

Synthesis of Reactive Control Protocols for Switch Electrical Power Systems for Commercial Application with Safety Specifications

Benson Christalin¹, Michele Colledanchise², Petter Ögren², and Richard M. Murray¹

Abstract—This paper presents a method for the reactive synthesis of fault-tolerant optimal control protocols for a finite deterministic discrete event system subject to safety specifications. A Deterministic Finite State Machine (DFSM) and Behavior Tree (BT) were used to model the system. The synthesis procedure involves formulating the policy problem as a shortest path dynamic programming problem. The procedure evaluates all possible states when applied to the DFSM, or over all possible actions when applied to the BT. The resulting strategy minimizes the number of actions performed to meet operational objectives without violating safety conditions. The effectiveness of the procedure on DFSMs and BTs is demonstrated through three examples of switched electrical power systems for commercial application and analyzed using runtime complexity analysis. The results demonstrated that for large order system BTs provided a tractable model to synthesize an optimal control policy.

I. INTRODUCTION AND MOTIVATION

In an effort to facilitate the increasing complexity and automation of smart systems, there has been a growing demand for reliable and efficient Electrical Power Systems (EPS's). Computational complexity theory provides the tools to handle modeling, analyzing, and ensuring reliability of the intricate circuitry and operation of these safety critical EPS's. These tools enable the synthesis of control protocols that allow an EPS to quickly actuate to meet system objectives, while maintaining safe operating conditions. The synthesis of reactive control protocols involves modeling the system, safety, and mission specifications in a tractable form, then computing an optimal policy that accounts for the specifications. This paper details the construction of two independent models of the EPS, provides an algorithm to synthesize robust reactive controllers, evaluates the complexity of the algorithm in relation to the models, and demonstrates the realizability of the control protocols.

Synthesizing control protocols involves examining the discrete event space, and constructing a finite subset of the event space, subject to safety constraints. The finite subset of the event space is used to compute the sequential decisions necessary for power transfer under switching constraints. There has been work conducted that implements correct-by-construction control synthesis for power allocation and distribution in aircraft electric power systems [1] [2]. These papers

use linear temporal logic (LTL) specifications to synthesize a controller that is guaranteed, by construction, to satisfy formalized properties. The LTL specifications accounted only for safety and requires a GR(1) design. Using the collection of mathematical tools afforded by dynamic programming, this paper demonstrates the synthesis of a reactive controller for an electrical power system modeled by a deterministic finite state machine (DFSM) diagram and a behavior tree (BT). Furthermore, multiple electrical power system topologies are discussed in this paper. The first example allow readers to gain an intuition for the synthesis algorithm, the second example is motivated by an experimental test fixture used to validate automatically synthesized reactive control protocols [3], and the last is an industrial application from [4], seen in Figure 10, demonstrating feasibility and scalability of the synthesis procedure.

The choice to model the electrical power system as a discrete event system stems from the fact that the actuation of an electrical power system occurs at finite time intervals, the state of the electric power system is static unless triggered, and the information is represented in discrete form. A DFSM and BT was used to model the electrical power system. The DFSM for the electrical power system induces a directed graph known as a transition diagram. The nodes of the graph are the states of the system and the edges are transitions. With such an abstraction one can describe the necessary conditions to reach certain states using modal logic [5]. Furthermore, state transition diagrams are intrinsically sequential in nature, which aligns with the assumptions for actuation of the electrical power system. However, state transition diagrams can be infeasible. The conventional DFSM diagram formalism requires explicit representation of all states, and as the model grows linearly, the number of states grows exponentially.

Dimensionality reduction techniques have been studied in machine learning and statistic [6] to reduce the number of states. Common approaches are based on the *principal component analysis* [7] where it uses an orthogonal transformation to convert a collection of correlated variables into a collection of uncorrelated variables called *principal components*. However, the DFSM derived by an electrical power system has by nature uncorrelated variables. This is due to the fact that each state is a possible combination of closed/open switches (see Figure 5), hence a point in a orthogonal basis.

An alternative to state transition diagrams are Behavior Trees (BT). First introduced in the computer gaming industry [8] to meet their needs of compactness, modularity, and reusability in the artificial intelligence for non player

¹The authors are with the Department of Control and Dynamical Systems, California Institute of Technology, Pasadena, CA, USA bchrista@caltech.edu

²The authors are with the Computer Vision and Active Perception Lab., Centre for Autonomous Systems, School of Computer Science and Communication, Royal Institute of Technology (KTH), SE-100 44 Stockholm, Sweden {miccol|petter}@kth.se

characters [9], [10], BTs are a recent alternative to Controlled Hybrid Systems (CHSs) for fault tolerant execution of tasks. Different studies highlight the advantages of BTs over more classical CHSs in various applications [11]–[15], in particular robotic applications [16]–[18]. BTs are often used to describe fully reactive systems in a convenient and compact way [18]. A comparison of behavior trees and transition diagrams is provided, demonstrating the advantages of the compactness of the BTs in reducing the synthesis process’s computational complexity.

The remainder of the paper provides: an exposition of the mathematical setting, followed by a formal problem statement, a description of the EPS and problem description, a demonstration of the synthesis procedure through three examples used to compare the transition diagram and behavior tree model, and concluding remarks with suggestions for future work.

II. PRELIMINARIES

This section summarizes the mathematical background needed for the formulation of the control synthesis problem and solution.

A. $EPS \rightarrow Graph \rightarrow DFSM \rightarrow Transition Diagram$






The EPS is a network of components used in the generation and flow of power, and its inherent topological properties allows for a graphical representation.

Definition 1. Let $\mathcal{G} = (V, E)$ denote a *directed graph*. V is a nonempty set of *vertices* of the graph, denoted by $v \in V$. E is a subset of the Cartesian product $V \times V$ known as the *edges* of the graph, and denoted using set notation, $\{v, w\} \in E$ or $\{w, v\} \in E$, where v and w are nodes and the ordering indicates direction.

Definition 2. A *path* is a finite sequence of nodes $\langle v_0, v_1, \dots, v_k \rangle$. The set of paths of \mathcal{G} is $Paths(\mathcal{G})$.

Table I shows the circuit symbols and their corresponding graphical representation.

TABLE I. Symbols used and their graphical representations

Symbol	Description	Graphical Symbol
$ACGen \subseteq V$ $C \subseteq E$	AC Generators Switches	 
$DCLoad, ACLoad \subseteq V$	DC, AC Loads	 
$RU \subseteq V$	Rectifier Units	

A single path on the graph denotes a connection between circuit components and the flow of power. $Paths(\mathcal{G})$ represents all configurations/states of the EPS.

The states and transitions of the EPS can be represented with a deterministic finite state machine.

Definition 3. A *deterministic finite state machine* is a quadruple (X, Σ, f, Y) where:

- a finite set of states, X

- a finite set called the input alphabet, Σ
- a transition function mapping pairs of a state and an input symbol to the corresponding next state, $f : X \times \Sigma \rightarrow X$
- a finite set of final states, $\mathcal{Y} \subset X$

Remark 1. The DFSM induces a graph known as a *transition diagram*.

Definition 4. The information provided by the transition function mapping, f , can be represented as a *transition matrix* denoted by F , and is equivalent to an adjacency matrix for the transition diagram. The entries of the transition matrix are assigned values from zero to infinity based on safety requirements and configuration preferences. These values are also the weights on the edges of the transition diagram.

B. Behavior Trees

This subsection presents a digest of behavior trees (BT), describing the execution of the nodes used throughout the paper. Refer to [10] for a more detailed description.

Definition 5. A *BT* is a directed tree where the internal nodes are classified as *control flow nodes* and the leaf nodes as *execution nodes*, using the usual definition of parent and children for each connected nodes. Graphically, the children of a control flow node are sorted from its bottom left to its bottom right, as depicted in Figures 1-2. The execution of a BT starts from the root node (i.e. the control flow nodes with no parents), which sends *ticks* to its children. When a node in a BT receives a tick, its execution starts and it returns to its parent a status of *running* if its execution is under completion; *success* if its execution is accomplished; or *failure* if the execution cannot be accomplished. Here we describe the execution of the two control flow nodes (selector, sequence) and the execution nodes (action and condition).

Definition 6. Fallback node (also known as Selector or Priority) When a fallback node receives a tick, then it ticks its children in succession from left to right, until a child returns the status of success or running. Then this status is returned to the parent of the fallback node. A fallback node returns failure only when all the children return a status failure. The purpose of the fallback node is to carry out a task that can be performed using different approaches (e.g. powering a bus can be either done by switching the generator on or by plugging the external battery). A fallback node is graphically represented as a box with a “?”, as in Fig. 1.

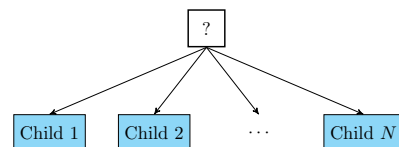


Fig. 1. Graphical representation of a fallback node with N children.

Definition 7. When a sequence node receives a tick, then it ticks its children in succession from left to right, until a child returns the status of failure or running. Then this status is returned to the parent of the sequence node. A sequence node returns success only when all the children return a status success. The purpose of the sequence node is to carry out a task that is defined as a strict sequence of sub-tasks (e.g. powering a load need to have the load connected and the generator on) A fallback node is graphically represented as a box with a “→”, as in Fig. 2..

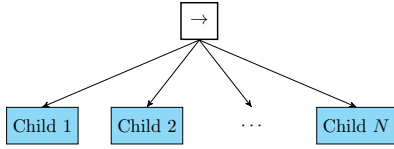


Fig. 2. Graphical representation of a sequence node with N children.

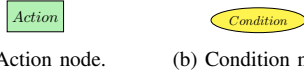


Fig. 3. Graphical representation of action and condition nodes.

Definition 8. When an action node receives a tick, it returns the status of success if the action is completed or failure if the action cannot be completed. While it is performing the action, it returns the status of running. An action node is graphically represented as in Fig. 3(a)

Definition 9. When a condition node receives a tick, it returns the status of success if the condition is satisfied or failure otherwise. A condition node never returns running. A condition node is graphically represented as in Fig. 3(b).

C. Dynamic Programming: Shortest Path Problem

Definition 10. A *control policy* for DFSM is a sequence of control vectors

$$\{u_0, u_1, \dots, u_{N-1}\} \in \Sigma$$

where the subscript 0 and $N - 1$ denotes the initial control input and final control input, respectively.

Therefore a DFSM system can be denoted by

$$x_{k+1} = f(x_k, u_k)$$

where $x_{k+1}, x_k \in X$, and f_k is the transition function for $k = 0, 1, \dots, N - 1$.

Interested in finding an optimal control policy, let us denote the cost of transition from a state x_i to x_j by $c_{i,j}^k$.

Remark 2. The cost associated with transitions to states that are not reachable either by design or due to failure are $c_{i,j}^k = \infty$. All other cost have a value between 1 and ∞ based on connectivity preferences and specifications, with the terminal cost denoted by $c_{i,j}^N$.

The DFSM can be transcribed into a graph such that the nodes correspond to the states of the system, the edges correspond to the transitions, and the weight of the edges

correspond to the cost of transition. This construction lends itself to view the DFSM problem as a shortest path problem from an initial node to a terminal node on the graph which can be posed as a DFSM dynamic programming (DP) problem.

III. PROBLEM STATEMENT

The computations on the graphs should be equivalent to determining subsets of the discrete event space that represent the events to be controlled. The objective is to minimize the expected number of switches to get to safe-on final state while always remaining in $X_{safe-on} \cup X_{safe-off}$ where $\mathcal{Y} \subseteq X_{safe-on}$.

Remark 3. $X_{safe-on}$ is the set of states that meet mission specifications and do not violate safety specification, while $X_{safe-off}$ is the set of states that only satisfy safety specifications.

For the simple case, the transition cost is 1 if the transition is to safe space of the events space, 0 if the transition is in \mathcal{Y} or ∞ otherwise. Therefore the minimum cost from an initial node v_i to a terminal node v_t is

$$J_k(i) = \min_{t \neq i} \{J_{k+1}(t) + c_{i,t}^k\} \text{ for } k = 0, 1, \dots, N - 1.$$

Provided the appropriate labeling of the DFSM, the problem likens to determining the sequences with the minimum number of transitions needed to traverse from an initial state to a final state, with the constraint imposed the safe-on set, or the minimum number of actions needed to reach the same state under equivalent constraints for a BT.

IV. OPTIMAL REACTIVE CONTROL POLICY SYNTHESIS

This section provides the methodology to find a robust controller which can be summarized in four steps:

- Using specifications construct a DFSM or BT to obtain the event space set and necessary subsets;
- Using a DP algorithm search the DFSM or BT for a static policy;
- Verify that the final state satisfies the mission objective and system specifications;
- If the mission objective and system specifications are not met re-label the transition diagram and BT; accordingly and re-run the DP algorithm in search of a new policy.

A. Shortest Path on Transition Diagram

Using recursive fixing, a backward search is performed to converge on a policy that account for the entire event space and therefore the convergent rate is dependent on the cardinality of the event space. The procedure starts at the terminal vertex, v_t , and iterates of the transition diagram until the iteration that spans all vertices reachable from v_t . Vertices are evaluated in order of cost, first v_t , then the vertex with the least cost connect to v_t , and so on. $J(x)$ defines the cost from the initial vertex and the algorithm describe below considers only the evaluated vertices and that minimize the cost.

Algorithm 1: shortest path DP Algorithm adapted from [19]

Input: cost function, states, & transitions

Output: shortest path

```

1 Initialization: set  $J_N(i) = c_{i,t}^N, k = 0$ 
2 repeat
3   foreach  $k$  do
4     foreach state  $t \neq i$  do
5       compute vector
            $J_k(i) = \min_{t|t \neq i} \{J_{k+1}(t) + c_{i,t}^k\}$ 
6 until  $J_{k+1} \approx J_k$ ;
7 return  $J_k^*$ 

```

Under assumptions outlined, the stationary policy is a set of transitions that satisfy

$$J_k(i) = \min_{t|t \neq i} \{J_{k+1}(t) + c_{i,t}^k\} \text{ for } k = 0, 1, \dots, N-1$$

where i denotes the node $v_i \in X$ and t denotes the node $v_t \in \mathcal{Y}$

B. BT Search

Algorithm 2 aims to create a BT that returns success if and only if the current switch configuration satisfies the mission requirement without violating the safety specifications. For each switch configuration $Y \in \mathcal{Y}$ we create a BT \mathcal{T}_Y that returns success if and only if the such configuration is met. Initially this \mathcal{T}_Y is composed by conditions only (Algorithm 3 Line 9). We execute \mathcal{T}_x on the graph starting with the initial configuration x_0 . If the the switch configuration Y is not met, \mathcal{T}_Y returns failure. We identify which single element $y_i \in Y$ is not met (Algorithm 3 Line 12). For each x_i we identify a BT that will meet such single switch configuration without violating any safety requirements (Algorithm 3 Line 15). We repeat the procedure until \mathcal{T}_x return success. Finally we order \mathcal{T} to achieve optimality. In the ordered BT, each fallback node has its children ordered by the the number of actions in an ascending fashion.

C. Label Correction

Label correction allows for the synthesis procedure to be reactive. Label correction occurs when a policy does not satisfy a mission objective or safety specification, such as a failure in the system that results in an unreachable state. This information is provided by sensors. If a mission is not satisfied, the final state is re-labeled as a unsafe state, and the DP search is again performed with new values in the transition diagram and pre and post conditions for the BT in accordance to the new label.

Algorithm 2: get_safe_subtree

input : Single switch configuration: x

Mission Configuration: Y

Violating Configurations: \mathcal{S}

output: Subtree to safely satisfy y

```

1 for  $S \in \mathcal{S}$  do
2    $\mathcal{T}_S \leftarrow \text{fallback\_node}()$ 
3   if  $x \in S$  then
4      $\tilde{S} \leftarrow S \setminus Y$ 
5     for  $\tilde{s} \in \tilde{S}$  do
6       if  $\tilde{s} > 0$  then
7          $\tilde{c} \leftarrow \text{condition\_node}(\text{Switch } \tilde{s}$ 
           Is OPEN)
8       else
9          $\tilde{c} \leftarrow \text{condition\_node}(\text{Switch } \tilde{s}$ 
           Is CLOSED)
10       $\mathcal{T}_S.\text{add\_child}(\tilde{c})$ 
11   if  $x > 0$  then
12      $a \leftarrow \text{action\_node}(\text{Switch } x \text{ CLOSED})$ 
13      $c \leftarrow \text{condition\_node}(\text{Switch } x \text{ Is}$ 
           CLOSED)
14   else
15      $a \leftarrow \text{action\_node}(\text{Switch } x \text{ OPEN})$ 
16      $c \leftarrow \text{condition\_node}(\text{Switch } x \text{ Is}$ 
           OPEN)
17    $\mathcal{T}_a \leftarrow \text{sequence\_node}()$ 
18    $\mathcal{T}_a.\text{add\_child}(\mathcal{T}_S)$ 
19    $\mathcal{T}_a.\text{add\_child}(a)$ 
20    $\mathcal{T} \leftarrow \text{fallback\_node}()$ 
21    $\mathcal{T}.\text{add\_child}(c)$ 
22    $\mathcal{T}.\text{add\_child}(\mathcal{T}_a)$ 
23 return  $\mathcal{T}$ 

```

V. ALGORITHM ANALYSIS

A. Worst-Case Complexity Analysis Using DFSM

The abstraction used to perform calculations on the DFSM was extended from the EPS graph. The complexity of the construction of the EPS graph is $O(V+E)$. The nodes of the transition diagram are 2^V and the edges are $V \times 2^V$ therefore the complexity to construct the graph is $O(2^V \times (1+V))$. The worst-case work bound of the policy synthesis procedure is $O(|V \times 2^V| |2|^V)$.

B. Worst-Case Complexity Analysis Using BT

Given the mission set \mathcal{Y} and the safety violations set \mathcal{S} Algorithm 3 computes a number $|\mathcal{Y}|$ of BTs \mathcal{T}_Y for each switch configuration $Y \in \mathcal{Y}$. In each \mathcal{T}_Y , Algorithm 3 finds a subtree \mathcal{T}_y to safely satisfy each single switch configuration $y \in Y$. Each \mathcal{T}_y is computed in $O(|\mathcal{S}||E|)$ time where $|E|$ is the cardinality of the edges of the graph (number of switches). For each \mathcal{T}_Y we compute at most a number $|E|$ of \mathcal{T}_y (worst case where we have to perform an action on each switch). Finally Algorithm 3 sorts the children of \mathcal{T} . The sorting is

Algorithm 3: get_optimal_safe_subtree

input : Initial configuration: x_0
Mission Configurations: \mathcal{Y}
Violating Configurations: \mathcal{S}

output: BT \mathcal{T}

```
1  $\mathcal{T} \leftarrow \text{fallback\_node}()$ 
2 for  $Y \in \mathcal{Y}$  do
3    $\mathcal{T}_Y \leftarrow \text{sequence\_node}()$ 
4   for  $Y \in \mathcal{Y}$  do
5     if  $m > 0$  then
6        $c \leftarrow \text{condition\_node}(\text{Is Contactor}$ 
7          $y \text{ CLOSED})$ 
8     else
9        $c \leftarrow \text{condition\_node}(\text{Is Contactor}$ 
10         $y \text{ OPEN})$ 
11      $\mathcal{T}_Y.\text{add\_child}(c)$ 
12    $\text{status} \leftarrow \mathcal{T}_Y.\text{execute\_on\_graph}(x_0)$ 
13   while  $\text{status} \neq \text{success}$  do
14      $C \leftarrow \mathcal{T}_Y.\text{failed\_conditions}()$ 
15     for  $c \in C$  do
16        $\tilde{c} \leftarrow \text{get\_value\_of}(c)$ 
17        $\mathcal{T}_c \leftarrow \text{get\_safe\_subtree}(\tilde{c}, Y, \mathcal{S})$ 
18        $\mathcal{T}_Y.\text{replace\_child}(c, \mathcal{T}_c)$ 
19      $\text{status} \leftarrow \mathcal{T}_Y.\text{execute\_on\_graph}(x_0)$ 
20    $\mathcal{T}.\text{add\_child}(\mathcal{T}_Y)$ 
21  $\mathcal{T}.\text{reorder}()$ 
22 return  $\mathcal{T}$ 
```

done in $O(|\mathcal{Y}||E|\log(|\mathcal{Y}| + |E|))$ time, as for each \mathcal{T}_M we sort each subtree \mathcal{T}_m . times. Hence the proposed approach computes a BT in $O(|\mathcal{Y}||\mathcal{S}||E|^2 + |\mathcal{Y}||\mathcal{S}||E|\log(|\mathcal{Y}| + |E|))$ time.

Proposition 1. Algorithm 2 finds a BT in finite time.

Proof. The sets \mathcal{S} and $\tilde{\mathcal{S}}$ are finite. Hence the loops inside Algorithm 2 executes a finite number of operations \square

Proposition 2. Algorithm 3 finds an optimal BT in finite time.

Proof. The set \mathcal{Y} is finite set. Hence the number of \mathcal{T}_Y computed is finite. The \mathcal{T}_Y is updated until it return success. To prove that it will return success in finite time we need to prove that an action required to satisfy the safety specification do not conflict with an action required to perform the mission. Algorithm 3 computes the actions required to satisfy the safety specification using Algorithm 2. Algorithm 2 consider only those switches that are not listed in the mission specification (Line 4) Hence no conflicting actions are performed. The optimality is ensured by the sorting algorithm. \square

VI. APPLICATION FRAMEWORK

A. Electrical Power System

An electrical power system is a modular collection of components and circuits necessary for the generation, transmission and distribution of power. The typical components of an EPS are generators/alternators, rectifier units, transformers, switches, batteries, and loads. Generators provide the source of power that the system will convert into electrical energy. Rectifier units convert alternating current (AC) to direct current (DC). Transformers use electromagnetic induction through a "step-up" or "step-down" process that involves the increase or decrease of voltages to transfer electrical energy. Batteries store electrical energy. Switches connect and disconnect circuit components to maintain system requirements while allowing the EPS to provide energy to the loads. And loads consume the electrical energy to perform a function. These components can be arranged to form various topologies in order to regulate and control the conversion and transmission of electrical energy for consumption. Maintaining power flow is the key objective of the electrical power system for safety critical applications. For this paper, it is assumed that the EPS operates according to a reflected binary code (RBC), also known as Gray code, switching scheme, where two successive states values differ in only one bit (binary digit) translating to one switch being turned on or off.

B. Problem Description

Given an EPS topology, an initial state, and mission objective, synthesize a reactive switching control robust to faults that yields the optimal sequence of actions and final state of the system. An optimal sequence, in reference to the control policy, is defined as the minimal number of actions required to reach a final state that does not violate safety specification.

VII. EXAMPLES

A switch configuration is denoted by a vector where if the i^{th} entry is 1 the i^{th} -switch is close and if 0 the i^{th} -switch is open. Actuation of the system corresponds to a transition and the observed switch configuration corresponds to a state of the system. Use the states and the transitions to create a DFSM and DFSM transition diagram. Now, derive the control policy. For the DFSM-based policy, a search is performed on the DFSM transition diagram to identify a path from the initial configuration to the a goal configuration. Transitions are assigned cost values to ensure specifications are satisfied. In the BT-based policy a recursive search is performed to find the sequence of actions needed to satisfy the mission and safety requirement. Regardless of the EPS model, a backward search approach from the goal configuration identifies the actions that yield the desired results. Before performing an action, the BT ensures that it does not violate any safety requirement (e.g. it turns off some other switches to avoid safety violation). Among all the possible BTs or paths on the DFSM transition diagram,

the one that represents the least number of actions from the initial configuration to the terminal configuration is chosen.

A. Motivating Simple Three-Switch Example

Consider the EPS in figure 4(a). The graph representation of the EPS is depicted in figure 4(b). The mission is to power the load without short-circuits. The initial condition of the system is all switches opened, $x_0 = [0, 0, 0]$. The indexed set of terminal configurations that satisfy the mission is denoted by \mathcal{Y} ; thus, for this example, $\mathcal{Y} = \{[1, 0, 1] \cup [0, 1, 1]\}$. The states that correspond to a safety violation are denoted by \mathcal{S} and therefore, for this example, $\mathcal{S} = \{[1, 1, 1] \cup [1, 1, 0]\}$.

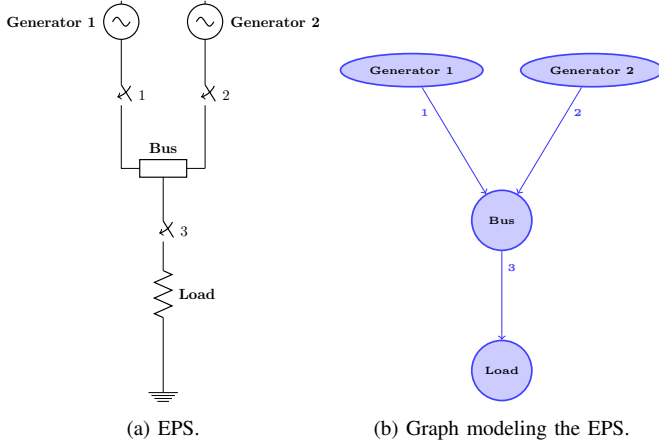


Fig. 4. EPS and Graph for Example VII-A. The EPS has four components: two generators, one bus, and one load, and three switches: switch 1, 2, and 3.

Notice that all the possible states and transitions of the electrical power system DFSM are depicted in the DFSM diagram shown in figure 5. In the figure, the dotted lines denote the transitions that yield a safety violation, the boxed nodes are viable terminal configurations, and the arrow indicates the initial state. A weighted value of infinity is assigned to edges related to transitions to unsafe states. Weighted values are also assigned to edges indicating preferential states. These values are stored in a cost matrix and used to when computing value and policy iteration, which provide the optimal set of actions from every state to every state. The distant from one state to another is the sum of the weighted edges that make up the path connecting the two nodes. For this example, assume that all non-dotted edges in figure 5 have a value of one; one can immediately identify potential paths through safe states, and the related actions, that result in a satisfactory terminal state such as: close switch 2 and then close switch 3 or close switch 1 and then close switch 3. Since both paths are equidistant from the initial position and no preference was indicated, a probabilistic approach or which ever sequence was queued first.

For the BT, figure 6 shows the construction of the tree \mathcal{T}_{Y_1} step by step. Algorithm 3 creates the initial \mathcal{T}_{Y_1} depicted in figure 6(a). \mathcal{T}_{Y_1} returns failure since the condition *is 1 CLOSED* is not satisfied. Then, Algorithm 3 expands the tree as seen in figure 6(b). Now, \mathcal{T}_{Y_1} returns failure since the condition *is 3 CLOSED* is not satisfied. Again, Algorithm 3 expands the tree as shown in figure 6(c). \mathcal{T}_{Y_1} finally returns

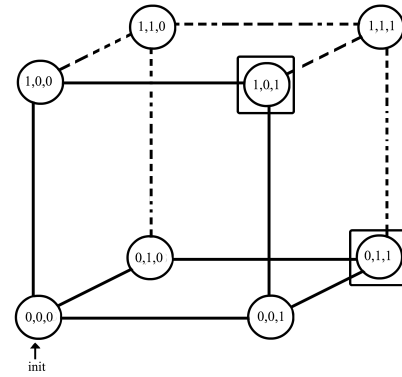


Fig. 5. FSM Diagram of Example VII-A

success. \mathcal{T}_{Y_2} is constructed in a similar way and depicted in figure 6(d). Given the initial state, both \mathcal{T}_{Y_1} and \mathcal{T}_{Y_2} results in two operations performed hence their order in \mathcal{T} is irrelevant. If there is a preference indicated by a liveness specification, simply order the tree accordingly. \mathcal{T} is depicted in figure 7.

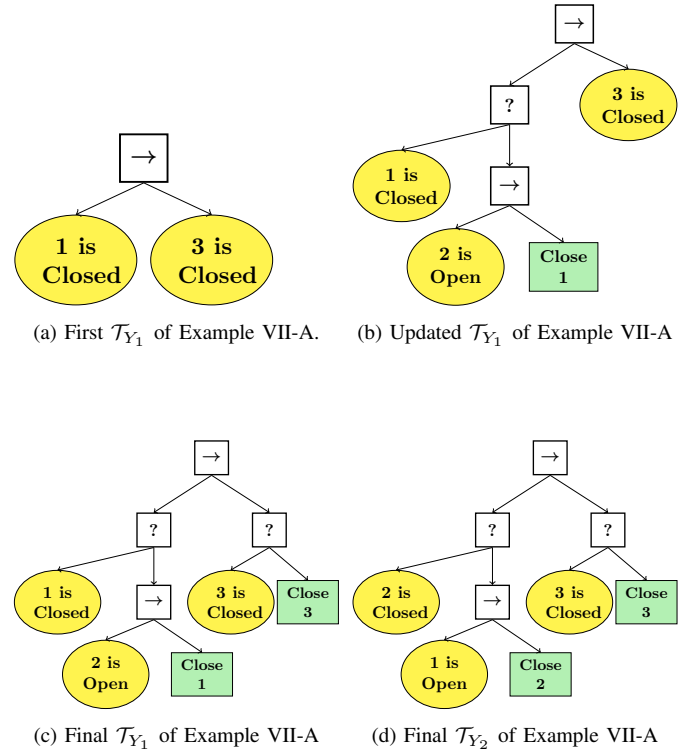


Fig. 6. Construction of \mathcal{T}_{Y_1} and \mathcal{T}_{Y_2} for Example VII-A.

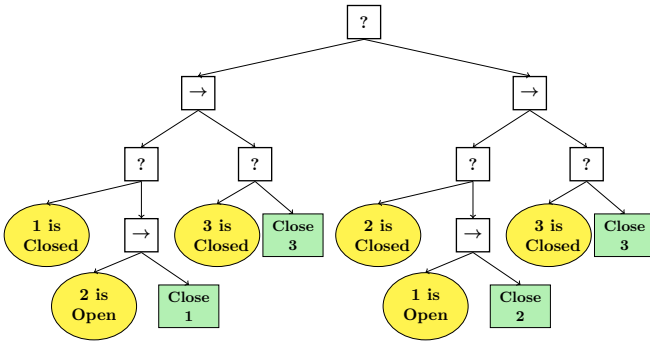


Fig. 7. \mathcal{T} of Example VII-A

B. Experimental Test Fixture

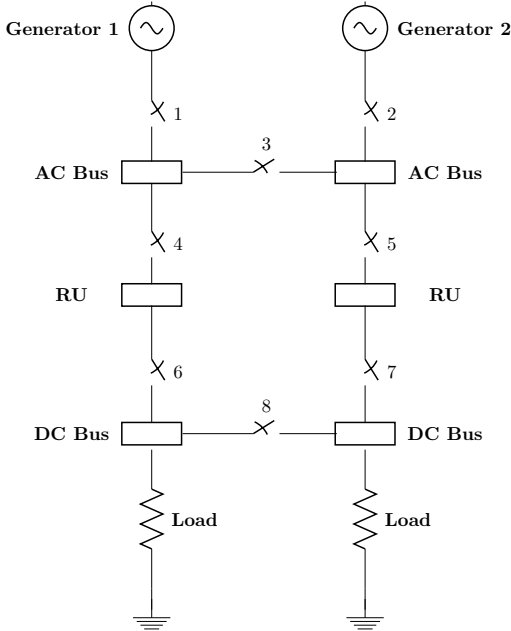


Fig. 8. EPS of Example VII-B

Consider the EPS in Fig. 8 where the mission requirement is to power the four loads and the safety specification is to not parallel the two generator. The mission set is $\mathcal{Y} = \{y_1 \cup y_2 \cup y_3 \cup y_4 \cup y_5\}$ where:

$$y_1 = [1, 1, 0, 1, 1, 1, 1, 0]$$

$$y_2 = [1, 0, 1, 0, 1, 0, 1, 1]$$

$$y_3 = [0, 1, 1, 1, 0, 1, 0, 1]$$

$$y_4 = [1, 1, 0, 1, 0, 1, 0, 1]$$

$$y_5 = [1, 1, 0, 0, 1, 0, 1, 1]$$

The safety violation set is $\mathcal{S} = \{s_1 \cup s_2 \cup s_3\}$ where:

$$s_1 = [1, 1, 1, 0, 0, 0, 0, 0]$$

$$s_2 = [1, 1, 0, 1, 1, 1, 1, 1]$$

$$s_3 = [1, 1, 1, 1, 1, 1, 1, 1]$$

The initial switch configuration is

$$x_0 = [1, 1, 0, 1, 1, 1, 1, 0]$$

that is all the switches except 3 and 8 are closed. We chose such initial configuration to demonstrate how the framework satisfies the safety conditions. The resulting BT \mathcal{T} is shown in Fig. 9. Each child in \mathcal{T} achieves a switch configuration in the mission set \mathcal{Y} . The children are ordered by the number of actions they execute. Some children have to open switches to satisfy the safety requirement. In the nominal case the actions executed are: *Switch 1 CLOSED*, *Switch 5 CLOSED*, *Switch 7 CLOSED*. None of this action violate a safety constraint. Note that if, due to several faults, the \mathcal{T} executes its third child (i.e. it performs the third best plan), the switch 3 has to be opened before closing switch 2 to avoid the two generators being paralleled. The DFSM transition diagram yields the same result, however it required additional computing power provided by Amazon Web Services and the diagram is too large to display. It contains 256 nodes and 2048 edges. Nodes pertaining to safety violation shown below as switch configuration were removed in order to reduce computational complexity.

C. Large-Scale Electrical Power System

In this example we consider an electrical power system for commercial applications. The electrical power system is depicted in figure 10. For this example the objective was to power the *LVDC Bus 3*. The safety specification is to not parallel any two generators. The initial switch configuration is such that only the switches 1, 6 and 26 are on. We chose to not enumerate the sets \mathcal{Y} and \mathcal{S} due to magnitude of the cardinality of the event space to search, $|X| = 10^{13}$.

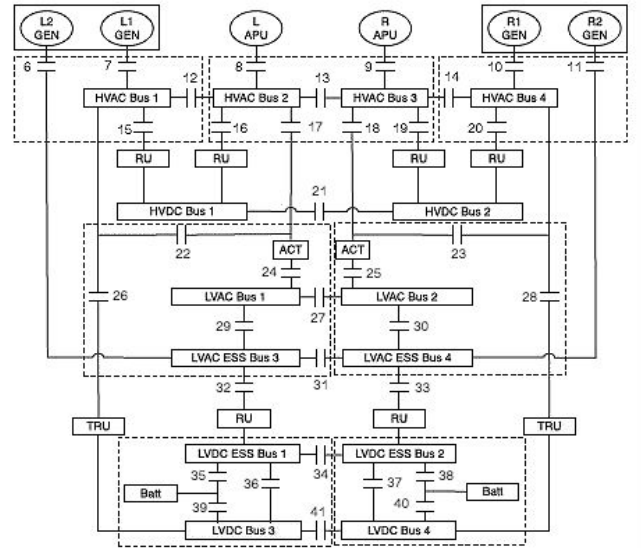


Fig. 10. Electrical Power System Schematic for Example VII-C

The resulting \mathcal{T} has $|\mathcal{Y}|$ children. We chose to depict only the most left one (nominal path) in Fig. 11. Intuitively to

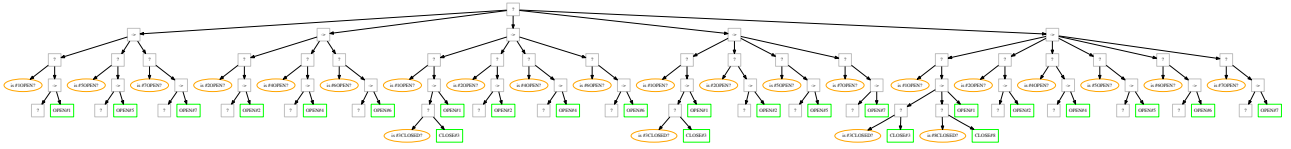


Fig. 9. \mathcal{T} of Example VII-B. The precondition that are satisfied on the initial state are not shown for space limitation (fallback nodes without children).

power the LVDC Bus 3, the switches 1, 27, and 31 must be turned on. Since at the initial condition the switches 11, and 31 are on, turning 6 on will yield an unsafe switch configuration (Generator L2 and Generator R2 in parallel). To avoid such a configuration, either the switch 6 or 26 have to be switched open before closing switch 6. The DFSM formulation was found intractable for this example.

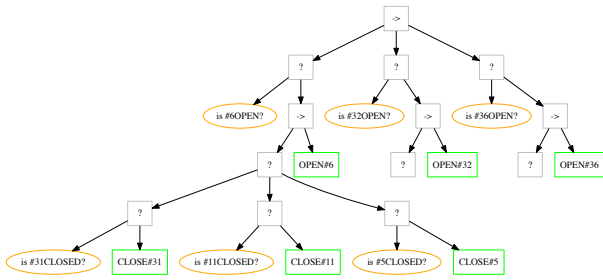


Fig. 11. First child of BT of Example VII-C.

VIII. CONCLUSION AND FUTURE DIRECTION

In this paper we presented a methodology to synthesize fault tolerant control protocol for an aircraft EPS modeled as discrete event system. The electrical power systems were modeled as DFSMs and BTs. The algorithm used dynamic programming techniques on graphical abstractions of the systems to synthesize a switching protocol. The results demonstrated, for small scale systems, that the algorithm was effective with a DFSM diagram model. However, unless additional modeling techniques were implemented to compress the representation of the system’s event space, the problem became intractable. BTs provide a tractable formulation for the large-scale electrical power systems. The proposed approach computed a minimal switching protocol for an EPS whose switching is defined by Gray code convention. Future work will seek to improve the algorithm and formulation presented in this paper and use the results to address other problems related to heuristic decision-making for electrical powers systems such as sensor placement, fault detection and isolation, and synthesis of built-in-test.

ACKNOWLEDGMENTS

The authors would like to thank Scott Livingston, Ivan Pashusha, and the anonymous reviewers for helpful comments. This work was supported in part by IBM and UTC via the iCyPhy consortium

REFERENCES

- [1] N. Ozay, U. Topcu, R. M. Murray, and T. Wongpiromsarn, “Distributed synthesis of control protocols for smart camera networks,” in *Proceedings of the 2011 IEEE/ACM Second International Conference on Cyber-Physical Systems*. IEEE Computer Society, 2011, pp. 45–54.
- [2] H. Xu, U. Topcu, and R. M. Murray, “A case study on reactive protocols for aircraft electric power distribution,” in *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*. IEEE, 2012, pp. 1124–1129.
- [3] R. Rogersten, H. Xu, N. Ozay, U. Topcu, and R. M. Murray, “An aircraft electric power testbed for validating automatically synthesized reactive control protocols,” in *Proceedings of the 16th international conference on Hybrid systems: computation and control*. ACM, 2013, pp. 89–94.
- [4] R. G. Michalko, “Electrical starting, generation, conversion and distribution system architecture for a more electric vehicle,” Oct. 21 2008, uS Patent 7,439,634.
- [5] J. Hopcroft, *Introduction to automata theory, languages, and computation*. Boston: Pearson/Addison Wesley, 2007.
- [6] K. Pearson, “On lines and planes of closest fit to systems of points in space,” *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.
- [7] P. Cunningham, “Dimension reduction,” 2007.
- [8] D. Isla, “Handling Complexity in the Halo 2 AI,” in *Game Developers Conference*, 2005.
- [9] I. Millington and J. Funge, *Artificial Intelligence for Games*. Taylor & Francis, 2009. [Online]. Available: <https://books.google.com/books?id=1OJ8EhvuPXAC>
- [10] S. Rabin, *Game AI Pro: Collected Wisdom of Game AI Professionals*. Natick, MA, USA: A. K. Peters, Ltd., 2013.
- [11] R. d. P. Pereira and P. M. Engel, “A framework for constrained and adaptive behavior-based agents,” *arXiv preprint arXiv:1506.02312*, 2015.
- [12] K. R. Guerin, C. Lea, C. Paxton, and G. D. Hager, “A framework for end-user instruction of a robot assistant for manufacturing,” in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, May 2015, pp. 6167–6174.
- [13] M. Colledanchise, R. Parasuraman, and P. Ögren, “Learning of behavior trees for autonomous agents,” *arXiv preprint arXiv:1504.05811*, 2015.
- [14] A. Klöckner, “Behavior trees for uav mission management,” in *INFORMATIK 2013: Informatik angepasst an Mensch, Organisation und Umwelt*. Köllen Druck + Verlag GmbH, Bonn, 2013, pp. 57–68.
- [15] D. Hu, Y. Gong, B. Hannaford, and E. J. Seibel, “Semi-autonomous simulated brain tumor ablation with ravenii surgical robot using behavior tree,” in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, May 2015, pp. 3868–3875.
- [16] A. Klöckner, “Interfacing Behavior Trees with the World Using Description Logic,” in *AIAA conference on Guidance, Navigation and Control, Boston*, 2013.
- [17] M. Colledanchise and P. Ögren, “How Behavior Trees Modularize Robustness and Safety in Hybrid Systems,” in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, Sept 2014, pp. 1482–1488.
- [18] P. Ögren, “Increasing Modularity of UAV Control Systems using Computer Game Behavior Trees,” in *AIAA Guidance, Navigation and Control Conference, Minneapolis, MN*, 2012.
- [19] D. P. Bertsekas, D. P. Bertsekas, D. P. Bertsekas, and D. P. Bertsekas, *Dynamic programming and optimal control*. Athena Scientific Belmont, MA, 1995, vol. 1, no. 2.