Fuel Cell Thermal Management: Modeling, Specifications and Correct-by-Construction Control Synthesis

Liren Yang, Amey Karnik, Benjamin Pence, Md Tawhid Bin Waez, Necmiye Ozay

***Abstract*— The safe and efficient operation of fuel cells requires thermal management. The goal of this paper is to algorithmically synthesize a provably correct controller for a fuel cell thermal management system. We begin by developing a control-oriented model for the thermal management system of a fuel cell stack. Then, we list the requirements associated with thermal management and formalize them using linear temporal logic. The model and the requirements are then used for controller synthesis with an abstraction-based technique. To make the abstraction-based synthesis algorithm computationally efficient, some structural properties of the fuel cell system dynamics are identified and leveraged. Finally, the closed-loop system behavior with the synthesized controller is demonstrated via simulations.**

I. Introduction

Fuel cells are electrochemical devices that convert chemical energy of gaseous fuel (i.e., hydrogen) into electricity [15]. In a fuel cell stack, the electrochemical reaction of oxygen and hydrogen generates electrical power, while heat and water are produced as by-products. In this work, we focus on developing the thermal management portion of the controller, which guarantees that the fuel cell operates in a proper temperature range (340K to 350K), for safety and efficiency considerations [7].

A simplified schematic of the fuel cell thermal management system is shown in Fig. 1. The two main factors that affect the heat supplied or removed from the fuel cell stack, and hence the stack temperature, are the stack coolant inlet temperature and the coolant flow-rate. The coolant flow rate is controlled by an electric pump, while the coolant inlet temperature is regulated by appropriately flowing the coolant through a radiator or a heater, where the flow path is selected by a 2-position 3-way valve, thus making the system dynamics hybrid in nature.

The electrical power requirements have a direct influence on thermal management, some aspects of which are studied in [5], [9]. First, the power requirements by the motor and the fuel cell temperature constraint need to be met simultaneously through system hybridization, that is, through appropriate power management between the fuel cell stack and an energy storage device. Since the heat generated in the stack increases with increasing fuel cell output power, the stack temperature may exceed the desired range while the fuel cell tries to match its output power with the instantaneous power request from the motor. Hence, a battery is introduced to "moderate" the power generated by the fuel cell. In addition, power requirements from the heater when used for warm-up under cold conditions also affect power management. In this paper, we provide some of the key requirements for fuel cell thermal management in the presence of battery state of charge energy constraints. These requirements are evaluated in the situation where the ambient temperature is near 283K, where the fuel cell stack loses significant heat to the ambient due to the large temperature gradient.

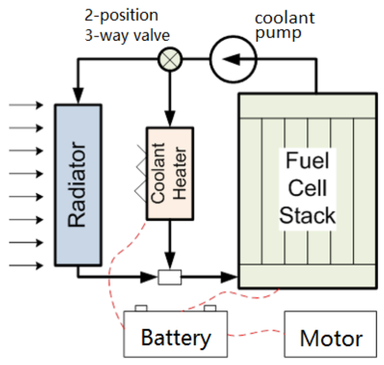


Fig. 1: Layout of the fuel cell thermal management system. The arrows in the circuit represent the direction of coolant flows.

In works such as [5], [9], requirements due to both power management and thermal management are considered in the controller design. The design approaches in these works are based on optimal control, where the requirements are combined into one objective function and the controller is developed by solving an optimal control problem with the combined objective function. The correctness of the designed controllers need to be verified by running a large number of tests.

In this paper we propose to synthesize a controller for the fuel cell thermal management system using abstraction-based formal synthesis techniques. Such techniques allow us to algorithmically generate a controller that is correct-by-construction, meaning the closed-loop system can be proved to satisfy a given specification, typically expressed in temporal logic. The idea is to create a discrete graph structure (abstraction) that captures some key properties of the given continuous state space dynamics (concrete system) and to solve the control problem on this abstraction by leveraging

LY and NO are with the Dept. of Electrical Engineering and Computer Science, Univ. of Michigan, Ann Arbor, MI 48109, USA yliren,necmiye@umich.edu. BP is with the Dept. of Mechanical and Civil Engineering, Brigham Young University, Rexburg, ID 83460, USA penceb@byui.edu. AK and MTBW are with Ford Research & Advanced Engineering, Dearborn, MI 48121, USA akarnik,mwaez@ford.com. This work is supported by Ford Motor Co.

fixed-point algorithms on graphs [17]. The main advantage of such approaches is that they avoid validating the obtained controller with exhaustive Monte-Carlo simulations, as the constructed solution is provably correct by construction (i.e., the closed-loop system is guaranteed to satisfy the specification) and the domain of validity of the controller is clearly marked. Additionally, these techniques are especially good at handling discrete actuators and hybrid dynamics: while providing correctness guarantees directly is known to be difficult for hybrid systems in general, the problem on the abstraction already has a more standard solution [17].

However, there are two main challenges in applying the existing abstraction-based synthesis techniques to the fuel cell thermal management problem. The first challenge is that computing abstractions for general nonlinear systems is expensive. In the fuel cell thermal system considered in this work, the system states and inputs are coupled in a highly nonlinear way due to the complex electrochemical reactions in the fuel cell. Hence, we need to exploit the structural properties, if any, of the system dynamics to mitigate the computational burden in computing the abstraction. The second challenge is that an abstraction may contain more behaviors than the underlying concrete system. These extra behaviors are spurious and render the synthesized controller conservative. It has been shown in [12], [16], [11] that some spurious behaviors can be removed by introducing a notion called progress groups, which capture extra transience properties of the underlying concrete system and enforce those transience properties onto the abstraction during synthesis. The scalability of the progress group computation, however, is not satisfactory when the abstraction contains a large number of control actions that come from discretizing the continuous control inputs of the concrete system. This scalability issue hence prevents us from applying the available abstraction-based synthesis techniques to fuel cell thermal management problem.

To overcome these two challenges, we extend the existing techniques in two directions. First, we prove that the fuel cell thermal management system is mixed monotone, and leverage this system property to ease the abstraction process. Second, we introduce the key notion of multi-action state-dependent progress groups, which capture a richer set of transience properties of the system under different controls, and can be used to eliminate some of the spurious behaviors from the abstraction. This new notion of multi-action progress groups improves the scalability of the synthesis procedure whenever the abstraction contains a large number of control actions.

The conference version of this paper has previously appeared as [21]. In this paper, in addition to providing a more comprehensive exposition of the problem and the solution approach, the conference version is extended in two ways. First, we consider a more complete set of requirements by considering the battery state of charge (SOC) recurrence specification, which was omitted in the early paper. Secondly, we present the pseudo code for the synthesis algorithm and give more details for computing multi-action state-dependent progress groups.

Fig. 2 summarizes the methodology used in this paper. For modeling, we adopt the fuel cell stack thermal model developed in [13] and enhance it using radiator and heater model components. For specifications, we include requirements regarding temperature targets, energy management as well as requirements for battery SOC, and formally express them in linear temporal logic (LTL). The model and specifications are further analyzed to develop a formally correct switching controller using abstraction-based synthesis.

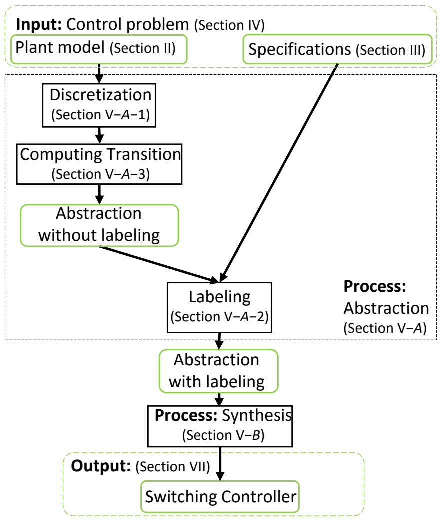

Input: Control problem (Section IV)
Plant model (Section II)
Specifications (Section III)
Discretization (Section V-A-1)
Computing Transition (Section V-A-3)
Abstraction without labeling
Labeling (Section V-A-2)
Process: Abstraction (Section V-A)
Abstraction with labeling
Process: Synthesis (Section V-B)
Output: (Section VII)
Switching Controller

Fig. 2: Methodology and paper organization.

II. Fuel Cell Model

A block diagram of the fuel cell thermal management system is shown in Fig. 3. The solid lines (red) are temperature signals, the dotted lines (purple) are power signals, the dashed lines (blue) are battery SOC signals, and the thinner solid lines (black) are control/reference input signals. The physical meanings of the variables in Fig. 3 can be found in Appendix A. Other operating conditions (such as hydrogen and oxygen partial pressure, ambient temperature, vehicle speed) that affect system dynamics are not included in the block diagram for simplicity.

In what follows, we give the formulas describing each block in Fig. 3.

A. Fuel Cell Power Generation

The fuel cell stack output power and generated heat are computed using the formulas developed in [13],

$$P_{\mathrm{FC,output}} = iA_{\mathrm{G}}E_{\mathrm{FC,stack}}, \tag{1}$$

$$P_{\mathrm{FC,self-heat}} = iA_{\mathrm{G}}\frac{\Delta h_{\mathrm{rxn}}}{2F}n_{\mathrm{FC,cell}} - P_{\mathrm{FC,output}}, \tag{2}$$

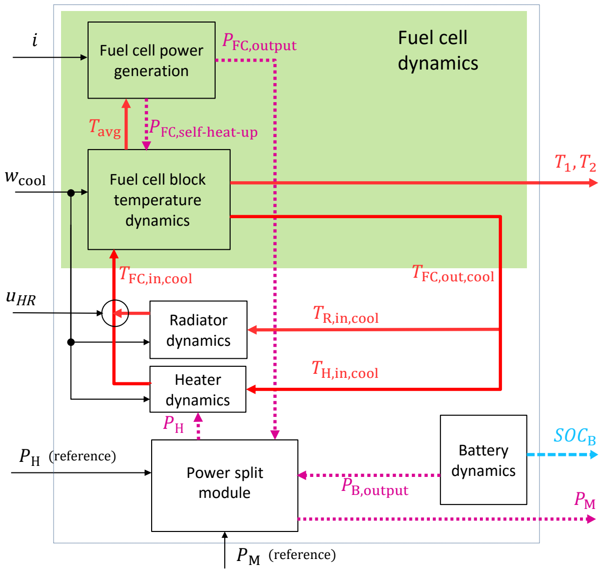


Fig. 3: Block diagram of the fuel cell thermal management system.

and

$$E_{\mathrm{FC,stack}} = n_{\mathrm{FC,cell}} \Bigg(\frac{\Delta h_{\mathrm{rxn}}}{2F} - T_{\mathrm{avg}} \frac{\Delta s_{\mathrm{rxn}}}{2F} + \frac{RT_{\mathrm{avg}}}{2F} \ln \Big(\frac{p_{\mathrm{H}_2}}{P_{\mathrm{ref}}} \big(\frac{p_{\mathrm{O}_2}}{P_{\mathrm{ref}}} \big)^{\frac{1}{2}} \Big) - \frac{RT_{\mathrm{avg}}}{\alpha F} \ln \big(\frac{i + i_{\mathrm{x}}}{i_0} \big) - iR_{\Omega} - a_{\mathrm{MT}} \big(\frac{i}{i_{\mathrm{MT}}} \big)^{b_{\mathrm{MT}}} \Bigg), \tag{3}$$

where $\frac{\Delta h_{\mathrm{rxn}}}{2F}$ and $T_{\mathrm{avg}} \frac{\Delta s_{\mathrm{rxn}}}{2F}$ correspond to the effect of enthalpies and entropies, iR_{Ω} describes Ohmic loss due to cell resistivity, and $a_{\mathrm{MT}}(\frac{i}{i_{\mathrm{MT}}})^{b_{\mathrm{MT}}}$ describes potential loss caused by mass transport limitations. The variables h_{rxn}, s_{rxn}, i_0, R_{Ω} depend on fuel cell average temperature T_{avg} and operating conditions [13].

B. Fuel Cell Temperature Dynamics

The fuel cell stack is divided into two control volumes to capture its temperature gradient. One control volume is at the coolant inlet side and the other is at the coolant outlet side. The fuel cell temperature dynamics is described in terms of the temperature of the two control volumes, i.e., T_1, T_2. The temperature dynamics are governed by the following differential equation [13]:

$$\frac{dT_1}{dt} = \frac{1}{c_{\mathrm{FC}}\rho_{\mathrm{FC}}} \Bigg(\frac{c_{\mathrm{cool}} w_{\mathrm{cool}} (T_{\mathrm{FC,in,cool}} - T_1)}{n_{\mathrm{FC,cell}} A_{\mathrm{FC}} \delta_{\mathrm{FC}}/2} + \frac{\kappa_{\mathrm{T}}(T_2 - T_1)}{(\delta_{\mathrm{FC}}/2)^2} + k_{\mathrm{amb}\to\mathrm{FC}}(T_{\mathrm{amb}} - T_1) + \frac{P_{\mathrm{FC,self-heat}}}{V_{\mathrm{FC}} n_{\mathrm{FC,cell}}} - r_{\mathrm{v}} \Delta h_{\mathrm{v}} \Bigg), \tag{4}$$

$$\frac{dT_2}{dt} = \frac{1}{c_{\mathrm{FC}}\rho_{\mathrm{FC}}} \Bigg(\frac{c_{\mathrm{cool}} w_{\mathrm{cool}} (T_1 - T_2)}{n_{\mathrm{FC,cell}} A_{\mathrm{FC}} \delta_{\mathrm{FC}}/2} + \frac{\kappa_{\mathrm{T}}(T_1 - T_2)}{(\delta_{\mathrm{FC}}/2)^2} + k_{\mathrm{amb}\to\mathrm{FC}}(T_{\mathrm{amb}} - T_2) + \frac{P_{\mathrm{FC,self-heat}}}{V_{\mathrm{FC}} n_{\mathrm{FC,cell}}} - r_{\mathrm{v}} \Delta h_{\mathrm{v}} \Bigg), \tag{5}$$

where the inlet coolant temperature $T_{\mathrm{FC,in,cool}}$ in Eq. (4) is defined as

$$T_{\mathrm{FC,in,cool}} = u_{\mathrm{HR}} T_{\mathrm{H}} + (1 - u_{\mathrm{HR}}) T_{\mathrm{R}}, \tag{6}$$

where u_{HR} is the binary variable controlling the 2-position 3-way valve. The average fuel cell temperature used in Eq. (3) is defined as $T_{\mathrm{avg}} = (T_1 + T_2)/2$, while $T_{\mathrm{FC,out,cool}}$, the outlet coolant temperature from fuel cell stack, is assumed to be equal to T_2.

C. Radiator and Heater Temperature Dynamics

The radiator and heater dynamics are given by

$$\frac{dT_{\mathrm{R}}}{dt} = \frac{1}{C_{\mathrm{R}}} \big((1 - u_{\mathrm{HR}}) c_{\mathrm{cool}} w_{\mathrm{cool}} (T_{\mathrm{FC,out,cool}} - T_{\mathrm{R}}) + c_{\mathrm{air}} \varepsilon(v) v (T_{\mathrm{amb}} - T_{\mathrm{R}}) \big), \tag{7}$$

$$\frac{dT_{\mathrm{H}}}{dt} = \frac{1}{C_{\mathrm{H}}} \big(u_{\mathrm{HR}} c_{\mathrm{cool}} w_{\mathrm{cool}} (T_{\mathrm{FC,out,cool}} - T_{\mathrm{H}}) + P_{\mathrm{H}} \big). \tag{8}$$

Note that when binary control $u_{\mathrm{HR}} = 1$ (or 0), the coolant is fed to the heater (or the radiator). The term $\varepsilon(v)$ in radiator dynamics is the vehicle-speed-dependent effectiveness of the radiator, which is modeled as an affine function of vehicle speed v. The outlet coolant temperature from the radiator (the heater, respectively) is assumed to be T_{R} (T_{H}, respectively).

D. Battery SOC Dynamics

The battery SOC dynamics is adopted from that given in [8],

$$\frac{dSOC_{\mathrm{B}}}{dt} = -n_{\mathrm{s}} n_{\mathrm{p}} E_{\mathrm{B,cell}} \frac{E_{\mathrm{B,cell}} - \sqrt{E_{\mathrm{B,cell}}^2 - \frac{4 P_{\mathrm{B,output}} r_{\mathrm{B,cell}}}{n_{\mathrm{s}} n_{\mathrm{p}}}}}{2 r_{\mathrm{B,cell}} G_{\mathrm{B,stack,total}}}. \tag{9}$$

Note that in Eq. (9), $P_{\mathrm{B,output}}$ can be negative, meaning charging the battery.

E. Power Split Module

The power split module combines the output power from the fuel cell and the battery, and passes part of the combined power to the heater, and the remaining portion to the motor. To deliver the required power to the motor, we assume the battery always provides the right amount of power to compensate for what is generated by the fuel cell, that is,

$$P_{\mathrm{B,output}} = P_{\mathrm{M}} + P_{\mathrm{H}} - P_{\mathrm{FC,output}}. \tag{10}$$

III. Specifications

In this section, we give the specifications (or requirements) of fuel cell thermal management. The listed specifications are classified into the following three types:

(i) "reach-stay" type specifications require that the system variables (e.g., state or control input) reach a target region in finite time and stay in that region once they arrive;
(ii) "avoid" type specifications require that the variables avoid some undesired regions forever (or equivalently, the variables always stay in the complement of the undesired region);
(iii) "recurrence" type specifications require that the variables visit a region repetitively.

More formally, these specifications are expressible with a fragment of LTL. The fragment used is defined by the following grammar:

$$\varphi ::= \pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \Box\varphi, \tag{11}$$

where π is from a given set of atomic propositions, and $\neg$, $\vee$, $\Box$ are logical operators. An atomic proposition is a statement on system and environment variable whose truth value can be determined by checking whether the associated variables are within given sets. Operator $\neg$ and $\vee$ correspond to negation and conjunction in boolean logic. We write $\varphi \wedge \psi$ (conjunction) and $\varphi \rightarrow \psi$ (implication) as abbreviations for the formulas $\neg(\neg\varphi \vee \neg\psi)$ and $\neg\varphi \vee \psi$, respectively. Operator $\Box$ is a temporal operator, and $\Box\varphi$ means that an LTL formula φ holds true for all time. We also use the short-hand notation $\Diamond\varphi$ for $\neg\Box\neg\varphi$, which states that there exists a time instant when φ eventually holds. With the LTL fragment defined above, the three types of requirements involved in fuel cell thermal management are expressed in the following forms:

(i) $\varphi_{\text{reach-stay}} = \Diamond\Box\pi_{\text{target}}$, where π_{target} is a proposition saying that the variable belongs to a designated target set. The formula $\Diamond\Box\pi_{\text{target}}$ says that a time instant exists starting from which π_{target} is always true.

(ii) $\varphi_{\text{avoid}} = \Box\pi_{\text{safe}}$, where proposition π_{safe} says that the variable is not in the undesired set, or the variable is in the desired set.

(iii) $\varphi_{\text{recurrence}} = \Box\Diamond\pi_{\text{recurrence}}$, where $\pi_{\text{recurrence}}$ says the variable belongs to a recurrence target set. The formula $\Box\Diamond\pi_{\text{recurrence}}$ is interpreted as: for all time instants, a time exists in the future at which $\pi_{\text{recurrence}}$ holds, therefore guaranteeing repetition.

For the remainder of this section, we give the fuel cell thermal management specifications in plain English and express them in LTL. The resulting LTL formulas are listed in TABLE I.

A. Limitations of Fuel Cell Output Power

Some requirements regarding fuel cell output power are imposed in this part.

Spec1: (avoid) The fuel cell output power $P_{\text{FC,output}}$ should not drop below zero.

Fig. 4 gives the fuel cell output power predicted by the model in Section II-A. As show in Fig. 4, the model-predicted fuel cell output power becomes negative when the current density is too high, which makes the model invalid at that value of current density. This requirement is essential to avoid operating in the region where the model is invalid.

Spec2: (avoid) Let i^* be the fuel cell current density that gives the maximum fuel cell output power, graphically illustrated by Fig. 4. The current density should not exceed the one that gives the maximum output power $P_{\text{FC,max}}$ because operating above $P_{\text{FC,max}}$ is inefficient and could lead to irreversible degradation [6]. Note that i^* is a function of state and operating conditions.

Based on Fig. 4, it is obvious that *Spec2* actually implies *Spec1*. In this work, we exclude *Spec2* from the correct-by-construction synthesis because of the difficulty in computing

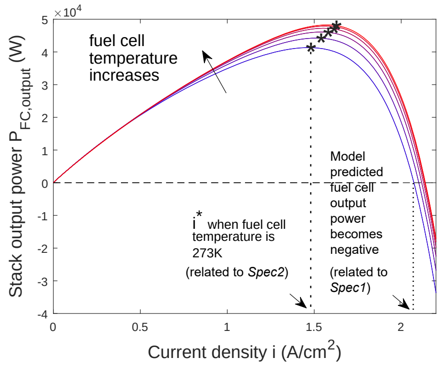


Fig. 4: Fuel cell power versus current density, fuel cell average temperature varies in [273, 360]K, membrane water content $\lambda = 6$.

i^*. Instead, we handle *Spec2* by restricting the current density i to be smaller than a fixed upper bound $\widetilde{i^*}$, which is found experimentally. In addition, some control action selecting heuristics are developed to incorporate *Spec2* (see section VI). However, since no formal guarantee can be made for achieving *Spec2*, we still consider *Spec1* in the correct-by-construction synthesis as a hard constraint.

B. Battery Energy & Power Limitations

In this part, we give some requirements regarding the battery SOC and power. These requirements are important for guaranteeing the health of the battery and its incorporation with energy management.

Spec3: (avoid) The battery stack energy should not drop below 10% or exceed 90%.

Spec4: (recurrence) Battery energy should always recover to $SOC_{\text{B,target}}$ (with at most an error δ) in finite time, where $SOC_{\text{B,target}}$ is a set point given by the energy management module.

This specification can be viewed as a relaxation of the charge sustaining requirement of the battery. Since we do not assume any knowledge of the driving cycle in advance, it is impossible for the battery SOC to recover to the starting level exactly at the end of the driving period (unless one restricts the battery SOC to always stay close to the target level, which is a conservative strategy). Hence, we require only that the battery SOC have the capability of recovering to the target level.

Spec5: (avoid) The battery power should not exceed peak power requirements.

Spec6: (avoid) Power for the battery charge should not exceed maximum allowable charging power. Note that by our convention, charging powers (both $P_{\text{B,output}}$ and $P_{\text{B,charge,max}}$) are negative.

C. Regular Operation Requirements

The following requirements are related to fuel cell temperature regulation.

Spec7: (reach-stay) Fuel cell block temperatures should reach and then stay in the target temperature range $[340, 350]$K.

By this requirement, when the fuel cell is temporarily shut down and motor power is completely delivered by the battery, we still want the fuel cell temperature to stay in the range.

Spec8: (avoid) Fuel cell block temperatures should never exceed the maximum allowable temperature of 353K.

TABLE I: Specifications in LTL

Specification	LTL formula	type	
Spec1	$\varphi_1 = \Box(P_{\text{FC,output}} \geq 0)$	avoid	✓
Spec2	$\varphi_2 = \Box(i \leq i^*)$	avoid	$\sim$
Spec3	$\varphi_3 = \Box(0.1 \leq SOC_{\text{B}} \leq 0.9)$	avoid	✓
Spec4	$\varphi_4 = \Box\Diamond\big(SOC_{\text{B,target}} - \delta \leq SOC_{\text{B}} \leq SOC_{\text{B,target}} + \delta\big)$	recurrence	✓
Spec5	$\varphi_5 = \Box(P_{\text{B,output}} \leq P_{\text{B,output,max}})$	avoid	✓
Spec6	$\varphi_6 = \Box(P_{\text{B,output}} \geq P_{\text{B,charge,max}})$	avoid	✓
Spec7	$\varphi_7 = \Diamond\Box\big(\wedge_{j=1,2}\,(T_j \in [340, 350])\big)$	reach-stay[1]	✓
Spec8	$\varphi_8 = \Box\big(\wedge_{j=1,2}\,(T_j \leq 353)\big)$	avoid	✓

"✓": the specification is considered in the synthesis.
"$\sim$": the specification is handled by heuristics.

IV. Problem Statement

In this section we formally state the control problem, by summarizing the plant model given in section II, and selecting a set of requirements defined in section III.

As a short notation, denote the system dynamics described in section II by

$$\frac{dx}{dt} = f(x, u, d) \tag{12}$$

where $x = [T_1, T_2, T_{\text{R}}, T_{\text{H}}, SOC_{\text{B}}]^T$ denotes the state, $u = [i, w_{\text{cool}}, u_{\text{HR}}, P_{\text{H}}]^T$ denotes the control, $d = [P_{\text{M}}, p_{\text{O}_2}, p_{\text{H}_2}, T_{\text{amb}}, v, \lambda, r_{\text{v}}]^T$ denotes the operating condition that is beyond our control. In particular, vector field f is defined by Eqs. (4) to (9). Let X, U, D denote the domains for x, u, d. Sets X, U, D are rectangular sets defined in Appendix A. From now on, we will refer to the model in Eq. (12) as fuel cell system for short.

In this work we consider all requirements listed in Section III except *Spec2*. In addition to the selected requirements, define LTL formula

$$\varphi_9 = \Box(x \in X),$$

to constrain the system states to never leave the considered domain X, and define assumptions on the environment

$$\varphi_{\text{env}} = \Box(d \in D),$$

to assure that the operating conditions always stay in the allowable range. Then the overall specification for consideration in LTL is given by

$$\Phi := \varphi_{\text{env}} \rightarrow \bigwedge_{\substack{i=1 \\ i \neq 2}}^{9} \varphi_i, \tag{13}$$

[1]We refer to this LTL specification as "reach-stay" type with a slight abuse of terminology. In fact, $\Diamond\Box(x \in \text{target set})$ does not require that x stays in the target set after its first arrival.

so that if the environment variables (operating conditions) always remain in their allowable ranges, all the selected specifications are satisfied.

Problem 1: Given the fuel cell model defined in (12), and desired closed-loop behavior specified by LTL formula Φ, defined by (13), synthesize a feedback controller with finite memory $K : X \rightarrow U$, under which all closed-loop trajectories governed by $\dot{x} = f(x, K(x), d)$, satisfy the LTL specification Φ.

V. Solution Approach

We formulate the control problem as a temporal logic game [14] on a hybrid system and solve the game using abstraction-based synthesis technique. This section is divided into three parts. First, we describe the basic steps involved in computing an abstraction, and show how to leverage the system's properties at each step to simplify computation. The system properties and related results can be found in Appendix B. Second, we briefly describe the synthesis process on the abstraction. Finally, we motivate and propose multi-action state-dependent progress groups, and show how they remove spurious behavior from the abstraction.

A. Abstraction

The abstraction process returns a finite transition system for a given plant model and specifications. The transitions capture the flow of the continuous plant dynamics, and the (discrete) states of the finite transition system are properly labeled according to the given specifications. In this work, the finite transition systems serving as abstractions are nondeterministic. That is, given the current state and the control action applied at that state, there might be multiple succeeding states. Specifically, we also reinforce such transition systems with so called progress groups, which encode additional transience properties of the underlying concrete system. Such transition systems are already well-established in the literature. We refer the reader to [11] for a formal definition of such abstractions and detailed algorithms generating them.

As shown in Fig. 2, the abstraction process is decomposed into three steps, that is, discretization, labeling and transition computation. We now describe each step, incorporated with the fuel cell system properties for computational efficiency.

1) Discretization: We first partition the state space of a given concrete system into finitely many regions. Each region is mapped to a discrete state in the finite transition system. In the rest of this paper we will call a "discrete state" as "state" for short, when the context is unambiguous. We use a manually constructed non-uniform rectangular grid partition in the state space. A rectangular partition reduces the abstraction computation effort significantly when the system flow/vector field are (mixed) monotone [4] or multi-affine [23].

Notice that both continuous-valued control inputs $(i, w_{\text{cool}}, P_{\text{H}})$ and boolean control input (u_{HR}) are present in the system. We discretize control space U by creating a grid on the continuous-valued control space, which leads to a finite set of control actions. It should be noted that the discretization of the continuous control variables leads to more than 70 control actions in total in this application.

2) Labeling: After the state space partition, each region in the partition needs to be labeled as "target", "recurrence target", "safe" and "unsafe" according to the specifications.

Consider "reach-stay" specification *Spec7*. The regions contained by set $\{x \in X \mid T_1 \text{ and } T_2 \in [340, 350]\}$ are labeled as "target". For "recurrence" specification *Spec4*, the regions contained by set $\{x \in X \mid SOC_{\mathrm{B}} \in [SOC_{\mathrm{B,target}} - \delta, SOC_{\mathrm{B,target}} + \delta]\}$ are labeled as "recurrence target". The remainder of the specifications are of the "avoid" type. A region is labeled "safe" if the "avoid" specification is satisfied everywhere in that region for all operating conditions; it is labeled "unsafe" if the specification is violated somewhere in the region for some operating conditions.

The challenge is that some "avoid" specifications are implicitly related to states and operating conditions. For example, requirement *Spec1* requires fuel cell output power $P_{\mathrm{FC,output}} \geq 0$ (or equivalently $E_{\mathrm{FC,stack}} \geq 0$ by Eq. (1)), $P_{\mathrm{FC,output}}$ is a function of both system state (fuel cell temperature T_{avg}) and operating condition (membrane water content λ, hydrogen-oxygen partial pressure $p_{\mathrm{H}_2}, p_{\mathrm{O}_2}$). Therefore, to label a region safe or unsafe in terms of *Spec1*, we need to check the worst case in that region. That is, if the minimum fuel cell output power $P_{\mathrm{FC,output}}$ (or equivalently $E_{\mathrm{FC,stack}}$) in the region is negative under some operating conditions (which violate specification *Spec1*), the region is labeled unsafe.

As described in Section II-A, $E_{\mathrm{FC,stack}}$ is a nonlinear function in state x and operating condition d. Therefore, determining the exact minimum value of $E_{\mathrm{FC,stack}}$ requires solving a nonlinear optimization problem over x and d, which might be intractable. However, function $E_{\mathrm{FC,stack}}(x, d)$ allows an efficient and reasonable approximation by Theorem 1 in Appendix B if the considered regions are rectangles. Theorem 1 applies to function $E_{\mathrm{FC,stack}}(x, d)$ because $E_{\mathrm{FC,stack}}(x, d)$ is continuously differentiable w.r.t. x and d on compact set $X \times D$. This means that all the continuous partial derivatives $\frac{\partial E_{\mathrm{FC,stack}}}{\partial x}$, $\frac{\partial E_{\mathrm{FC,stack}}}{\partial d}$ are bounded on $X \times D$; thus, $E_{\mathrm{FC,stack}}(x, d)$ satisfies the hypothesis of Theorem 1.

By Theorem 1, under-(over-)approximating the minimum (or maximum) value of $E_{\mathrm{FC,stack}}$ reduces to evaluating $E_{\mathrm{FC,stack}}$ at the two extreme points of the considered rectangular region. The result of the approximation is illustrated using Fig. 5: the dashed line is the maximum and minimum value when x_1, x_2, or T_1, T_2 varies in $[273, 360]$K. Fig. 5 shows a gap between the approximated minimum value of $E_{\mathrm{FC,cell}}$ and the true values. This gap indicates that the approximation is conservative. However, when the size of the region to be labeled is smaller, the approximation gets tighter.

3) Computing Transitions: We compute transitions in the abstraction by arguing about the vector field directions of the concrete system, over a region in state space, and also over all operating conditions. In this part we provide an efficient way to compute these transitions using Theorems 1 and 2 in

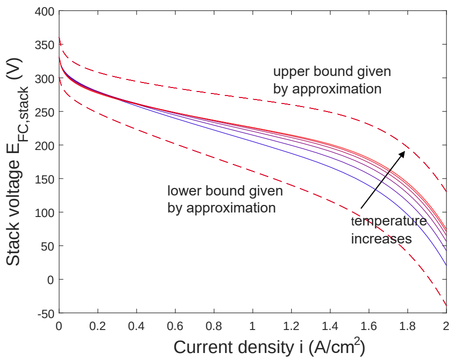


Fig. 5: Approximation of polarization, fuel cell average temperature varies in [273, 360]K.

Appendix B.

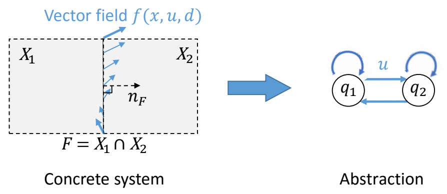


Fig. 6: Computing transitions by arguing direction of vector field.

As shown in the left half of Fig. 6, X_1 and X_2 are two adjacent regions in the state space of concrete system, $F = X_1 \cap X_2$ is the adjacent facet between two regions, dashed arrow n_F is the normal vector of facet F (pointing from X_1 to X_2), and the solid arrows on F are the vector field $f(x, u, d)$ under some given u and operating conditions d. The right half of the figure shows the discrete states in the abstraction, in particular, discrete state q_1 (q_2) corresponds to region X_1 (X_2), and the transitions between q_1 and q_2 are defined as follows

$$q_1 \xrightarrow{u} q_2 \text{ if } \max_{\substack{x \in F, \\ d \in D}} n_F^T f(x, u, d) > 0. \tag{14}$$

Finally, since any trajectory starting from region X_i will stay in that region after a small amount of time, we need to add a self-transition on the corresponding discrete state q_i.

Assume a rectangular partition of the state space, the adjacent facets are all rectangular facets, i.e., $F = \{x \in X \mid x_j \in [\underline{x}_j, \overline{x}_j]\}$, and their normal vectors are natural basis vectors e_i (a vector whose i^{th} entry is one, and the other entries are zeros). Also note that allowable operating condition set D is a rectangular set by definition, i.e., $D = \{d \mid d_k \in [\underline{d}_k, \overline{d}_k]\}$. Eq. (14) is thus equivalent to

$$q_1 \xrightarrow{u} q_2 \text{ if } \max_{\substack{x_j \in [\underline{x}_j, \overline{x}_j] \\ d \in [\underline{d}_k, \overline{d}_k]}} f_i(x, u, d) > 0. \tag{15}$$

The optimum values in (15) can be over approximated using Theorem 1. Fixing control u, and letting ϕ^u be the decomposition function of $f(\cdot, u, \cdot)$ defined by Eq. (22), we have

$$\max_{\substack{x_j \in [\underline{x}_j, \overline{x}_j] \\ d \in [\underline{d}_k, \overline{d}_k]}} f_i(x,u,d) \leq \phi_i^u([\overline{x}, \overline{d}], [\underline{x}, \underline{d}]), \tag{16}$$

where $\underline{x} = [\underline{x}_1, \ldots, \underline{x}_n]^T$ and $\overline{x} = [\overline{x}_1, \ldots, \overline{x}_n]^T$ (similar for $\underline{d}$, $\overline{d}$). We hence replace Eq. (16) by the following to compute the transitions:

$$q_1 \xrightarrow{u} q_2 \text{ if } \phi_i^u([\overline{x}, \overline{d}], [\underline{x}, \underline{d}]) > 0. \tag{17}$$

By Remark 1 in Appendix B, if the partial derivative $\frac{\partial f_i}{\partial x_j}$ is not sign-stable, the approximations in Eq. (16) are not tight. In other words we may have $\phi_i^u([\overline{x}, \overline{d}], [\underline{x}, \underline{d}]) > 0$ but $\max f_i(x,u,d) \leq 0$. In that case, Eq. (16) and Eq. 17 are not equivalent. In fact, we create more transitions when using Eq. (17), and hence introduce more spurious behavior in the abstraction, thus leading to a more conservative solution but conserving the correctness.

Note that the partial derivative $\frac{\partial f_3}{\partial v}$ is not sign-stable, where $f_3 = \frac{dT_\mathrm{R}}{dt}$ is defined by Eq. (7). The sign of $\frac{\partial f_3}{\partial v}$ depends on which one of T_R and T_amb is larger. In this case, the conservatism can be reduced by using Theorem 2 in Appendix B. We show f_3 is affine in state x and multi-affine in $[T_\mathrm{amb}, w]$ where $w := \varepsilon(v)v$. By Theorem 2, to maximize (minimize, respectively) vector field component f_3, one need only evaluate f_3 at both upper and lower bounds of w, and pick the maximum (minimum, respectively) f_3 value. This practice is equivalent to evaluating f_3 at the upper and lower bounds of vehicle speed v, because $w = \varepsilon(v)v$ is monotone increasing in v. With this modification, Eq. (16) becomes

$$\max_{\substack{x_j \in [\underline{x}_j, \overline{x}_j] \\ d \in [\underline{d}_k, \overline{d}_k]}} f_i(x,u,d) \leq$$
$$\max\left\{\phi_i^u([\overline{x}, \overline{\overline{d}}], [\underline{x}, \underline{d}]),\ \phi_i^u([\overline{x}, \overline{d}], [\underline{x}, \underline{\underline{d}}])\right\}, \tag{18}$$

where $\underline{\underline{d}}$ is the same as $\underline{d}$ except that its fifth entry (representing vehicle speed v) takes upper bound value; and $\overline{\overline{d}}$ is the same as $\overline{d}$ except that its fifth entry takes lower bound value.

Note that to compute the transition from q_2 to q_1, the only change is to choose the normal vector in Eq. (14) to be $n_F = -e_i$, and the above approximation process still applies to this case.

B. Synthesis

To synthesize a controller, we solve a temporal logic game on the obtained abstraction using graph search algorithms. As mentioned at the beginning of Section V-A, the abstraction used in this work is nondeterministic. The actual evolution of such an abstraction can be viewed as the outcome of a two player game between the controller and the environment [18]. In each round of the game, the controller selects an action first, and then the environment selects a transition that is available under the current state and action. The goal of the controller is to win the game, that is, to satisfy the specification regardless of the moves of the environment. The algorithms for solving such games are fairly standard [18], [3], [10], [20], [11], [2], and refer the readers to these references for details. Fig. 7 shows an illustration of the synthesis process, and we only briefly describe the process with the following three steps:

1. We first solve the "stay" part of the game, by searching for the maximal controlled invariant set C_1 within the discrete states labeled as both "target" and "safe". Each discrete state in C_1 will be assigned a set of control actions. Under the assigned actions, the closed-loop path starting from C_1 will stay in C_1 forever. Such a maximal controlled invariant set can be found by a fixed-point algorithm given in [10].
2. Next we solve the "recurrence" part of the game, restricted within set C_1. In particular, at each state in C_1, we can use only the actions assigned to that state in step 1 that render C_1 to be invariant. The recurrence game can be solved with the algorithm given in [20]. This gives a set of states contained by C_1, starting from where the "recurrence target" states (again, in C_1) can be reached infinitely often. This set of states is denoted as C_2. As shown in Fig. 7, $C_2 \subseteq C_1$. It should be noted that C_2 is also a controlled invariant set when the recurrence specification is achieved.
3. Then we solve the "reach-avoid" part of the game, using the algorithm given in [10]. The solution contains a set of states B, called the backwards reachable set of C_2, together with a set of actions assigned to each state in B. If these actions are applied accordingly, a path starting from B will reach C_2 in finite time, while avoiding "unsafe" states[2]. The obtained backwards reachable set B together with set C_2 form a winning set of the overall game.

Finally, once the winning set and the associated control actions are determined on the abstraction, we can extract a switching controller for the concrete system, that is, we map the actions assigned to each "winning" state to its corresponding region in the concrete system's state space. This leads to a look-up table controller defined in the concrete system state space.

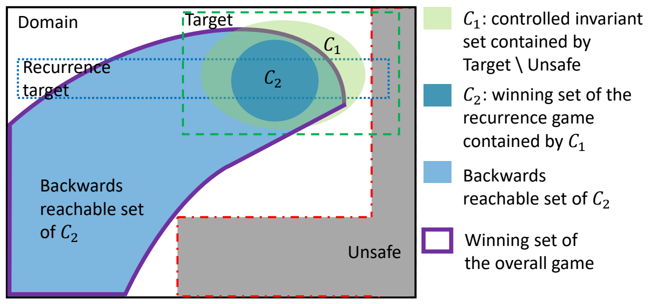
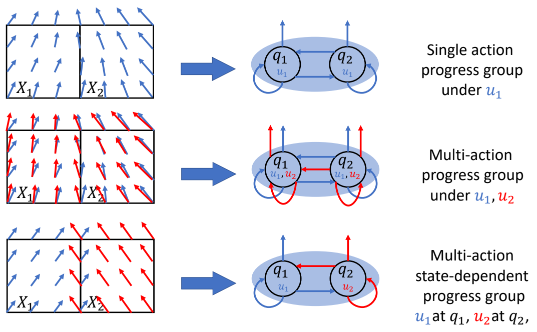

Domain
Target
C_1
Recurrence target
C_2
Backwards reachable set of C_2
Unsafe
C_1: controlled invariant set contained by Target \ Unsafe
C_2: winning set of the recurrence game contained by C_1
Backwards reachable set of C_2
Winning set of the overall game

Fig. 7: Illustration of synthesis procedure.

To solve the reachability part, both in the reach-avoid-stay game and the recurrence game, the existence of spurious

[2]As noted in Footnote 1, there is a difference between specification $\Diamond\Box(x \in \text{target set})$ and "reaching the target set in finite time and staying there once arrived". Here we are achieving the latter one, which implies the satisfaction of the former one.

loops in abstraction may prevent the target from being reached, therefore reducing the winning set. Hence, it is crucial to eliminate the spurious behavior in the abstraction as much as possible. To this end we encode in abstraction some transient properties of the underlining continuous system by progress groups. A set of states (each state assigned a set of control actions) forms a progress group if these discrete states correspond to a transient region in the original concrete system, under the assigned actions. A region is transient under some control actions if all trajectories starting from that region eventually leave the region in finite time under assigned control actions. Fig. 8 illustrates different notions of progress groups. In all three illustrations, the vector field in region $X_1 \cup X_2$ is pointing upwards under the assigned actions. Thus, all trajectories starting from region $X_1 \cup X_2$ will eventually leave the region. We hence group the corresponding states $\{q_1, q_2\}$ as a progress group, and forbid any infinite path from staying within states $\{q_1, q_2\}$. Notice that infinite paths within $\{q_1, q_2\}$ exist otherwise due to the self-loops and cycling between two states. The novel concept introduced here of multi-action state-dependent progress group will be motivated and explained in more details in the following section.

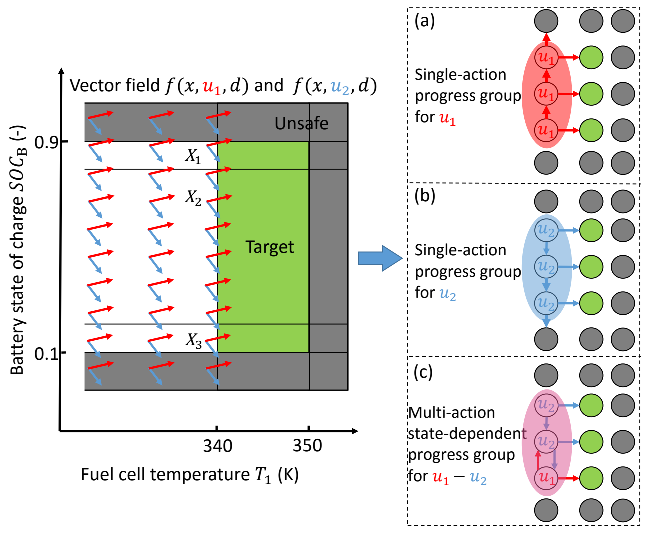


X_1
X_2
q_1
q_2
Single action progress group under u_1
Multi-action progress group under u_1, u_2
Multi-action state-dependent progress group u_1 at q_1, u_2 at q_2,

Fig. 8: Different notions of progress groups.

C. *Multi-action State-dependent Progress Group*

In previous works [12], [16], progress groups were defined only for a single action. In a more recent paper [11], the notion of multi-action progress groups was introduced to encode richer transience properties of the underlying concrete system. In all of these works, progress groups were pre-computed before synthesis and are stored as part of the abstraction. These notions of progress groups, however, are not sufficient for the specific application considered in this paper. First, we find that single-action progress groups cannot accommodate the battery SOC requirement and some reachability requirements at the same time. Moreover, pre-computing and storing multi-action progress groups requires an unacceptable computation load because the number of actions in our application is relatively large. Hence, we develop a procedure here to construct multi-action progress groups on-the-fly during synthesis. This procedure leads to a more general notion called multi-action state-dependent progress groups.



Vector field $f(x, u_1, d)$ and $f(x, u_2, d)$
Unsafe
Target
X_1
X_2
X_3
0.9
0.1
Battery state of charge SOC_B (-)
340
350
Fuel cell temperature T_1 (K)
(a)
Single-action progress group for u_1
(b)
Single-action progress group for u_2
(c)
Multi-action state-dependent progress group for $u_1 - u_2$
u_1
u_2

Fig. 9: Illustration of system flow projected onto SOC_B-T_1 subspace (left), abstraction with single-action progress group (right (a), (b)) and abstraction with multi-action state-dependent progress group (right (c)). The self-loops on the discrete states are removed because they are part of some progress groups and hence correspond to transient regions.

We use Fig. 9 to illustrate why single-action progress groups are not sufficient for this application, that is, why the battery SOC requirement *Spec3* and the reachability part of requirement *Spec7* cannot be satisfied at the same time with single-action progress groups. On the left side of the figure, we plot the rectangular partition and the system's vector field projected onto SOC_B-T_1 space. By requirement *Spec3*, the regions where battery SOC falls below 0.1 or exceeding 0.9 are labeled as unsafe (gray). By requirement *Spec7*, the fuel cell temperature should reach and stay between 340K to 350K, and the corresponding regions are labeled as target (green). To reach the target region, we can either (i) let the fuel cell do self-heat-up, meanwhile using the excess power generated to charge the battery (corresponding to action u_1), or (ii) use the heater to warm up the fuel cell, by drawing power from the battery (corresponding to action u_2)[3]. Note that no action can keep battery SOC constant while steering the fuel cell temperature towards the target. This is true due to the variation in motor requested power P_M, which is an operating condition (uncontrolled variable).

The right section of the Fig. 9 shows the abstraction, where the colored ellipses mark some progress groups and the arrows show the paths when the corresponding actions are applied. By choosing a single action (i.e., case (a) or (b)), the paths either go all the way up (case (a)) or down (case(b)) and lead to unsafe discrete states. Therefore battery SOC requirement *Spec3* is violated on the abstraction. Note that such paths are spurious because they do not represent any real trajectories of the concrete system (e.g., choosing u_1 at low battery SOC actually leads the trajectory into the target

[3] In practice there are more than 70 actions in total, but we only plot the system's flow under two typical actions u_1 and u_2 to simplify the illustration.

region before saturating the battery). Such spurious behaviors exist in abstraction due to the conservatism introduced by partitioning. On the other hand, the battery SOC requirement can be satisfied by applying multiple actions. As shown in Fig. 9 (c), all the paths stay safe when u_1 is applied at the bottom discrete state and u_2 is applied at the upper two discrete states. However, the reachability requirement is violated by an infinite loop caused by alternatively choosing u_1 and u_2. Again, this loop is spurious, as there is a constant flow towards the left no matter what action is chosen. We thus need multi-action (not necessarily state-dependent) progress groups to eliminate such loops when they are spurious.

Since the total number of multi-action progress groups grows exponentially in the number of available control actions (70 for this application), it easily exhausts time and memory to pre-compute these progress groups and encode them in the abstraction before synthesis. Therefore, instead of doing computation and storage before synthesis, we compute progress groups with multiple actions during the synthesis process, and we restrict the control actions on-the-fly based on the synthesis. As will be shown later, under such restrictions, the control actions assigned to different discrete states in the progress group may vary from one to the other, that is, the action is state-dependent. Algorithm 1 shows how to construct such progress groups and how to use them to compute a backwards reachable set of the target set.

Algorithm 1 $[B] = \text{BackwardsReach}(C, S, \alpha)$
Compute the safe backwards reachable set of C, with multi-action state dependent progress groups constructed on-the-fly during the computation.

Input: the set C of discrete states to reach, the set S of all discrete states labeled as safe, abstraction α that maps a region in concrete system's state space to a discrete state.
Output: Set B, a backwards reachable set of set C.

```
1:  $B = C$
2:  $P = \textbf{Pre}^1(B)$
3:  $i = 0$
4:  while $B$ is not satisfactory and $i \leq$max_iter do
5:      $(B', K) = \textbf{CInv}\big(B \cup (P \cap S)\big)$
6:      if $\exists v : \forall b \in B' \setminus B : \exists u_b \in K(b) : \forall x \in \alpha^{-1}(b), d \in D : v^T f(x, u, d) > 0$ then
7:          $B' \setminus B$ is a multi-action state-dependent progress group whenever $u_b$ is applied at $b \in B' \setminus B$
8:          $B = B'$
9:          $P = \textbf{Pre}^1(B)$
10:     else
11:         Replace $P$ by a proper subset of $P$ that has not been used before
12:     end if
13:     $i = i + 1$
14: end while
15: return $B$
```

Algorithm 1 is summarized with the following steps:

1. We start from an initial set B and compute its one-step-predecessors $P = \textbf{Pre}^1(B)$. A discrete state p is called a one-step-predecessor of a set B, if there is a transition (under some actions assigned to p) leading p to some discrete state $b \in B$. Here, since we want to remain in the target set after arriving at it, set B is initialized as a controlled invariant set contained by the target.
2. Next, sub-procedure $(B', K) = \textbf{CInv}\big(B \cup (P \cap S)\big)$ computes the largest controlled invariant set B' that is contained by set $B \cup (P \cap S)$, together with an invariance controller $K : B' \to 2^U$ that maps every discrete state in B' to a set of control actions that renders B' invariant.
3. If the discrete states in $B' \setminus B$ correspond to a transient region under some actions restricted by invariance controller K (this is checked by a sufficient condition in line 6), set $B' \setminus B$ form a multi-action state-dependent progress group, and is added to the backwards reachable set.
4. We repeat steps 1, 2, 3 until the winning set reaches a satisfactory size or a maximum number of iterations has been reached.

We explain the intuition behind Algorithm 1 as follows. For simplicity, we will temporarily omit the "avoid" requirements. Suppose that B is indeed a backward reachable set of some controlled invariant set C, the following facts must hold:

- Set B is computed in a backwards expanding manner starting from and containing set C; hence, the paths satisfying the reach-stay requirement will always stay within set B. Thus, set B is controlled invariant under the actions that achieve the reach-stay requirement.
- For C to be reachable in finite time, no path stays in $B \setminus C$ forever. That is, set $B \setminus C$ will form a progress group under the actions that achieve the reach-stay requirement.

Therefore, in order to find a progress group that helps expanding the backwards reachable set of C, one need only explore the controlled invariant sets B' that contain C, and restrict to the action assignments that render B' invariant. These restrictions avoid exploring all subsets of discrete states with all possible action combination when computing multi-action state-dependent progress groups.

The remainder of this part explains the condition in line 6, Algorithm 1 and shows how to check this condition. Take Fig. 9 as an example. Shaded discrete states form a multi-action state-dependent progress group when u_2 is assigned to the two discrete states on the top and u_1 is assigned to the bottom discrete state. This is because the region represented by these discrete states is transient under corresponding actions. The transience can be checked efficiently by arguing the direction of the vector field of the underlying concrete system. As shown on left side of Fig. 9, the union of three regions $X_1 \cup X_2 \cup X_3$ is transient because the horizontal component of the vector field is always positive (i.e., pointing rightwards) when control u_2 is applied in X_1 X_2, and u_1 is applied in X_3. More generally, given a set of regions

$\{X_k\}_{k=1}^m$ in n dimensional state space, each region equipped with one control action u_k, $X = \bigcup_{k=1}^m X_k$ is transient under assigned actions if there exists $v \in \mathbb{R}^n$,

$$\forall k = 1 \ldots m : \min_{x \in X_k, d \in D} v^T f(x, u_k, d) > 0. \tag{19}$$

If vector $v = \pm e_i$ (the vector with the i^{th} entry being 1 and the rest being 0), and if X_k's are rectangles, the optimization problem in (19) can be approximated efficiently by the approach developed in section V-A.3.

VI. Results and Discussion

Using the solution approach described in Section V, a switching controller is synthesized. The controller is in the form of a look-up table (see Step 3 in Section V-B). At each time instant, the current state locates in one of the regions of the look-up table and the control action in that region is applied accordingly. Although multiple actions might be available in the region, selecting an arbitrary one is sufficient to guarantee the correctness. We implement a "lazy switching" heuristic for action selection to reduce the change in the control inputs, allowing us to always maintain the previously used control action, where possible. Whenever we enter a new region in the look-up table and the previous action is no longer available there, we change the action but try to maintain the position of as many actuators as possible. In particular, we can assign different priority to different actuators in terms of maintaining their current positions. If there is no such priority, we always tend to fix the actuator whose position has changed most recently. This practice helps to balance the position change of different actuators, avoiding frequent switches. The action selecting heuristics can be also designed to incorporate with specification *Spec2*, which is excluded from the correct-by-construction synthesis. Since *Spec2* requires that the fuel cell current density i is smaller than i^*, we can always select the control actions with the smallest current density i to avoid violating *Spec2*.

The controller is shown to be able to achieve the specifications on the entire state domain X. That is, the computed winning set is equal to the domain X. We illustrate the closed-loop behaviors by simulating the system at 285K. We modify FTP-72 vehicle speed data [19] to obtain motor power P_{M} and vehicle speed v profile in a driving cycle. First, since the controller allows the vehicle speed v to vary only in [5,25]m/s, we saturate the v whenever its value falls below 5m/s. Second, P_{M}, the power requested by the motor, is assumed to be proportional to acceleration $\frac{dv}{dt}$ whenever the acceleration is positive, and is assumed to be 0 whenever the acceleration is negative. P_{M} is also scaled so that the value lies in the allowable range, that is, $[0, 17]$kW.

The simulation results are plotted in Figure 11, from which we make the following observations:

1. By plots (1-1) (1-2) (5-2), all states stay in the domain, and the battery SOC never exceeds upper or lower bounds.
2. By plot (1-1) fuel cell temperature reaches and stays in the target range (marked by a dashed green line).

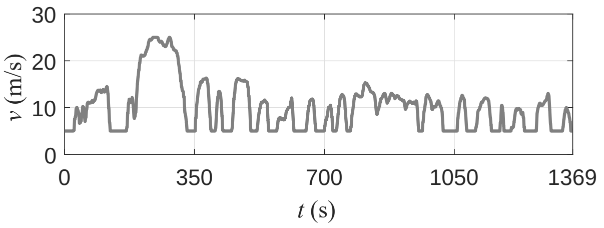


Fig. 10: FTP-72 vehicle speed data, saturated to fit the operating condition constraints of the synthesized controller.

3. By plot (5-2), the battery SOC recurrently visits the reference interval marked by the dashed blue lines. Here we assume the reference can vary over time, and is determined by a higher level power management module. Note that the battery SOC need not stay in the interval after arriving, even if the reference has not yet changed: it simply maintains the capability to recover to the desired level. This can be seen from the simulation before 300s. Such behavior is desired because it gives more freedom to the controller, while guaranteeing that the battery SOC is able to recover. For example, while the fuel cell stack does self-heat-up at the warm-up stage, it generates more power than requested with the extra power stored in the battery. This explains why the battery SOC increases and exceeds the reference interval at the beginning of the simulation. However, once the target temperature is reached, the battery SOC begins to drop towards the reference interval.
4. By plot (2-2), it can be seen that the heater is not turned on at the warm-up stage. The fuel cell does self-heat-up instead, indicating that while the controller is not optimal in terms of warm-up speed, it does not harm the correctness of the controller.

VII. Conclusions

In this paper, we developed a control-oriented model for a fuel cell thermal management system, collected a set of requirements and compiled them in linear temporal logic formulas. We then formulated the control problem as a temporal logic game and synthesized a correct-by-construction switching controller using abstraction-based techniques. The major difference between the approach in this work and that in existing work is that we provide a provably correct controller and the set of all initial conditions from which this controller guarantees requirement satisfaction. This set turns out to be the entire domain X. On the other hand, we do not consider optimality. Some properties of the system model under consideration were identified to make the synthesis computationally efficient. Existing synthesis algorithms with progress groups were also extended to tackle the problem of a large number of actions in this specific application. Finally, the correctness of the closed-loop system with synthesized controller, as suggested by the theory, was illustrated by simulations.

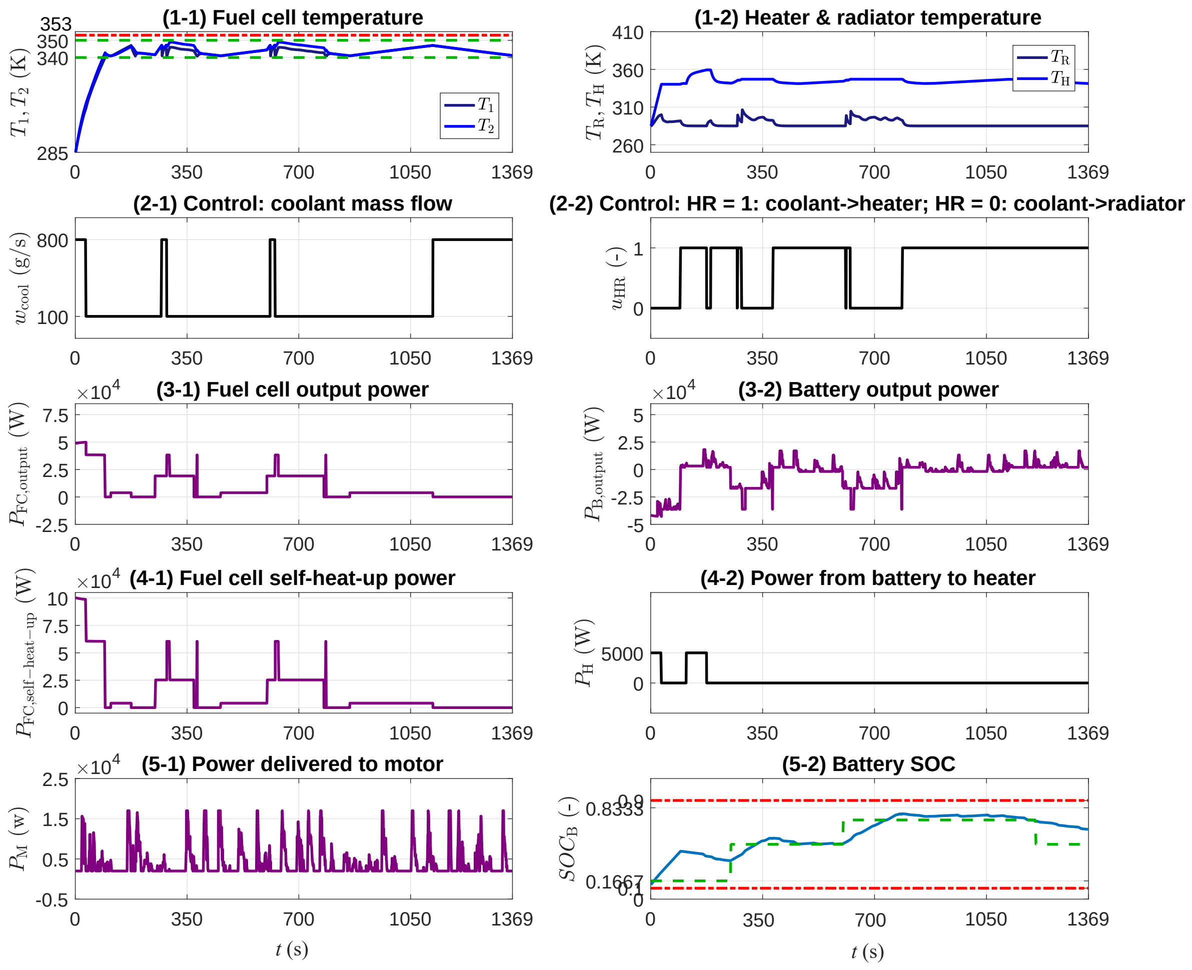


Fig. 11: Simulations results: states, powers and selected controls. Temperature states start from 285K and battery SOC starts from 0.105.

Appendix

A. Variables and Constants

Control u

u_{HR}	u_{HR} = 1	indicating that the coolant flow goes through the heater
	u_{HR} = 0	indicating that the coolant flow goes through the radiator
i	[0,1.5]	(A cm^{-2}) Cell current density
P_H	$[0, 35000]$	(W) Power requested by heater
w_{cool}	[0,800]	(g s^{-1}) Coolant mass flow rate

State x

SOC_B	[0,1]	(-) Battery energy
T_1	[273, 360]	(K) Temperature of first control volumes
T_2	[273, 360]	(K) Temperature of second control volumes
T_H	[250, 400]	(K) Heater temperature
T_R	[250, 340]	(K) Radiator temperature

Operating Condition d

P_M	$[2, 17]$	(kW) Power requested by motor
p_{O_2}	5×10^4	(Pa) Oxygen partial pressure
p_{H_2}	1.5×10^5	(Pa) Hydrogen partial pressure
r_v	$[0, 10^{-7}]$	(mol cm^{-3} s^{-1}) Volumetric evaporating rate
v	[10,20]	(ms^{-1}) Vehicle speed
T_{amb}	[273,290]	(K) Ambient temperature
λ	[4,22]	(-) Membrane water content

Other Variables

$E_{FC,stack}$	(V) Fuel cell stack electrical potential
i_0	(A cm^{-2}) Exchange current density
$P_{B,output}$	(W) Battery output power
$P_{FC,output}$	(W) Fuel cell output power
$P_{FC,self-heat}$	(W) Power for fuel cell self-heat-up
R_Ω	(Ω cm^2) Cell resistivity
T_{avg}	(J mol^{-1} K^{-1}) Average fuel cell temperature
$T_{FC,in,cool}$	(K) Inlet coolant temperature (into fuel cell)
$T_{FC,out,cool}$	(K) Outlet coolant temperature (from fuel cell)
Δh_{rxn}	(J mol^{-1}) Reaction enthalpy
Δh_v	(J mol^{-1}) Evaporation enthalpy
Δs_{rxn}	(J mol^{-1} K^{-1}) Reaction entropy

Constants

c_{air}	(1.0 J g^{-1}K^{-1}) Air specific heat capacity
F	(96485 C mol^{-1}) Faraday constant
P_{ref}	(101325 Pa) Reference pressure
R	(8.314 J mol^{-1} K^{-1}) Universal gas constant

Parameters

a_{MT}	(V) Mass transfer potential loss coefficient
A_{FC}	(cm^2) Fuel cell cross section area
A_{G}	(cm^2) Fuel cell geometric area
b_{MT}	(-) Mass transfer potential loss exponent
c_{cool}	(J g^{-1}K^{-1}) Coolant specific heat capacity
c_{FC}	(J g^{-1}K^{-1}) Fuel cell specific heat capacity
C_{H}	(J K^{-1}) Heater heat capacity
C_{R}	(J K^{-1}) Radiator heat capacity
$E_{\mathrm{B,cell}}$	(V) Battery cell open-circuit potential
$G_{\mathrm{B,stack,total}}$	(Ws) Battery stack energy capacity
$i_{0,\mathrm{ref}}$	(A cm^{-2}) Reference exchange current density
i_{MT}	(A cm^{-2}) Mass transfer current density
i_{x}	(A cm^{-1}) Crossover current density
$k_{\mathrm{amb}\to\mathrm{FC}}$	(W cm^{-3} K^{-1}) Heat transfer coefficient: ambient to stack
$k_{\mathrm{amb}\to\mathrm{H}}$	(W cm^{-3} K^{-1}) Heat transfer coefficient: ambient to heater
n_{p}	(-) Number of battery cells in parallel
n_{s}	(-) Number of battery cells in series
$n_{\mathrm{FC,cell}}$	(-) Number of fuel cells in stack
$r_{\mathrm{B,cell}}$	(Ω) Battery cell internal resistance
V_{FC}	(cm^3) Fuel cell volume
α	(-) Charge transfer coefficient
δ_{FC}	(cm) Channel or cell length
κ_T	(W cm^{-1} K^{-1}) thermal conductivity
ρ_{FC}	(g cm^{-3}) Fuel cell density

B. Preliminary Results of System Properties

This part provides some useful results for making abstraction computation efficient. The proofs can be found in [22].

Theorem 1: Assume $f : \mathbb{R}^n \to \mathbb{R}^m$ is differentiable, and

$$\frac{\partial f_i}{\partial x_j}(x) \in [a_{ij}, b_{ij}], \forall x \in X \subseteq \mathbb{R}^n, \tag{20}$$

where a_{ij} and b_{ij} are finite real numbers, set $X = \{x \in \mathbb{R}^n \mid x_j \in [\underline{x}_j, \overline{x}_j]\}$ is a rectangle, then the following inequality holds in element-wise sense:

$$\phi(\underline{x}, \overline{x}) \le f(x) \le \phi(\overline{x}, \underline{x}), \forall x \in X, \tag{21}$$

where $\underline{x} = [\underline{x}_1, \ldots, \underline{x}_n]^T$ and $\overline{x} = [\overline{x}_1, \ldots, \overline{x}_n]^T$, and function $\phi : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^m$ is defined to be:

$$\begin{aligned} &\forall i \in 1 \ldots, m: \\ &\phi_i(x, y) = f_i(z) + (\alpha_i - \beta_i)^T (x - y). \end{aligned} \tag{22}$$

In Eq. (22), $z = [z_1, \ldots, z_n]^T$, $\alpha_i = [\alpha_{i1}, \ldots, \alpha_{in}]^T$, $\beta_i = [\beta_{i1}, \ldots, \beta_{in}]^T$ are n vectors defined as follows

$$z_j = \begin{cases} x_j & \text{if } b_{ij} \ge |a_{ij}| \\ y_j & \text{otherwise} \end{cases} \tag{23}$$

$$\alpha_{ij} = \begin{cases} |a_{ij}| + \epsilon & \text{if } a_{ij} \le 0, b_{ij} \ge |a_{ij}| \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

$$\beta_{ij} = \begin{cases} -|b_{ij}| - \epsilon & \text{if } b_{ij} \ge 0, a_{ij} \le -|b_{ij}| \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

where ϵ is a small positive number.

Remark 1: Theorem 1 is related to mixed monotonicity of function f, and function ϕ is called a decomposition function of f. The decomposition function constructed above is a natural extension of the one given in [4], which only handles f with sign-stable partial derivatives. The idea here is to use linear terms to create an additional offset to overcome the sign-unstable partial derivatives. In the case where all the partial derivatives $\frac{\partial f_i}{\partial x_j}$ are sign-stable, the decomposition function constructed by Theorem 1 gives a tight approximation in Eq. (21); that is, the inequality in Eq. (21) reduces to equality at some $x \in X$ [4]. However, this is not true when there are sign-unstable partial derivatives. Thus, in general, the approximation given by (21) might be conservative.

Theorem 2: (Theorem 1 in [23]) Let $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^p$ be a function affine in the first argument $x \in \mathbb{R}^n$ and multi-affine [1] in the second argument $d \in \mathbb{R}^m$, i.e., $f(x, d) = A(d)x + K(d)$, where $A(d)$, $K(d)$ are matrixes whose entries are in form of

$$\sum_{p_1,\ldots,p_m \in \{0,1\}} c_{p_1,\ldots,p_m} \prod_j^m (d_j)^{p_j}. \tag{26}$$

Let $X \subseteq \mathbb{R}^n$ be a polytope, and $D \subseteq \mathbb{R}^m$ be a rectangle. Define V_X and V_D to be the set of vertices of set X and D. Then the maximum and minimum values of f on $X \times D$ are obtained at vertices set $V_X \times V_D$, i.e.,

$$\max_{\substack{x \in X \\ d \in D}} f(x, d) = \max_{\substack{x \in V_X \\ d \in V_D}} f(x, d), \tag{27}$$

and similarly (27) holds for minimization of f.

References

[1] C. Belta and L. Habets. Controlling a class of nonlinear systems on rectangles. *IEEE Trans. Autom. Control*, 51(11):1749–1759, 2006.

[2] C. Belta, B. Yordanov, and E. A. Gol. *Formal Methods for Discrete-Time Dynamical Systems*, volume 89. Springer, 2017.

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

[4] S. Coogan and M. Arcak. Efficient finite abstraction of mixed monotone systems. In *Proceedings of International Conference on Hybrid Systems: Computation and Control*, pages 58–67. ACM, 2015.

[5] N. Henao, S. Kelouwani, K. Agbossou, and Y. Dubé. Proton exchange membrane fuel cells cold startup global strategy for fuel cell plug-in hybrid electric vehicle. *Journal of Power Sources*, 220:31–41, 2012.

[6] K. Jiao and X. Li. Water transport in polymer electrolyte membrane fuel cells. *Progress in Energy and Combustion Science*, 37(3):221–291, 2011.

[7] S. G. Kandlikar, Z. Lu, and T. A. Trabold. Current status and fundamental research needs in thermal management within a pemfc stack. *in ASME Journal of Fuel Cells Science and Technology*, 2008.

[8] S. J. Moura, D. S. Callaway, H. K. Fathy, and J. L. Stein. Impact of battery sizing on stochastic optimal power management in plug-in hybrid electric vehicles. In *Vehicular Electronics and Safety (ICVES), 2008 IEEE International Conference on*, pages 96–102. IEEE, 2008.

[9] E. A. Muller, A. G. Stefanopoulou, and L. Guzzella. Optimal power control of hybrid fuel cell systems for an accelerated system warm-up. *IEEE transactions on control systems technology*, 15(2):290–305, 2007.

[10] P. Nilsson and N. Ozay. Incremental synthesis of switching protocols via abstraction refinement. In *Decision and Control (CDC), 2016 IEEE Conference on*, pages 6246–6253. IEEE, 2014.

[11] P. Nilsson, N. Ozay, and J. Liu. Augmented finite transition systems as abstractions for control synthesis. *Discrete Event Dynamic Systems*, 27(2):301–340, 2017.

[12] N. Ozay, J. Liu, P. Prabhakar, and R. Murray. Computing augmented finite transition systems to synthesize switching protocols for polynomial switched systems. In *Proceedings of American Control Conference*, pages 6237–6244, 2013.

[13] B. L. Pence and J. Chen. A framework for control oriented modeling of pem fuel cells. In *ASME 2015 Dynamic Systems and Control Conference*, pages V002T26A002–V002T26A002. American Society of Mechanical Engineers, 2015.

[14] N. Piterman, A. Pnueli, and Y. Saar. Synthesis of reactive (1) designs. In *Proceedings of the International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 364–380, 2006.

[15] J. T. Pukrushpan, A. G. Stefanopoulou, and H. Peng. *Control of fuel cell power systems: principles, modeling, analysis and feedback design*. Springer Science & Business Media, 2004.

[16] F. Sun, N. Ozay, E. M. Wolff, J. Liu, and R. M. Murray. Efficient control synthesis for augmented finite transition systems with an application to switching protocols. In *Proceedings of American Control Conference*, pages 3273–3280, 2014.

[17] P. Tabuada. *Verification and control of hybrid systems: a symbolic approach*. Springer, 2009.

[18] W. Thomas et al. *Automata, logics, and infinite games: a guide to current research*, volume 2500. Springer Science & Business Media, 2002.

[19] United States Environmental Protection Agency. Dynamometer drive schedules. Available at `https://www.epa.gov/vehicle-and-fuel-emissions-testing/dynamometer-drive-schedules`.

[20] E. M. Wolff, U. Topcu, and R. M. Murray. Efficient reactive controller synthesis for a fragment of linear temporal logic. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5033–5040. IEEE, 2013.

[21] L. Yang, A. Karnik, B. Pence, M. T. B. Waez, and N. Ozay. Fuel cell thermal management: Modeling, specications and correct-by-construction control synthesis. In *Proceedings of American Control Conference*, pages 1839–1846, 2017.

[22] L. Yang and N. Ozay. A note on some sufficient conditions for mixed monotone systems. Technical report, University of Michigan, Department of EECS, 2017. Available at `http://hdl.handle.net/2027.42/136122`.

[23] L. Yang, N. Ozay, and A. Karnik. Synthesis of fault tolerant switching protocols for vehicle engine thermal management. In *Proceedings of American Control Conference*, pages 4213–4220, 2016.