Synthesis of reactive controllers for vehicular electric networks via SMT solving

Jing Ji
Undergraduate Student
Department of Electrical Engineering and Computer Science
University of Michigan

Faculty Advisor: Necmiye Ozay

ABSTRACT

The safety and performance of modern aircraft highly depends on the reliability of electric power systems and subsystems, including flight-critical ones. Inevitably, some components in these systems may fail operating and causing danger. In this project, we find a control strategy that senses the health status of various components and reacts to possible faults by reconfiguring the circuit via switches to make sure critical loads are always powered. We have investigated the use of satisfiability modulo theory (SMT) solvers as an efficient means for correct-by-construction synthesis of reactive controllers for aircraft power system. To make a circuit more readable in a computer sense, we convert the circuit into a directed graph and apply assumption-guarantee contracts on it, namely some basic rules and requirement on the circuit, which can then be formatted into a SMT problem.

I. INTRODUCTION

The fundamental idea of reconfiguring an electric power circuit is that the controller takes in data collected from sensors, and outputs a sequence of commands on contactors to switch them on and off, in order to make sure that all essential buses are properly powered on. This leads to several problems: How to recognize a circuit diagram in a computer? What rules must we follow during the reconfigure process? How to find such strategy to reconfigure the circuit under various situations?
Fortunately, the last problem can be reduced to a Boolean satisfiability problem and SAT solvers can efficiently solve such Boolean satisfiability problems. The constraints are, however, that some components in the circuit may have more than two possible states, and thus is not well applicable using a Boolean satisfiability solver. Nevertheless, in this summer project, we restrict our attention to the case that all components can only acquire two states.
A circuit diagram contains plenty of information, but for our project purpose, we only need some of the basics. Converting a circuit diagram into a directed graph makes it more readable for computer and easier to process. The reason why we use a directed graph instead of just graph will be explained in the following II section, part B.
The middle question is somewhat challenging. Intuitively, we can come up with rules like: all essential buses, those to which all flight critical loads are connected, should be powered on at all time, or open all contactors that are right next to the component if it is not functioning correctly. But are these two rules adequate? Is there no more? And how can we apply them on the circuit? These new questions will be discussed in detail in the following II section, part C.

II. PRELIMINARIES

A. Components

In this project, we only consider some of the very basic components in an aircraft electric power circuit: Generator, auxiliary power unit (APU) (provides energy for functions other than propulsion), transformer rectifier unit (TRU), bus, and contactor (or switch). Loads are ignored since they are connected to buses and thus it would be adequate to consider only buses. A simple representative circuit is shown in Figure 1.
We denote generator by G, APU by A, TRU by T, bus by B, and contactor by C.
All buses can be powered on or unpowered, corresponding to Boolean 1 and 0.
Contactors are the only components that are controllable in the circuit; they can acquire values of close and open, represented in binary as 1 and 0. Any other components including generators, APUs and TRUs are uncontrollable and they can be either healthy (1) or unhealthy (0).
For generators and APUs, healthy means that they can produce an admissible voltage, while unhealthy can either represent outputting an improper voltage or no voltage at all, in whichever cases should be isolated from the rest of the circuit.

B. Directed Graph

The electric power system topology can be represented by a directed graph *G=(N, E)* that indicates the power flow direction. The set *N* of nodes consists of generators, APUs, buses, TRUs, and dummy nodes where a collection of wire meets. The set *E* of edges contains contactors and wires links between components.
The reason why we use a directed graph is because a rectifier unit can only convert alternating current (ac) to direct current (dc), but not the opposite. We separate a rectifier node into two, one for ac part, and the other for dc part. All edges are bidirectional, except for the edge from the ac part of a rectifier to its dc part.
A representative circuit is shown in Figure 1 and the corresponding graph is given in Figure 2.

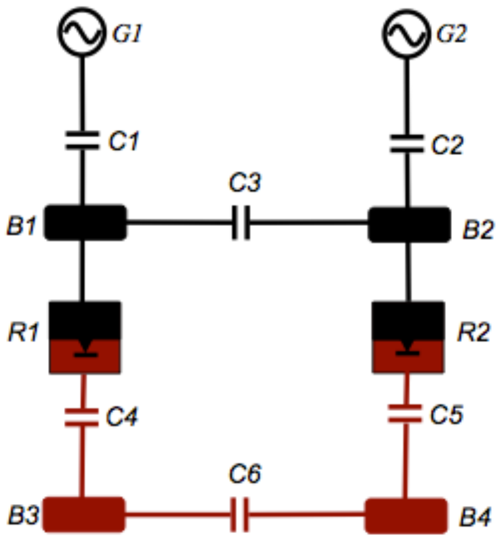

Figure 1. A single-line diagram of a simple electric power circuit

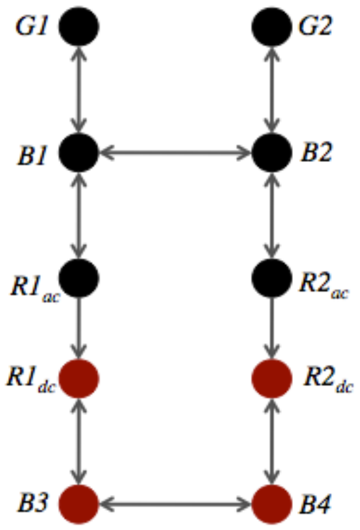

Figure 2. The directed graph representation of Figure 1

C. Rules

On a beginner's level, we come up with three general rules:

i. All essential buses should be powered on
ii. No-paralleling between ac sources
iii. Disconnect with unhealthy components

The first rule, or more like a goal we want to achieve, strictly states that all essential buses should be powered on at all time. For non-essential buses, however, they are also critical to the system; and although we do not require non-essential buses to be always powered on, there is an upper limit on their unpowered time. We only apply rules on essential buses, because SAT solvers cannot handle timing related requirements.
The second rule restrict that there should be no power flow from one ac source to another. This is because in modern aircraft variable frequency asynchronous generators are used and it is dangerous to connect two different asynchronous generators.
The last rule states that if any components (generator, APU or TRU) go unhealthy, all contactors around them should open immediately to avoid breaking down any other nearby connected components (generator, APU or TRU). These two rules ensure a safe operation of the electric power system for the electric power system.
As we dig deeper, we would want to apply more rules, or specifications on the circuit. It maybe commands from pilot, or rules unique to specific circuit. And these three general rules stated at the beginning are no longer sufficient for all outcomes we want.

D. Assumption-Guarantee Contract

A new concept called assumption-guarantee contract becomes handy, where assumptions capture possible internal and external events, for example like failures on circuit components or a command from the pilot; while guarantees capture safety requirement.

Guarantees as it refers to are basic rules that guarantee the electric power circuit to function properly under some most common configurations. Assumptions on the other hand are more like additional requirements and expectations to the system. They reflect the a priori knowledge about the uncontrollable events (like failures). We would want toe circuit to always operate correctly with assumptions we capture under some certain conditions. Some would restrict values on generators; others would want to have some certain contactors never close in some period. In this summer project, I only listed a few frequently used assumptions, for example, at least one generator is assumed to be healthy. These assumptions should not contradict to any of the guarantees.

III. PROCEDURES

What we want to achieve ultimately is to determine whether an electric power circuit under certain environment situation can be functioning properly, so as to say satisfied, by reconfiguration. If the electric power circuit under some assumption-guarantee contract can be satisfied, then the programming we implemented would present a set of solutions for all possible environment conditions of this circuit. The user can choose to ask for every plausible reconfiguration strategies under each environment conditions given, or our programming would provide only one reconfiguration strategy for each environment condition by default. The user can also add more specifications on the circuit not restricted to the ones already given.

To begin with, the user must provide an accurate netlist file corresponding to the electric power circuit. A netlist file is a file describing all components in a circuit following certain topology if preferable. The first step to generate a netlist file is to rank every wire junction usually beginning with generators and APUs. The figure shown in Figure 3 is an example picturing how a circuit is partitioned in order to create a netlist file.

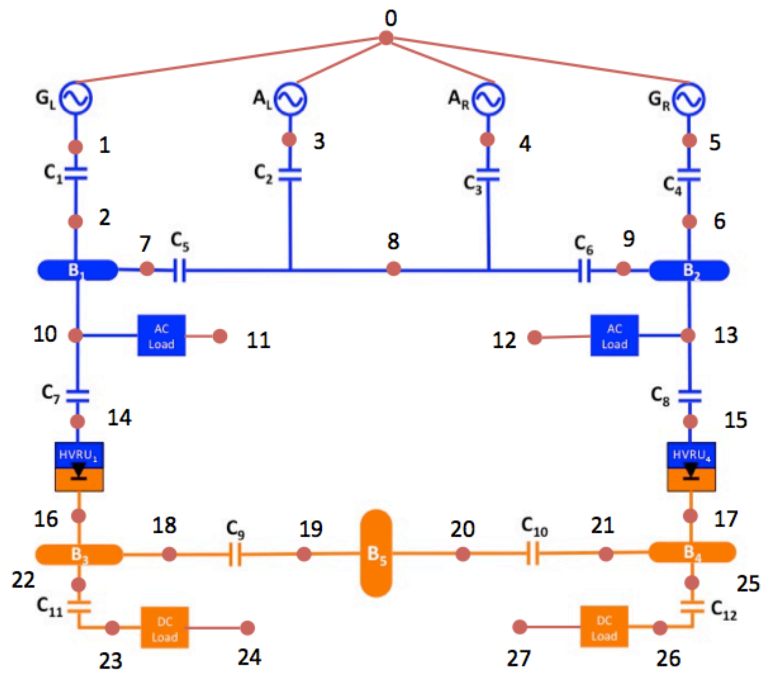

Figure 3. An electric power circuit described in SLD level of abstraction

In a netlist file for the aircraft electric power system, there are five types of components defined. Their initials are listed below:

- *C* stands for contactors
- *G* stands for generators
- *B* stands for buses
- *A* stands for APUs
- *T* stands for TRUs (rectifiers are considered the same)

The netlist file will then present a series of components followed by their port number indicating their positions in the electric power circuit. For generators and APUs, since they only have one port connecting to the rest of the circuit, we usually conclude a '0' port that connects to all generators and APUs.

The second step would be to read in this netlist file, and generate a corresponding directed graph. We include package 'networkx' in python (see [4]) to achieve this. Nodes in this directed graph as introduced earlier represent components except for contactors, while edges are composed of contactors or purely wire. The 'networkx' package has this function that can find if there exist paths between two nodes, which can be used to produce the logic rules guaranteeing no paralleling between ac sources.

We then generate assertions put into SAT solver from the circuit's directed graph. These assertions include both assumptions and guarantees, where the user can adjust assumptions. To understand these assertions, we first write down a few statements or so-called equations indicating a set of circuit connections among components. The figure presented in Figure 4 is a script for the circuit shown Figure 1. Symbols used such as $\wedge$,$\vee$,$\neg$, and $\rightarrow$ are notations in set theory representing intersection, union, negation and implication respectively.

$$B_1 = (G_1 \wedge C_1) \vee (G_2 \wedge C_2 \wedge C_3)$$
$$B_2 = (G_2 \wedge C_2) \vee (G_1 \wedge C_1 \wedge C_3)$$
$$B_3 = (B_1 \wedge R_1 \wedge C_4) \vee (B_2 \wedge R_2 \wedge C_5 \wedge C_6)$$
$$B_4 = (B_2 \wedge R_2 \wedge C_5) \vee (B_1 \wedge R_1 \wedge C_4 \wedge C_6)$$
$$L_1 = \neg(C_1 \wedge C_2 \wedge C_3)$$
$$L_2 = \neg(C_4 \wedge C_5 \wedge C_6)$$
$$S_1 = G_1 \wedge G_2 \wedge R_1 \wedge R_2$$
$$S_2 = G_1 \wedge G_2 \wedge R_1 \wedge (\neg R_2)$$
$$S_3 = G_1 \wedge G_2 \wedge (\neg R_1) \wedge R_2$$
$$S_4 = G_1 \wedge (\neg G_2) \wedge R_1 \wedge R_2$$
$$S_5 = (\neg G_1) \wedge G_2 \wedge R_1 \wedge R_2$$
$$S_6 = G_1 \wedge (\neg G_2) \wedge R_1 \wedge (\neg R_2)$$
$$S_7 = G_1 \wedge (\neg G_2) \wedge (\neg R_1) \wedge R_2$$
$$S_8 = (\neg G_1) \wedge G_2 \wedge R_1 \wedge (\neg R_2)$$
$$S_9 = (\neg G_1) \wedge G_2 \wedge (\neg R_1) \wedge R_2$$
$$F_1 = (\neg G_1 \rightarrow \neg C_1) = G_1 \vee (\neg C_1)$$
$$F_2 = (\neg G_2 \rightarrow \neg C_2) = G_2 \vee (\neg C_2)$$
$$F_3 = (\neg R_1 \rightarrow \neg C_4) = R_1 \vee (\neg C_4)$$
$$F_4 = (\neg R_2 \rightarrow \neg C_5) = R_2 \vee (\neg C_5)$$

Figure 4. Circuit statements script for Figure 1
(This is an early version and *R* here stands for rectifier)

B_1 through B_4 are necessary conditions to have these buses powered, corresponding to the first rule presented earlier. L_1 and L_2 are two constraints to avoid paralleling two generators. S_1 through S_9 are all possible states for the generators and rectifier units being either healthy or unhealthy, namely the environment assumptions. It is not necessary though to include these environment constraints. F_1 through F_4 indicate that if any of the generators or rectifier units goes unhealthy, we should turn off the contactor immediately next to it. This is the third rule that we would disconnect with the unhealthy components. Under any of the nine states S_1 through S_9 described above, we want to have all buses powered by adjusting some of the inputs, i.e. controllable contactors without breaking any constraints of the circuit. This could be represented in a general form as,

$$S_i \rightarrow ((L_1 \wedge L_2) \wedge (F_1 \wedge F_2 \wedge F_3 \wedge F_4) \wedge (B_1 \wedge B_2 \wedge B_3 \wedge B_4))$$

where $i = 1, 2, ..., 9$. We then use a SAT solver to find if this implication is true meaning the circuit is satisfiable under some environment conditions.

It would be much more neat and clear using an assumption-guarantee pair. Next, we explain how the SAT problem can be interpreted as an assumption guarantee pair.

Given a system S and an assumption-guarantee pair *(A, G),* we want to design a control protocol such that for all environment inputs in E_A (the set of environment inputs restricted by assumptions), the system *S* always satisfies its guarantees *G*. In other words,

$$\exists u \forall e \; in \; E_A \; s.t. \; G \; is \; satisfied$$

where *u* is the set of control inputs and *e* is an environment input.

In this project, we choose to use *cvc4* (see [5]) as our SAT solver for it is not only free but supports SMT-LIB. A *cvc4* input file includes a set of declarations over the module and all components, and also a set of assertions generated from the assumption-guarantee contract on the circuit. It checks on these assertions and gives a result indicating whether these assertions can be all satisfied. For *cvc4*, it only presents one solution if satisfiable, while it can point out which assertion makes the entire system unsatisfied. This makes it a bit troublesome to produce every possible solution. We apply a strategy by calling the *cvc4* and modifying the input file by restricting the result provided last time as fault, in order to force *cvc4* to produce different results each time until no more can be found. This strategy, however, would take more than exponential time and is therefore up to the user to choose if they want to receive all solutions or just one.

IV. RESULTS

In this project, we in total experimented virtually on three electric power circuit, from the simplest one with two generators and two rectifiers, to a comparatively complicated one from the literature [3]. In addition to the two circuits already shown in Figures 1 and 3, the complicated one is given in Figure 5.

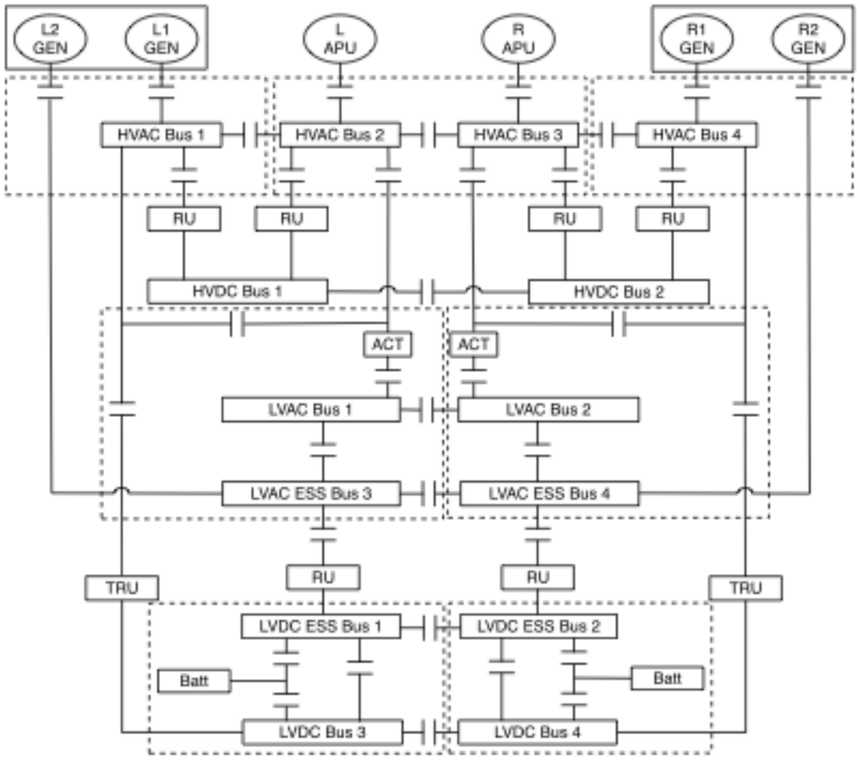

Figure 5. Single-line diagram of the most complicated circuit from literature [3] (transformers, i.e. ACT are neglected for now)

The processing time for the three circuits only applying the three rules listed before is shown in the table 1.

Table 1. Total processing time to give reconfiguration result

	Simplest circuit	Median-level circuit	Complicated circuit
One solution	0.564s	1.882s	>10 hours
All solutions	0.973s	17.805s	>2 days

‘One solution’ means the user choose to only produce one solution for each possible environment condition of the circuit, while ‘all solutions’ means to produce all reconfiguration combinations for each environment condition. For the complicated circuit, the exact time is inaccurate for it takes too long to give an result, and in this project we left it running on the computer for days but still could not get a complete result for the all-solutions case.
There are in total 8 nodes in the simplest circuit, and 11 nodes in the median-level one, while 28 nodes not including the dummy ones in the complicated circuit. The number of possible environment conditions is a power function proportionally related to the number of nodes.
the toolbox implementing ideas presented in this report is publicly available and can be accessed at https://github.com/EPS-Con/eps-reconfig

V. CONLUSION AND NEXT STAGE

We have succeeded in designing a centralized controller that can take in an environment condition and reconfigure the electric power circuit to make it function properly. What we would like to look into next step is a distributed controller design strategy.
We believe that distributed controller can be more efficiently synthesized even for large circuit. Moreover, a centralized controller can be huge for a large electric power circuit, which a mess of wire connection is imaginable. These are two of many reasons why a distributed controller is more desired and needed.
In this section we only introduce the idea of distributed controller in this section. For simple circuit, a direct and simple way to design distributed controller is to separate the ac part of the circuit from its dc part. This can be achieved from the directed graph as well. The dc buses in the circuit can then be considered as being powered on from ac buses, and this can save a large number of variables in their assertions, which means saving plenty of time solving SAT problem.
To make this separation on circuit more general, we use the idea of condensation. The condensation of a digraph *G* is based on the principle of strongly connected component which function is also implemented in the package ‘networkx’. After we obtain subsystems of the circuit, further questions remain unsolved whether the assumption-guarantee contract still holds, and if it does not, how should we modify it.

REFERENCE

[1]. Q. Maillet, H. Xu, N. Ozay and R. M. Murray, *Dynamic State Estimation in Distributed Aircraft Electric Control Systems via Adaptive Submodularity,* CDC’ 13, 2013
[2]. H. Xu, N. Ozay, R. M. Murray, *A Domain-Specific Language for Reactive Control Protocols for Aircraft Electric Power Systems,* HSCC’ 13, 2013
[3]. P. Nuzzo, H. Xu, N. Ozay, J. B. Finn, A. L. Sangiovanni-Vincentelli, R. M. Murray, A. Donzé, and S. A. Seshia1, *A Contract-Based Methodology for Aircraft Electric Power System Design*, volume 2, 2014
[4]. Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart, “Exploring network structure, dynamics, and function using NetworkX”, in Proceedings of the 7th Python in Science Conference (SciPy2008), Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 11–15, Aug 2008
[5]. Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. 2011. CVC4. In Proceedings of the 23rd international conference on Computer aided verification (CAV'11), Ganesh Gopalakrishnan and Shaz Qadeer (Eds.). Springer-Verlag, Berlin, Heidelberg, 171-177.