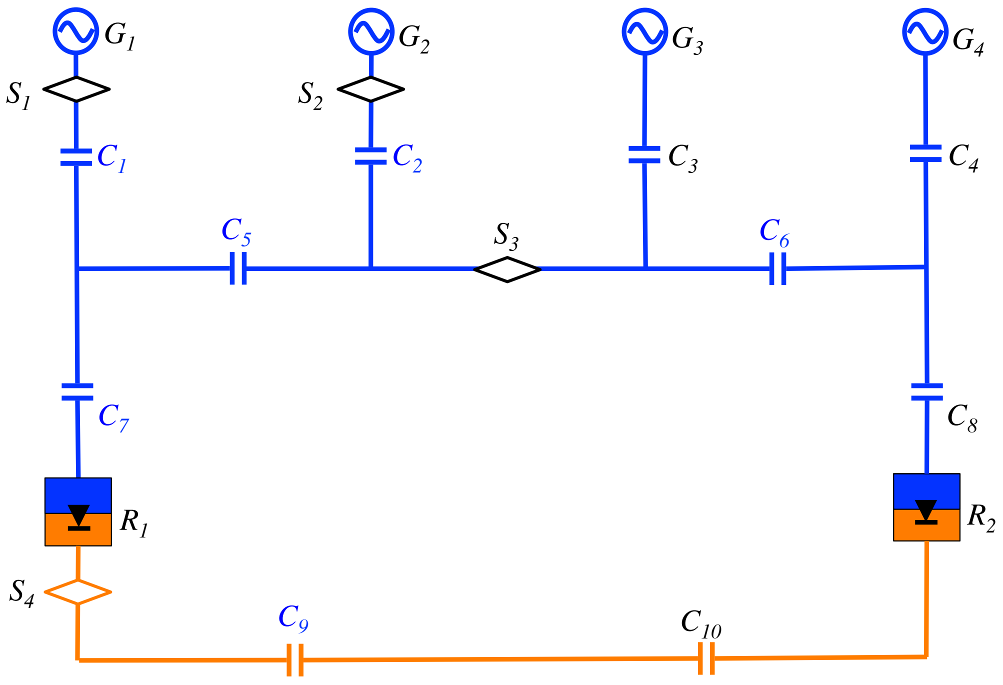


Figure 6.6: A single-line diagram of a larger circuit with AC and DC components.

Definition 6.6.2 *The function $f : 2^{V \times \mathcal{Y}} \times \Omega \to \mathbb{R}_+$ is adaptive monotone with respect to distribution $\mathbb{P}[x]$ if the conditional expected marginal benefit of any action is nonnegative. Thus, for all $v \in \mathcal{V}$ and ψ_t with $\mathbb{P}[\psi_t] > 0$,*

$$\Delta(v|\psi_t) \geq 0.$$

Definition 6.6.3 *The function $f : 2^{V \times \mathcal{Y}} \times \Omega \to \mathbb{R}_+$ is adaptive submodular with respect to distribution $\mathbb{P}[x]$ if the conditional expected marginal benefit of any fixed action v does not increase as more actions are performed and measurements are taken. Thus, f is adaptive submodular with respect to distribution $\mathbb{P}[x]$ if for all $\psi_t, \psi_{t'}$ such that ψ_t is a subrealization of $\psi_{t'}$, and for all $v \in \mathcal{V} \backslash \{v_0, \ldots, v_{t'}\}$,*

$$\Delta(v|\psi_t) \geq \Delta(v|\psi_{t'}).$$

The adaptive greedy algorithm, a generalization of the greedy algorithm [47], is a strategy that selects the action maximizing the conditional expected marginal benefit, conditioned on outcomes from all previous actions.

Theorem 6.6.4 (Theorem 1.14 in [47]) *Let π_l^{greedy} be a greedy strategy run for l iterations (so* *that it selects l actions). Let π_k^* be any policy selecting at most k actions for any realization x. Then,*

$$f_{avg}(\pi_l^{greedy}) \geq \left(1 - e^{-l/k}\right) f_{avg}(\pi_k^*),$$

where $f_{avg}(\pi) \doteq \mathbb{E}[f(\tilde{V}(\pi, X), X)]$ is the expected reward of π.

In particular, by setting $k = l$ we see that the greedy strategy selecting k items step by step obtains at least $(1 - 1/e)$ of the value of the optimal strategy that selects k items step by step.

6.6.2 Proofs

We first state a lemma that will be useful in the proof.

Lemma 6.6.5 *The function $b : \mathbb{R}^Y \to \mathbb{R}$, defined as*

$$b(\tau_1, \tau_2, \ldots, \tau_Y) = \sum_{i=1}^{Y} \tau_i - \frac{\sum_{i=1}^{Y} \tau_i^2}{\sum_{i=1}^{Y} \tau_i}, \tag{6.13}$$

is increasing on the positive orthant, i.e., $b(\tau_1, \tau_2, \ldots, \tau_Y) \geq b(s_1, s_2, \ldots, s_Y)$ if $\tau_i \geq s_i \geq 0$ for all $1 \leq i \leq Y$.

Proof: Note that because b is symmetric, i.e., permutation invariant with respect to its arguments, it is enough to show that it is increasing in one of its arguments. Let $k_1 \doteq \sum_{i=2}^{Y} \tau_i$ and $k_2 \doteq \sum_{i=2}^{Y} \tau_i^2$. Define $\tilde{b}(x) \doteq b(x, \tau_2, \ldots, \tau_Y) = k_1 + x - \frac{k_2 + x^2}{k_1 + x}$. The partial derivative of b with respect to state x is $\partial\tilde{b}/\partial x = \frac{k_1^2 + k_2}{(k_1+x)^2}$, which is non-negative by definitions of k_1 and k_2. $\square$

Now, we are ready to prove Proposition 6.3.2.

Consider two partial realizations ψ_t and $\psi_{t'}$ s.t $\psi_t \subseteq \psi_{t'}$ and the corresponding sets S_t and $S_{t'}$. Fix an action $v \in \mathcal{V} \backslash v_{0:t'}$. To prove adaptive submodularity, $\Delta(v, \psi_t)$ can be expressed as a function dependent on the size of S_t. We examine the variation of Δ between S_t and $S_{t'}$.

Since the probability measure is non-uniform and can take values in some set $\{p_1, \ldots, p_N\}$, we define the subsets of Ω where $\mathbb{P}[x]$ is constant: $F_n = \{x \in \Omega \mid \mathbb{P}[x] = p_n\}$ for $n \in \{1, \ldots, N\}$. The collection $F_{1:N}$ is trivially a partition of Ω. It is possible to show that the sets $\{D(y, v^*) \cap F_n | y \in \mathcal{Y}, n \in 1 : N\}$ form a partition of Ω and thus a partition of S_t.

Let $\alpha_{n,y} \doteq S_t \cap D(y, v) \cap F_n$. Then for all $x \in \alpha_{n,y}$, we have

$$\mu(v, x) = y \text{ and } \mathbb{P}[x] = p_n. \tag{6.14}$$

By Eq. (6.8) , we get a new expression for $\mathbb{P}[\psi_t]$:

$$\mathbb{P}[\psi_t] = \sum_{x \in S_t} \mathbb{P}[x] = \sum_{y \in \mathcal{Y}} \sum_{n \in 1:N} p_n |S_t \cap D(y, v) \cap F_n|. \tag{6.15}$$

Let $\tau_y \doteq \sum_{n \in 1:N} p_n |\alpha_{n,y}|$. Then, conditional probabilities on F_n can be rewritten as

$$\forall x \in F_n, \; \mathbb{P}[x \mid \psi_t] = \frac{p_n}{\sum_{y \in \mathcal{Y}} \tau_y}. \tag{6.16}$$

We then separately compute the two terms in Eq. (6.10). First term becomes:

$$\mathbb{E}[f(v_{0:t}, X) \mid \psi_t] = \sum_{x_0 \in S_t} \mathbb{P}[x_0 \mid \psi_t] \sum_{x \in h(v_{0:t}, x_0)} \mathbb{P}[x]. \tag{6.17}$$

For $x_0 \in S_t$, $h(v_{0:t}, x_0) = S_t$, we obtain

$$\mathbb{E}[f(v_{0:t}, X) \mid \psi_t] = \sum_{y \in \mathcal{Y}} \tau_y. \tag{6.18}$$

For the second term in Eq. (6.10), we first get

$$\begin{aligned} f(v_{0:t} \cup \{v\}, x) &= \sum_{\tilde{x} \in h(v_{0:t}, x) \cap D(\mu(v,x), x)} \mathbb{P}[\tilde{x}] \\ &= \tau_{\mu(v,x)}. \end{aligned}$$

From Eq. (6.14) and Eq. (6.16), we obtain:

$$\begin{aligned} \mathbb{E}[f(v_{0:t} \cup \{v\}, X) \mid \psi_t] &= \sum_{x \in S_t} f(v_{0:t} \cup \{v\}, x) \mathbb{P}[x \mid \psi_t] \\ &= \sum_{n \in 1:N} \sum_{y \in \mathcal{Y}} \sum_{x \in \alpha_{n,y}} \tau_y \frac{p_n}{\sum_{z \in \mathcal{Y}} \tau_z} \\ &= \sum_{y \in \mathcal{Y}} \frac{\tau_y}{\sum_{z \in \mathcal{Y}} \tau_z} \sum_{n \in 1:N} p_n |\alpha_{n,y}| \\ &= \sum_{y \in \mathcal{Y}} \frac{\tau_y^2}{\sum_{z \in \mathcal{Y}} \tau_z}. \end{aligned}$$

Finally, putting the two terms of Eq. (6.10) leads to

$$\Delta(v|\psi_t) = b(\tau_1, \tau_2, \ldots, \tau_Y) = \sum_{i=1}^{Y} \tau_i - \frac{\sum_{i=1}^{Y} \tau_i^2}{\sum_{i=1}^{Y} \tau_i}, \tag{6.19}$$

where $Y \doteq |\mathcal{Y}|$.

This expression of $\Delta(v|\psi_t)$ in terms of the variables τ_i is similar for the partial realization $\psi_{t'}$; the only change is the set S_t, which is represented in the function b by a different value of the τ_i denoted τ_i'. Since $\psi_t \subseteq \psi_{t'}$ and $S_{t'} \subseteq S_t$, τ_i and τ_i' satisfy $\tau_i' \leqslant \tau_i$ for all i.

Therefore, adaptive submodularity is equivalent to showing that b is increasing on the positive orthant, and Lemma 6.6.5 concludes the proof.

6.7 Conclusions and Future Work

The current dynamic state estimation problem assumes that a single fault scenario, among a set of possible scenarios, occurs and remains static throughout the entire greedy strategy implementation. Given sensor measurements, the greedy strategy outputs a set of possible system states, or a localized state if possible. Currently, placement of sensors on the circuit topology is a given. By changing the number and locations of sensors, however, it may be possible to improve state estimation performance.

Given sensor and state knowledge, we can synthesize a reactive controller by framing the synthesis problem as a two-player game with incomplete information. In [76], Reif shows that games with incomplete information (i.e., sets of states that cannot be localized given sensor measurements) can be transformed into games of perfect information. This is done by a powerset construction of states, similar to subset construction in finite state automata. In this formulation, the worst-case scenario is an exponential blow-up in state space. However, the sets of states are restricted given the sensor knowledge from the dynamic state estimation problem.

Future work will integrate results from partial information games and synthesis of control protocols that can react to dynamically changing faults with dynamic state estimation in order to configure an optimal sensor placement. Exploring this design space and trade-offs therein are likewise topics for future work.

Chapter 7

Conclusions and Future Work

7.1 Summary

This thesis addressed ways to design a system topology, formally and automatically specify requirements, and synthesize reactive control protocols using an aircraft electric power system as a representative application area. While current systems engineering relies on text-based specifications and manual design, we combine formal methodologies from computer science and control in order to create easier, more efficient, and verifiable ways to develop future control systems.

We demonstrated how text-based specifications can be converted into a temporal logic specification language using a representative single-line diagram as an example. Given a set topology for an electric power system and a set of system requirements formalized in linear temporal logic, we automatically synthesized a control protocol for an electric power system on a more-electric aircraft. The controller reacts to changes in the environment and is guaranteed, by construction, to satisfy the desired properties even in the presence component (i.e., generator) failures. We synthesized a centralized controller where statuses of all components are known, as well as distributed and decentralized controllers by refining the overall system specifications. This refinement involves additional assumptions and guarantees between subsystem interfaces (i.e., specifications on the components that interact with other subsystems).

In specifying formal requirements, we have introduced a tool to automatically convert high-level specifications into a formal specification language. We addressed techniques for synthesis of discrete-variable, untimed and discrete-time control protocols. Scenario-based diagrams were shown to be an additional way to specify system requirements. By utilizing a subset of diagrams referred to as assume-guarantee live sequence charts, additional semantics allow for these visual-based charts to be converted into GR(1) specifications and used to synthesize controllers. The limitations of timed synthesis tools has been discussed, and timed specifications have been converted into linear temporal logic in order to utilize the capabilities of current tools.

In the area of design space exploration, we have applied methodology to the design of an aircraft electric power system consisting of three main phases: topology synthesis, control synthesis, and simulation-based design space exploration and verification. Central and controllers previously synthesized are verified through real-time simulation as well as implemented on a hardware testbed. By characterizing real-time hardware constraints, we can further refine our synthesis formulation in order to better characterize timing requirements or evaluate how well a discretized-time problem can be translated into real-time simulations.To express system requirements, we adopt different formalisms supported by specialized synthesis and analysis frameworks. To generate the system topology, we cast a mixed integer-linear program that minimizes the overall cost while satisfying a set of connectivity, power flow and reliability requirements, expressed in terms of linear arithmetic constraints on Boolean variables and probabilistic constraints. To generate a correct-by-construction controller for a given topology, we leverage results from reactive synthesis from linear temporal logic specifications. We then refine these LTL specifications into signal-temporal logic constructs to assess the real-time system performance and explore the design space at a lower abstraction level, based on high fidelity behavioral models. Our compositional approach uses contracts to guarantee independent implementability of system topology and control, since both topology synthesis and control synthesis rely on a consistent set of models and design constraints.

Finally, we perform discrete state estimation using active control of switches within the electric power system in a distributed control architecture. We formulated a greedy strategy implementation, which, for a given set of sensor measurements, outputs a set of possible system states, or a localized state if possible. We provide a worst-performance bound for the greedy strategy, and detail abstraction methods in order to reduce the size of the problem state space.

7.2 Future Work

Timing specifications in the electric power system problem are addressed with the use of clocks by way of additional counter variables. This discretization of time further adds to the difficulties arising from state space explosion. While capable of synthesizing large-scale timed systems, UPPAAL-TIGA [11] is limited in the types of specifications and number of specifications it can handle. The efficiency of these timed verification tools, is likewise still dependent on the number of clocks used in the model. Other tools may be useful, and we are currently examining the ease and expressibility of other alternatives.

Another topic of future work is determining what level of abstraction is needed for modeling, design, and specification. Control of the power quality from generators is considered at a continuous level of abstraction. Load management and load shedding are considered at a discrete low-level of abstraction. Both of these problems, although at different levels of abstraction, should be interfaced with the primary distribution problem discussed earlier. The effects of transient voltages, significant changes in addition or removal of loads, should be investigated within this framework. Due to the nature of the electric power system design problem, continuous dynamics were able to be abstracted away to a high-level logic problem. Other systems, such as the air-management system on aircraft, are more highly coupled to dynamics. The influence of network effects, particularly at lower levels of abstraction, is an area of interest.

To that effect, future work will also examine different the use of timed temporal logics in order to capture specifications which may not be expressible using LTL or computation tree logic (CTL). UPPAAL-TIGA can synthesize controllers with respect to timed specifications formalized in timed computation tree logic (TCTL). Specifications, however, are limited to fragments of TCTL (i.e., no nested quantifiers). We also plan to capture these specifications using LTL, with additional system variables, in order to utilize the full range of LTL, which can be used in conjunction with other solvers, such as Lily [40]. As a future work, we will extend our control synthesis algorithms to support richer formal languages (e.g., timed logic, branching logic), continuous-time specifications and continuous dynamics (e.g., transients, network and communication delays). The hardware testbed can likewise be extended. Not only will more components be added, but also controllers synthesized from other solvers, such as Lily, can be tested.

Furthermore, we also plan to directly use the domain-specific language and tool to automatically convert assume-guarantee live sequence charts into specifications and synthesize a controller. Further extensions include broadening the domain-specific language to include user-specific requirements that may not be included in the high-level general specifications described earlier. With these kinds of languages, however, the functionality is still limited to specific scenarios or types of requirements. Generation specification from assume-guarantee live sequence charts are performed manually. While tools, such as PlayGo [57] exist to synthesize controllers from a given live sequence chart, this does not allow integration of live sequence chart specification with other system requirements. In other words, assume-guarantee live sequence charts can be used for synthesis, but the control logic is based solely on the charts themselves. We plan to extract the LTL specifications from PlayGo and integrate them with `TuLiP`. In addition, we are also exploring the use of PlayGo and domain-specific languages for distributed controller protocols. Namely, how to distribute a given topology among subsystems and generate interface specifications such that the overall system is realizable. Lastly, the problem of network effects, including transients and delays, has been mostly ignored or abstracted away within this problem formulation. Introducing specifications encompassing network effects would be an additional feature for a domain-specific language.

In addition, given sensor and state knowledge, we can synthesize a reactive controller by framing the synthesis problem as a two-player game with incomplete information (i.e., sets of states that cannot be localized given sensor measurements) by transforming them into games of perfect information. This is done by a powerset construction of states, similar to subset construction in finite state automata. In this formulation, the worst-case scenario is an exponential blow-up in state space. However, the sets of states are restricted given the sensor knowledge from the dynamic state estimation problem. Future work will integrate results from partial information games and synthesis of control protocols that can react to dynamically changing faults with dynamic state estimation in order to configure an optimal sensor placement. Exploring this design space and trade-offs therein are likewise topics to consider.

Finally, we plan to investigate techniques for automatic generation of local contracts for the synthesis of distributed and hierarchical control architectures. This can be integrated into the domain-specific language, thus automatically generating contracts as well as specifications. These techniques are not limited to the scope of electric power systems, and we plan to demonstrate the usability to other application areas.

Bibliography

[1] IBM ILOG CPLEX Optimizer, February 2012.

[2] A. Abur and A. Exposito. *Power system state estimation: theory and implementation*, volume 24. CRC, 2004.

[3] R. Alur and T. A. Henzinger. A really temporal logic. In *Symposium on Foundations of Computer Science*, pages 164–169, 1989.

[4] K. An, A. Trewyn, A. Gokhale, and S. Sastry. Model-driven performance analysis of reconfigurable conveyor systems used in material handling applications. In *Cyber-Physical Systems (ICCPS), 2011 IEEE/ACM International Conference on*, pages 141 –150, April 2011.

[5] E. Asarin, A. Donzé, O. Maler, and D. Nickovic. Parametric identification of temporal properties. In *Runtime Verification*, pages 147–160, 2011.

[6] C. Baier, J.-P. Katoen, et al. *Principles of model checking*, volume 26202649. MIT Press, 2008.

[7] P. Baker, S. Loh, and F. Weil. Model-driven engineering in a large industrial context: Motorola case study. In *Model Driven Engineering Languages and Systems*, pages 476–491. Springer, 2005.

[8] J. Bang-Jensen, G. Gutin, and A. Yeo. When the greedy algorithm fails. *Discrete Optimization*, 1(2):121–127, 2004.

[9] A. Barrett, R. Knight, R. Morris, and R. Rasmussen. Mission planning and execution within the mission data system. In *4th International Workshop on Planning and Scheduling for Space, Darmstadt, Germany, June 23-25, 2004*. Pasadena, CA: Jet Propulsion Laboratory, National Aeronautics and Space Administration, 2004., 2004.

[10] R. Behjati, T. Yue, S. Nejati, L. Briand, and B. Selic. Extending sysml with aadl concepts for comprehensive system architecture modeling. In *Modelling Foundations and Applications*, pages 236–252. Springer, 2011.

[11] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. Larsen, and D. Lime. Uppaal-tiga: Time for playing games! In *Computer Aided Verification*, pages 121–125. Springer, 2007.

[12] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. Henzinger, and K. Larsen. Contracts for systems design. *Proc. IEEE*, to appear 2013.

[13] A. Bhave, B. Krogh, D. Garlan, and B. Schmerl. View consistency in architectures for cyber-physical systems. In *Cyber-Physical Systems (ICCPS), 2011 IEEE/ACM International Conference on*, pages 151 –160, April 2011.

[14] L. Blackmore, S. Rajamanoharan, and B. Williams. Active estimation for jump markov linear systems. *Automatic Control, IEEE Transactions on*, 53(10):2223–2236, 2008.

[15] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Saar. Synthesis of reactive (1) designs. *Journal of Computer and System Sciences*, 78(3):911–938, 2012.

[16] A. Bose and K. Clements. Real-time modeling of power networks. *Proceedings of the IEEE*, 75(12):1607–1622, 1987.

[17] C. P. Cavas. Launch of navy's newest aircraft carrier delayed @ONLINE, May 2013.

[18] N. Clark. Boeing delays deliveries of 787 @ONLINE, 2007.

[19] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

[20] E. M. Clarke and J. M. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys (CSUR)*, 28(4):626–643, 1996.

[21] J. S. Cloyd. Status of the united states air force's more electric aircraft initiative. *Aerospace and Electronic Systems Magazine, IEEE*, 13(4):17–22, 1998.

[22] T. B. Corporation. Boeing revises 787 first flight and delivery plans; adds schedule margin to reduce risk of further delays @ONLINE, 2008.

[23] T. B. Corporation. Stopwatch reset âĂŞ again âĂŞ for mrh90 @ONLINE, May 2013.

[24] E. Davidson, S. McArthur, and J. McDonald. A toolset for applying model-based reasoning techniques to diagnostics for power systems protection. *Power Systems, IEEE Transactions on*, 18(2):680 – 687, may 2003.

[25] A. Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *Proc. Int. Conf. Comput.-Aided Verification*, pages 167–170, Berlin, Heidelberg, 2010. Springer-Verlag.

[26] B. Dutertre and L. D. Moura. The yices smt solver. Technical report, 2006.

[27] R. Ehlers. Symbolic bounded synthesis. In *Computer Aided Verification*, pages 365–379. Springer, 2010.

[28] R. Ehlers. Experimental aspects of synthesis. *Electronic Proceedings in Theoretical Computer Science*, 50, 2011.

[29] R. Ehlers. Unbeast: Symbolic bounded synthesis. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 272–275, 2011.

[30] E. A. Emerson. Temporal and modal logic. *Handbook of theoretical computer science*, 2:995–1072, 1990.

[31] E. Filiot, N. Jin, and J.-F. Raskin. Antichains and compositional algorithms for ltl synthesis. *Formal Methods in System Design*, 39(3):261–296, 2011.

[32] S. Friedenthal, A. Moore, and R. Steiner. *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2011.

[33] A. Galton. *Temporal logics and their applications*. Academic Press London, 1987.

[34] D. Golovin and A. Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*, 42(1):427–486, 2011.

[35] A. Gonzalez, R. Morris, F. McKenzie, D. Carreira, and B. Gann. Model-based, real-time control of electrical power systems. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 26(4):470–482, 1996.

[36] D. Gorinevsky, S. Boyd, and S. Poll. Estimation of faults in dc electrical power system. In *American Control Conference, 2009. ACC'09.*, pages 4334–4339. IEEE, 2009.

[37] A. Guillory and J. Bilmes. Average-case active learning with costs. In *Algorithmic Learning Theory*, pages 141–155. Springer, 2009.

[38] G. J. Holzmann. The model checker spin. *Software Engineering, IEEE Transactions on*, 23(5):279–295, 1997.

[39] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and reasoning about systems.* Cambridge University Press, 2004.

[40] B. Jobstmann and R. Bloem. Lily - a linear logic synthesizer, 2006.

[41] B. Jobstmann and R. Bloem. Optimizations for ltl synthesis. In *Formal Methods in Computer Aided Design, 2006. FMCAD'06*, pages 117–124. IEEE, 2006.

[42] B. Jobstmann, S. Galler, M. Weiglhofer, and R. Bloem. Anzu: A tool for property synthesis. In *Computer Aided Verification*, pages 258–262. Springer, 2007.

[43] E. Kang, E. Jackson, and W. Schulte. An approach for effective design space exploration. In *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems*, pages 33–54. Springer, 2011.

[44] K. Keller, K. Swearingen, J. Sheahan, M. Bailey, J. Dunsdon, K. Przytula, and B. Jordan. Aircraft electrical power systems prognostics and health management. In *Aerospace Conference, 2006 IEEE*, pages 12–pp. IEEE, 2006.

[45] M. Kloetzer and C. Belta. A fully automated framework for control of linear systems from temporal logic specifications. *Automatic Control, IEEE Transactions on*, 53(1):287–297, 2008.

[46] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Syst.*, 2(4):255–299, 1990.

[47] A. Krause and D. Golovin. Submodular function maximization. In *Tractability: Practical Approaches to Hard Problems (to appear)*. Cambridge University Press, 2012.

[48] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Temporal-logic-based reactive mission and motion planning. *Robotics, IEEE Transactions on*, 25(6):1370–1381, 2009.

[49] H. Kugler, D. Harel, A. Pnueli, Y. Lu, and Y. Bontemps. Temporal logic for scenario-based specifications. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 445–460. Springer, 2005.

[50] R. Kumar, E. G. Mercer, and A. Bunker. Improving translation of live sequence charts to temporal logic. *Electronic Notes in Theoretical Computer Science*, 250(1):137–152, 2009.

[51] T. Laengle, T. C. Lueth, and U. Rembold. A distributed control architecture for autonomous robot systems. *Series in Machine Perception and Artificial Intelligence*, 21:384–402, 1995.

[52] M. Lahijanian, M. Kloetzer, S. Itani, C. Belta, and S. B. Andersson. Automatic deployment of autonomous cars in a robotic urban-like environment. In *IEEE Intl. Conf. on Robotics and Automation*, pages 2055–2060, Kobe, Japan, 2009.

[53] L. Liu, K. Logan, D. Cartes, and S. Srivastava. Fault detection, diagnostics, and prognostics: software agent solutions. *Vehicular Technology, IEEE Transactions on*, 56(4):1613–1622, 2007.

[54] P. Madhusudan and P. Thiagarajan. Distributed controller synthesis for local specifications. *Automata, languages and programming*, pages 396–407, 2001.

[55] O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Formal Modeling and Analysis of Timed Systems*, pages 152–166, 2004.

[56] S. Maoz and Y. Saar. Assume-guarantee scenarios: semantics and synthesis. In *Model Driven Engineering Languages and Systems*, pages 335–351. Springer, 2012.

[57] S. Maoz and Y. Saar. Counter play-out: Executing unrealizable scenario-based specifications. 2013.

[58] O. J. Mengshoel, A. Darwiche, K. Cascio, M. Chavira, S. Poll, and S. Uckun. Diagnosing faults in electrical power systems of spacecraft and aircraft. In *Innovative Applications of Artificial Intelligence Conference*, pages 1699–1705, Chicago, IL, 2008.

[59] M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4):316–344, December 2005.

[60] R. Michalko. Electrical starting, generation, conversion and distribution system architecture for a more electric vehicle, 10 2008.

[61] L. Mihaylova, T. Lefebvre, H. Bruyninckx, K. Gadeyne, and J. De Schutter. A comparison of decision making criteria and optimization methods for active robotic sensing. *Numerical Methods and Applications*, pages 316–324, 2003.

[62] I. Moir and A. Seabridge. *Aircraft Systems: Mechanical, Electrical, and Avionics Subsystems Integration.* AIAA Education Series, 2001.

[63] A. Monticelli. Electric power system state estimation. *Proceedings of the IEEE*, 88(2):262–282, 2000.

[64] M. Mukund. From global specifications to distributed implementations. *Synthesis and Control of Discrete Event Systems*, pages 19–34, 2002.

[65] M. Naghshvar and T. Javidi. Active sequential hypothesis testing. *CoRR*, abs/1203.4626, 2012.

[66] P. Nuzzo, A. Sangiovanni-Vincentelli, X. Sun, and A. Puggelli. Methodology for the design of analog integrated interfaces using contracts. *IEEE Sensors J.*, 12(12):3329–3345, Dec. 2012.

[67] N. Ozay, U. Topcu, and R. M. Murray. Distributed power allocation for vehicle management systems. In *Proc. IEEE Conference on Decision and Control and European Control Conference*, pages 4841–4848, 2011.

[68] N. Ozay, U. Topcu, R. M. Murray, and T. Wongpiromsarn. Distributed synthesis of control protocols for smart camera networks. In *Cyber-Physical Systems (ICCPS), 2011 IEEE/ACM International Conference on*, pages 45–54. IEEE, 2011.

[69] R. S. Peak, R. M. Burkhart, S. Friedenthal, M. W. Wilson, M. Bajaj, and I. Kim. Simulation-based design using sysml part 2: Celebrating diversity by example. In *INCOSE intl. symposium, San Diego*, 2007.

[70] N. Piterman, A. Pnueli, and Y. Saar. Synthesis of reactive (1) designs. In *Verification, Model Checking, and Abstract Interpretation*, pages 364–380. Springer, 2006.

[71] A. Pneuli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Foundations of Computer Science, 1990. Proceedings., 31st Annual Symposium on*, pages 746 –757 vol.2, oct 1990.

[72] A. Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.

[73] A. Pnueli. Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends. *Current trends in Concurrency*, pages 510–584, 1986.

[74] A. Pnueli, Y. Saar, and L. Zuck. Jtlv: A framework for developing verification algorithms. In *Computer Aided Verification*, pages 171–174. Springer, 2010.

[75] S. Poll, A. Patterson-hine, J. Camisa, D. Garcia, D. Hall, C. Lee, et al. Advanced diagnostics and prognostics testbed. In *International Workshop on Principles of Diagnosis*, pages 178–185, 2007.

[76] J. H. Reif. The complexity of two-player games of incomplete information. *Journal of computer and system sciences*, 29(2):274–301, 1984.

[77] R. Rogersten, H. Xu, N. Ozay, U. Topcu, and R. M. Murray. An aircraft electric power distribution testbed for reactive control protocols. In *Proceedings of the 16th international conference on Hybrid systems: computation and control*, HSCC '13, 2013.

[78] J. Rosero, J. Ortega, E. Aldabas, and L. Romeral. Moving towards a more electric aircraft. *Aerospace and Electronic Systems Magazine, IEEE*, 22(3):3–9, 2007.

[79] J. Rumbaugh, I. Jacobson, and G. Booch. *Unified Modeling Language Reference Manual*. Pearson Higher Education, 2004.

[80] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. C. Teneketzis. Failure diagnosis using discrete-event models. *Control Systems Technology, IEEE Transactions on*, 4(2):105–124, 1996.

[81] A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone. Taming Dr. Frankenstein: Contract–based design for cyber–physical systems. In *Conf. Decision and Control*, Dec. 2011.

[82] S. Schewe and B. Finkbeiner. Bounded synthesis. *Automated Technology for Verification and Analysis*, pages 474–488, 2007.

[83] SIMULINK. 2011.

[84] A. Singh, A. Krause, C. Guestrin, W. Kaiser, and M. Batalin. *Efficient planning of informative paths for multiple robots.* Carnegie Mellon University, School of Computer Science, Machine Learning Department, 2006.

[85] M. Sinnett. 787 no-bleed systems: Saving fuel and enhancing operational efficiency @ONLINE.

[86] S. Sohail and F. Somenzi. Safety first: A two-stage algorithm for ltl games. In *Formal Methods in Computer-Aided Design, 2009. FMCAD 2009*, pages 77–84. IEEE, 2009.

[87] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli. Design space exploration of network processor architectures. *Network Processor Design: Issues and Practices*, 1:55–89, 2002.

[88] T. Wongpiromsarn, U. Topcu, and R. Murray. Formal synthesis of embedded control software: Application to vehicle management systems. In *Proceedings of the AIAA Infotech Aerospace Conference*, 2011.

[89] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Receding horizon control for temporal logic specifications. *Automatic Control, IEEE Transactions on*, 2012.

[90] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. M. Murray. Tulip: a software toolbox for receding horizon temporal logic planning. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*, pages 313–314. HSCC, 2011.

[91] T. Wu, S. Bozhko, G. Asher, and D. Thomas. Fast functional modelling of the aircraft power system including line fault scenarios. In *Power Electronics, Machines and Drives (PEMD 2010), 5th IET International Conference on*, pages 1–7. IET, 2010.

[92] H. Xu, U. Topcu, and R. M. Murray. A case study on reactive protocols for aircraft electric power distribution. In *Proc. IEEE Conference on Decision and Control*, 2012.

[93] S. Y. Yin, Y. Yang, X. W. Miao, and T. D. Zhao. Sysml-based safety analysis of thrust reverser. *Journal of Aerospace Power*, 3:005, 2011.

[94] J. Zumberge, J. Wolff, K. McCarthy, and T. O'Connell. Integrated aircraft electrical power system modeling and simulation analysis. *SAE Technical Paper*, pages 01–1804, 2010.