Formal Verification and Simulation: Co-Verification for Subway Control Systems

Huixing Fang*, Jian Guo*†, Huibiao Zhu*, Jianqi Shi*
**Shanghai Key Laboratory of Trustworthy Computing,*
Software Engineering Institute, East China Normal University, Shanghai, China
†*State Key Laboratory for Novel Software Technology, Nanjing University, China*
Email: {wxfang, jguo, hbzhu, jqshi}@sei.ecnu.edu.cn

Abstract—**For hybrid systems, hybrid automata based tools are capable of verification while Matlab Simulink/Stateflow is proficient in simulation. In this paper, a methodology is developed in which the formal verification tool PHAVer and simulation tool Matlab are integrated to analyze and verify hybrid systems. For application of this methodology, a Platform Screen Doors System (abbreviated as PSDS), a subsystem of the subway, is modeled with formal verification techniques based on hybrid automata and Matlab Simulink/Stateflow charts, respectively. The models of PSDS are simulated by Matlab and verified by PHAVer. It is verified that the sandwich situation can be avoided under time interval conditions. We conclude that this integration methodology is competent in verifying Platform Screen Doors System.**

I. Introduction

A hybrid system consists of digital controllers within a continuous environment. Hybrid automata have been proposed to model and verify hybrid systems [1], [2]. The reachability problem of verification of hybrid systems is important and challenging [3], [4], [5], [6], [7]. For linear hybrid systems, some model-checking algorithms have been developed [1], [8], [9]. The tool HyTech [10] is focuses on linear hybrid systems.

Moreover, PHAVer [11] is another tool for the exact verification of hybrid systems with piecewise constant bounds on the derivatives. PHAVer adopts exact and robust arithmetic based on the PPL (Parma Polyhedra Library) [12]. Piecewise affine dynamics with linear hybrid automata are handled by on-the-fly over-approximation. For more complex hybrid systems, in [13], experimental results indicate that SpaceEx can cope with hybrid systems with more than 100 variables which illustrate the scalability of the tool.

Furthermore, simulation based tools are widely used in industry. The most important tool is the Simulink/Stateflow toolset. We can model continuous-time systems in Stateflow charts. Continuous-time modeling allows us to simulate hybrid systems which respond to both continuous and discrete mode changes.

In this paper, we propose the Feedback-Advancement Verification methodology to integrate verification with simulation of subway control systems. Using Matlab toolset with Simulink and Stateflow we will benefit from its rich library of primitive components. Moreover, it provides modeling and simulation capabilities for complex systems, such as non-linear, continuous-time, discrete-time, multi-variable, and multi-rate systems. However, we are not sure about the correctness of the systems for unpredicted interactions with the environment. Formal method based tools (e.g., HyTech, PHAVer) avoid the flaw of simulation. However, to acquire appropriate analysis commands for analysis, it needs to experiment on verification repeatedly, as a result, it is time-consuming for complex systems. Simulation provides visual and concrete perspective which can be used as feedback to support the verification and improve our verification progress. Therefore, it is appropriate to combine simulation with formal verification.

We adopt tools, PHAVer and Matlab Simulink/Stateflow for verification and simulation of Platform Screen Doors System. Firstly, we construct the models in both tools, and then we take verification and simulation on liveness properties, respectively. During the simulation, the sandwich situation is revealed. This indicates that the passenger would be situated in the dangerous position that is between platform screen doors and train doors, therefore the passenger cannot go back to the platform. The reason for the sandwich situation is that the time interval between platform screen doors closed and train doors closed is too short to let the passenger return to the platform. We refine the models and simulate this sandwich situation and then refine the models in PHAVer and verify the property. As a result, the ultimate models satisfy the time interval constraint and sandwich free property if the passenger go back to the platform during the time interval.

This paper is organized as follows. In section II, we briefly introduce the hybrid automata, and the train automaton of subway control system as an example to depict the concepts of hybrid automata. In section III, we propose the Feedback-Advancement Verification methodology which integrates verification with simulation. Section IV describes the requirements in the Platform Screen Doors System. And in Section V, the subway system is modeled with hybrid automata firstly, and then with Simulink/Stateflow charts. Section VI simulates and verifies these models with Matlab toolset and PHAVer, respectively. We analyze the verification results and refine the models in section VII. We conclude our methodology and discuss future works in Section VIII.

II. Fundamentals of Hybrid Automata

In this section, we give a brief introduction about hybrid automata, including linear hybrid automata and affine hybrid automata. We also provide one example about the hybrid automaton of train travelling. A hybrid system is modeled by hybrid automata. A hybrid automaton consists of a finite control graph that represents the nondeterministic evolution of real-valued variables over time and discrete switches (transitions) with jump conditions. The nodes of the finite control graph are called control modes, and the edges are called control switches.

A. Hybrid Automata Definition

A hybrid automaton is a tuple $A = (X, V, \mathit{flow}, \mathit{inv}, \mathit{init}, E, \mathit{jump}, \Sigma, \mathit{syn})$ [9], where: $X = \{x_1, x_2, \cdots, x_n\}$ is a finite set of real-valued variables, where n is the dimension of A. Control modes V is a finite set of control modes. For each control mode $v \in V, \mathit{flow}(v)$ is a predicate over the variables in $X \cup \dot{X}$, where $\dot{X} = \{\dot{x_1}, \dot{x_2}, \cdots, \dot{x_n}\}$ and $\dot{x_i}(1 \leq i \leq n)$ is the first derivative of x_i with respect to time. Once the current control mode is v, the variables in X and their time-derivatives satisfy the flow condition $\mathit{flow}(v)$. For each control mode $v \in V$, the invariant condition $\mathit{inv}(v)$ for mode v is a predicate over the variables in X. When the control mode is v, $\mathit{inv}(v)$ must be satisfied by the variables in X. For each control mode $v \in V$, the initial condition $\mathit{init}(v)$ is a predicate over the variables in X. All initial states must satisfy their initial condition. For source control mode $v \in V$ and target control mode $v' \in V$, if the hybrid automaton A has a transition from v to v', then $(v, v') \in E$ is a control switch, note that E is a multi-set. For each control switch $e \in E$, the jump condition $\mathit{jump}(e)$ is a predicate over the variables in $X \cup X'$. For $1 \leq i \leq n$, $x_i \in X$ and $x'_i \in X'$ refer to the value of the variable x_i before and after the control switch, respectively. Σ is a finite set of events. For each control switch $e \in E$, the synchronization label $\mathit{syn}(e)$ is an event in Σ.

If v is a control mode and $r = (r_1, \cdots, r_n) \in \mathbb{R}^n$ is a value of variables X, then the pair (v, r) is a state of the hybrid automaton A. Since the invariant condition $\mathit{inv}(v)$ is a predicate over the variables in X, therefore the state (v, r) is admissible only if $\mathit{inv}(v)$ is true when the value of the variable x_i is r_i, for $1 \leq i \leq n$. We will illustrate this by an example in the next subsection.

B. Example: train automaton

Consider a train in the subway control systems. When the distance between train and station is -1000, a sensor signals its approach to the controller (as an arbitrator in the PSDS), then the train approaches the station at a speed of $v \in [0, 16]$ and the acceleration $\dot{v} = -0.128$, indicates that the train will slow down. And, when the distance is -0.5, the train sends the $\mathit{near\_stop}$ signal to the controller, and the train will stop. Both distance and speed are reset to zero when the train stops. When the train leaves the station, its speed $v \geq 0$ and acceleration $\dot{v} = 0.128$, the train will speed up. The train is modeled by the automaton shown in Fig. 1.

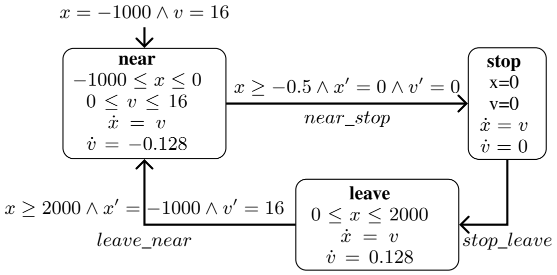


Figure 1. Train automaton.

The set of real-valued variables of the train automaton is $X = \{x, v\}$, where x is the distance between the train and the station, v is the speed of the train.

The train automaton has three control modes and the set of control modes is $V = \{\mathit{near}, \mathit{stop}, \mathit{leave}\}$. The flow condition of the control mode near is $\mathit{flow}(\mathit{near}) = (\dot{x} = v) \wedge (\dot{v} = -0.128)$. The invariant condition of the control mode near is $\mathit{inv}(\mathit{near}) = (-1000 \leq x \leq 0) \wedge (0 \leq v \leq 16)$. If real vector $r = (r_1, r_2)$, where $r_1, r_2 \in \mathbb{R}$, r_1 denotes the value of variable x, r_2 denotes the value of variable v, if $r = (-1000, 16)$, the state (near, r) is the initial state of the train automaton, and (near, r) is admissible in mode near according to $\mathit{inv}(\mathit{near})$.

The pair $cs = (\mathit{stop}, \mathit{leave})$ is a control switch of the train automaton, the control switch $(\mathit{stop}, \mathit{leave})$ corresponds to the event $\mathit{stop\_leave}$. The jump condition (omitted in Fig. 1) of the control switch cs is $x = 0 \wedge v = 0 \wedge x' = 0 \wedge v' = 0$. The pair (p, q) is a jump of the train automaton if $p = (\mathit{stop}, r)$ and $q = (\mathit{leave}, r')$, where $r = (0, 0)$ and $r' = (0, 0)$.

III. Co-Verification Methodology

The formal verification of hybrid system is based on the reachability analysis with hybrid automata. The formal method based tools (e.g., HyTech, PHAVer) have the ability to provide parametric analysis and analysis commands to users, and generate error-trace. On the other hand, using Matlab toolset with Simulink and Stateflow, many benefits can be achieved from its rich library of primitive components. Simulation data is visualized in the Matlab toolset and we can take advantage of its interactive features. Moreover, it provides modeling and simulation capabilities for complex systems. With simulation, we are not sure about the correctness of the systems because of the unpredicted interactions with environment. On the contrary, formal method based tools (e.g., HyTech, PHAVer) avoid the flaw of simulation that cannot guarantee design correctness. However, appropriate analysis commands are not easy to acquire, it needs to experiment with verification many times, hence it is time-consuming for complex hybrid systems.

Therefore, it is feasible to verify and analyze hybrid systems by combining formal verification with simulation. The following methodology called Feedback-Advancement Verification (FAV, Fig. 2) is proposed. This method is composed of three phases: base phase, exploration phase and conclusion phase.

1) **Base phase:** Construct models by formal method (e.g., hybrid automata) and simulation technology (e.g., Simulink/Stateflow models), respectively. Liveness and non safety-critical properties are checked in this phase. The models are modified according to the results of simulation and verification. The main objective of this phase is to check whether the models are correct or not in a coarse-grained manner. Therefore, the properties to be checked in this phase are simple and not concerning safety. The complex and safety-critical properties are checked in exploration phase.
2) **Exploration phase:** In this phase, safety critical properties or complex interactive situations of the system are researched with the combination method of formal verification and simulation. In terms of the conclusion of base phase, if errors or problems occur during verification, then we modify formal models and proceed with verification. This process is repeated until the model satisfies the safety properties or the process of verification is failed because of the limitation of tool or resources. On the other hand, if errors or problems occur during simulation, then we modify simulated models and proceed with simulation. Based on the capability of verification tools (e.g. state exploration), some complex critical properties cannot be verified in verification tools. As a result, it shall turn to simulation finally. For simulation, one cannot simulate all the interaction with environment, and it is required to verify the correctness of the refined models in verification tools.
3) **Conclusion phase:** After the second phase, the final model of formal verification or simulation is obtained and can be applied to new developing stages in model-based design.

The exploration phase is used to verify the safety properties and provide a correct model. Moreover, formal verification and simulation could influence each other by feedback.

In the following, we utilize our method FAV to model and verify the subway control system. We first illustrate requirements of the subway control system, and then model requirements and verify models in both tools: PHAVer and Matlab toolset. With simulation, we simulate subway environment and find that the models do not satisfy the safety property because of high-level abstraction. Then we refine models and verify the correctness of the refined models with PHAVer. The visualized simulation results are also given by Matlab toolset.

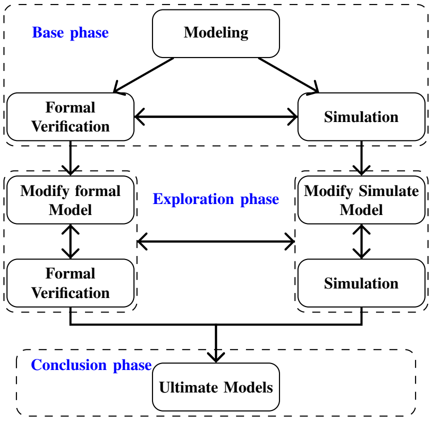


Figure 2. Feedback-Advancement Verification.

IV. Subway System Requirements

This paper focuses on the Platform Screen Doors System (PSDS). The platform screen doors are widely used at subway stations around the world. These doors screen the platform from the train, and prevent someone from falling off the platform that probably because of suicide or the piston effect [14]. The PSDS is a safety-critical system. One small design deficiency in the system may cause serious accident. For example, on 15 July 2007 in Shanghai, China, a man tried to enter a crowded train at the subway station, but failed. When the train doors and platform edge doors closed almost simultaneously, he was sandwiched between closed doors and fall off. Consider the three different schemes of closing train doors and platform screen doors:

1) **Simultaneously:** The platform screen (edge) doors and train doors close simultaneously. As we mentioned above, passengers may be sandwiched in the middle and unable to go back to the platform or enter the train. This scheme is not appropriate for the PSDS.
2) **Platform Screen Doors First (PSDF):** First, close the platform screen (edge) doors, and then the train doors. This scheme is also improper when the platform screen doors is closed but the train is crowded. Thus, some passengers failed to squash into the train, it needs to open the platform screen doors again.
3) **Train Doors First (TDF):** First, close the train doors, and then the platform screen doors. Passengers can return to the platform when the train doors are closed. Moreover, there is enough time to go back to the platform before the platform screen doors are closed.

Although it may need other measures to improve safety, we focus on the Platform Screen Doors Systems with the TDF scheme in this paper.

V. Models

As the PSDS consists of the train, the train doors, the platform screen doors and the controller (as an arbitrator). We divide the whole system into four parts, train, train doors, platform screen (edge) doors and controller.

A. Hybrid Automata Models

We use four automata to formalize the Platform Screen Doors System. The train automaton (Fig. 1) represents the travelling of the train. The train doors automaton (Fig. 3) and platform screen doors automaton represent the opening and closing of the train doors and screen doors, respectively. The controller automaton (Fig. 4) represents as an arbitrator between these three automata.

The train doors automaton (Fig. 3) has four control modes. Modes $open$ and $closed$ denote the train doors are opened and closed, respectively, modes $part$ and $shut$ denote that the train is opening doors and closing doors, respectively. Each train door consists of two sides, left side and right side. The variable y_1 represents the distance between the left side and right side. It means that the door is opened when the distance is 2 and then $y_1 = 2$. Similarly, if the door is closed then the distance is 0 and the value of y_1 is 0. Initially, the door is closed, and after receiving an $open_1$ command from the controller, the door will be opened at a rate of 1. When the door is opened and receives a $close_1$ command, it will be closed at a rate of dy_1, where the value of the variable dy_1 ranges between -1 and 0. When the rate is zero but the door is not closed, someone or something is clipped. Thus the door needs to be opened again and then after the door is open, passengers leave the door and the controller also needs to close the door again. The variable c_1 is a guard to re-close the door. We assign 1 to c_1 when the door receives the $close_1$ command. We check the value of c_1 when the door is open. If $c_1 = 1$, close the door again.

The platform screen doors automaton is the same as the train doors automaton except variables. These two automata have similar behavior. They are managed by the controller.

The controller automaton (Fig. 4) plays a role of an arbitrator. It receives signals from train, sends commands to train, train doors and platform screen doors. Initially, the controller is at control mode $idle$. When it receives the signal $near\_stop$, the controller enters the mode $about\_to\_open_2$. Five seconds later, it sends command $open_2$ to the platform screen doors and waits 5 seconds. The platform screen doors begin to open when it receives command $open_2$. Then the controller sends command $open_1$ to the train doors. At the same time, the train doors begin to open. Six seconds later, the controller starts to ring (as an alert). After the alert off, the controller waits 5 seconds and then sends command $close_1$ to train doors. Train doors begin to close. Five seconds latter, the controller sends command $close_2$ to platform screen doors if train doors are closed ($c_1 = 0$ denotes train doors are closed). Similarly, five seconds later, it sends command $stop\_leave$ to train if platform screen doors are closed. After that, the train leaves the station and the controller returns to mode $idle$.

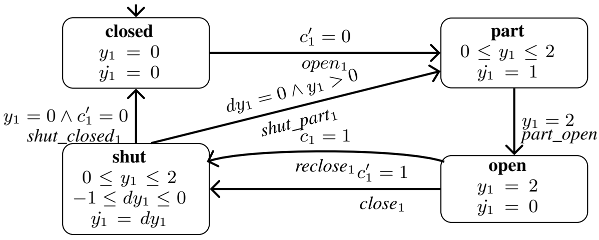


Figure 3. Train doors automaton.

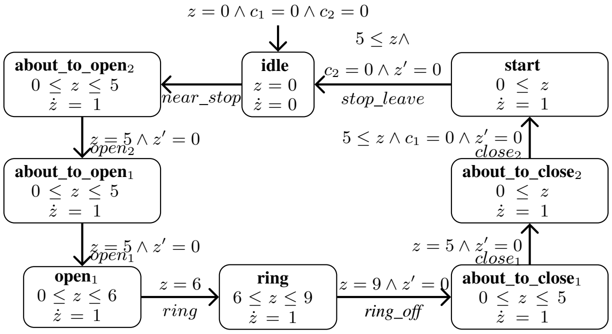


Figure 4. Controller automaton.

B. Simulation Models

In this section, we model our PSDS in Simulink/Stateflow. Our Stateflow chart consists of four subcharts. Fig. 5 represents the Stateflow chart of train. Fig. 6 represents the train doors (the chart of platform screen doors is the same as the train doors except the variable names). The controller is shown in Fig. 7.

We use two *Manual Switches* to simulate the velocity of closing doors, one for the train doors and another for the platform screen doors, they select values from -1 and 0. Accordingly, *v_shut1* and *v_shut2* are two *input variables* which represent the values of the *Manual Switches* selected. Furthermore, there are three *output variables*, *y1_out*, *y2_out* and *distance_out*. The value of *y1_out* equals *y1* (see Fig. 6) and the value of *y2_out* equals *y2*. *y1_out* and *y2_out* represent the distances between the two sides of doors, train doors and platform screen doors, respectively. The distance between train and station is represented by *distance_out*. We use the $Scope$ block to display these three *output variables* with respect to simulation time.

In the train Stateflow chart shown in Fig. 5, *distance* is a *local variable* which denotes the distance between train and station. And, *v* is also a *local variable* which denotes the velocity of the train. The symbol *distance_dot* represents the time derivative of the variable *distance*, therefore its value equals the velocity of train which is denoted by *v*. Similarly, the symbol *v_dot* represents the time derivative of *v*, thus, *v_dot* denotes the acceleration of the train. In addition, "en:" is the prefix for entry actions which execute when the state is entered, and, "du:" is the prefix for during actions in which the derivatives and updates for variables are defined, the during actions are executed at every time step when the sate is active and no valid transition to another state is available.

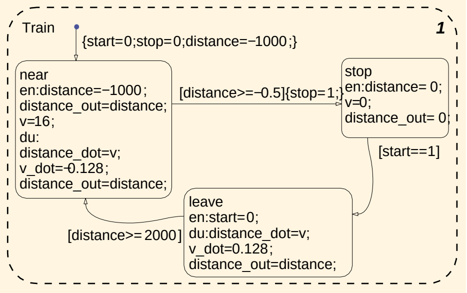


Figure 5. Train chart

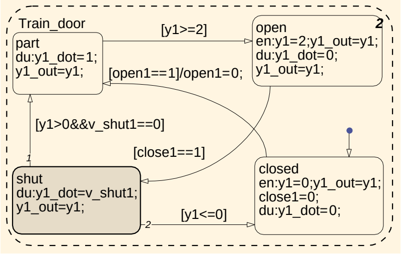


Figure 6. Train doors chart

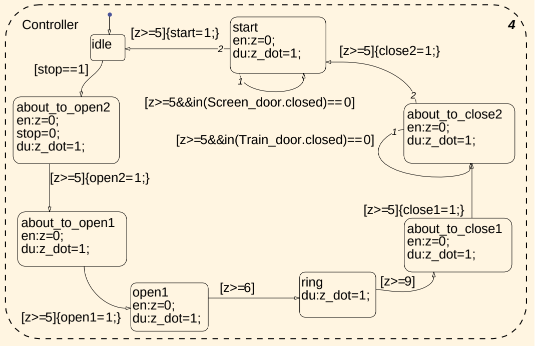


Figure 7. Controller chart

In the Stateflow chart, we use *local variables* as shared variables between different Stateflow subcharts. The *local variables* defined in a Stateflow chart can be read and written in the chart's all subcharts. For example, the variable *stop* is used in train chart and controller chart. The condition action $stop = 1$ is executed as soon as the condition $distance >= -0.5$ of the transition from state *near* to *stop* is true. And then in the controller chart, the condition $stop == 1$ is true. Therefore the transition from state *idle* to *about_to_open2* will be active. Similarly, variable *start* is also the shared variable between train and controller. Variable *open1* (*open2*) and *close1* (*close2*) are shared variables between train doors (screen doors) and controller.

The variable *close1* has the same purpose as c_1 (see Fig. 3) which is the guard to re-close train doors. And, the variable *open1* is the guard to open train doors. In the same way, *open2* and *close2* have corresponding purposes, respectively.

Moreover, we use the predicate *in(Train_door.closed)* == *0* in the controller Stateflow chart, indicating whether the train doors are closed. Here *train_door* is the name of the train doors chart (see Fig. 6) and *closed* is one of the states of the train doors chart. If the predicate is true, the train doors are not closed. Therefore, a self-loop transition to substates *about_to_close2* and *start* is added (see Fig. 7), and the controller continues waiting until train doors or platform screen doors are closed.

VI. Formal Verification and Simulation

Given a property P of a hybrid system, formal verification is to answer whether the hybrid system satisfies the property P. For safety-critical systems, we want to check whether a system cannot reach a set of unsafe states S_{unsafe}. Nevertheless, we do not verify the safety property of a system at first. Conversely, first of all, we check the liveness and simple properties of the system, and then the safety property. In the following, we will check four properties of PSDS by formal verification and simulation, respectively.

A. Formal Verification

PHAVer is a tool for the verification of hybrid systems with piecewise constant bounds on the derivatives. PHAVer deals with affine dynamics with linear hybrid automata by using an on-the-fly over-approximating algorithm. PHAVer is free from overflow errors by using PPL [12] and GMP (GNU Multiple Precision Arithmetic Library) [15]. In this paper, we adopt the formal verification tool PHAVer to verify hybrid automata models. The following command represents that the four automata are composed using ampersand (*&*):

sys= controller & traindoor & screendoor & train.

Thus, *sys* is the composition of the four automata which are described in section V, representing the PSDS illustrated in this paper. A *symbolic state* is denoted by a combination of a control mode name and a linear formula, for example, *stop & x==0* represents the state when the train is stop and the value of x is zero. The initial state is *idle∼closed∼closed∼near & z==0 & x==*-1000 *& v==16 & y1==0 & y2==0 & close1_flag==0 & close2_flag==0*, where mode names of automata in composition are concatenated with $\sim$. Here, *close1_flag* represents the variable c_1 in Fig. 3, and *close2_flag* represents the variable c_2. The following four liveness properties are checked:

1) **Leaving and Stopping:** Firstly, we want to know whether the train can leave from the station or stop. The states are denoted by

$$\begin{aligned} sys.\{\$\sim\$\sim\$\sim leave\&True,\\ \$\sim\$\sim\$\sim stop\&True\}.\end{aligned}$$

The identifier $\$\sim\$\sim\$\sim leave$ refers to all modes in which train automaton is in *leave*, and $\$\sim\$\sim\$\sim$ *stop* refers to those in which train automaton is in *stop*. And the identifier $True$ refers to all the possible evaluations of variables in automata. The verification of properties in PHAVer is by reachability analysis. The command *sys.reachable* returns the set of states reachable in the automaton sys from the initial states. And the command *identifier1.intersection_assign(identifier2)* intersects *identifier2* with *identifier1* and puts the result into *identifier1*. As a result, the two commands *sys.reachable* and *intersection_assign(forbidden)* can be used to verify whether the unsafe states can be reached from the initial states, here the identifier *forbidden* denotes the set of unsafe states. The verification results given by PHAVer support that the above states are reachable from the initial state.

2) **Ringing:** Both train doors and platform screen doors are opened when the bell is ringing. Consider the following states denoted by:

   $$sys.\{ring \sim \$ \sim \$ \sim \$\&True\}.$$

   The identifier $ring \sim \$ \sim \$ \sim \$$ refers to all modes in which controller automaton is in $ring$. We can list the intersection of the above states and all reachable states, and check in which modes the train doors automaton and platform screen doors automaton are. The intersection is given by PHAVer:

   $$\begin{aligned} controller \sim traindoor \sim screendoor \sim train.\{\\ ring \sim open \sim open \sim stop\&6 <= z <= 9\&\\ x == 0\&y2 == 2\&y1 == 2\&\\ close2\_flag == 0\&close1\_flag == 0\}.\end{aligned}$$

   Therefore, we are sure that both train doors and platform screen doors are open when the bell is ringing.

3) **Ordering:** The train doors need to be closed at first, and then the platform screen doors. The screen doors need to keep open when the train doors are being closed. Consider the following states denoted by:

   $$sys.\{about\_to\_close2 \sim \$ \sim \$ \sim \$\&True\}.$$

   The identifier $about\_to\_close2 \sim \$ \sim \$ \sim \$$ refers to all modes in which controller automaton is in *about_to_close*$_2$. The controller has sent command $close_1$ to the train doors when the controller automaton is in *about_to_close*$_2$. We can determine whether the property holds by checking the intersection of the above states and all reachable states. We have the following intersection by PHAVer, and the result states demonstrate that the screen doors are opened when the train doors are closing:

   $$\begin{aligned} controller \sim traindoor \sim screendoor \sim train.\{\\ about\_to\_close2 \sim shut \sim \boldsymbol{open} \sim stop\&\\ -1 <= dy1 <= 0\&0 <= y1 <= 2\&x == 0\&\\ y2 == 2\&close2\_flag == 0\&\\ close1\_flag == 1\&z + y1 >= 2,\\ about\_to\_close2 \sim open \sim \boldsymbol{open} \sim stop\&\\ x == 0\&y2 == 2\&y1 == 2\&\\ close2\_flag == 0\&close1\_flag == 1\&z >= 0,\\ about\_to\_close2 \sim closed \sim \boldsymbol{open} \sim stop\&\\ x == 0\&y2 == 2\&y1 == 0\&\\ close2\_flag == 0\&close1\_flag == 0\&z >= 2,\\ about\_to\_close2 \sim part \sim \boldsymbol{open} \sim stop\&\\ 0 < y1 <= 2\&x == 0\&y2 == 2\&\\ close2\_flag == 0\&close1\_flag == 1\&z + y1 >= 2\}.\end{aligned}$$

   Furthermore, the train doors should be closed when the controller is in mode *start*. And the states are denoted by:

   $$sys.\{start \sim \$ \sim \$ \sim \$\&True\}.$$

   In the same way, we can check the intersection of the above states and all reachable states, and then draw a conclusion.

4) **Operation of Doors:** There are no operations of doors before and after the train stop. Therefore, the following states should not be reachable:

   $$\begin{aligned} sys.\{\$ \sim open \sim \$ \sim \$\&x > 0,\\ \$ \sim part \sim \$ \sim \$\&x > 0,\\ \$ \sim shut \sim \$ \sim \$\&x > 0,\\ \$ \sim open \sim \$ \sim \$\&x < 0,\\ \$ \sim part \sim \$ \sim \$\&x < 0,\\ \$ \sim shut \sim \$ \sim \$\&x < 0\}.\end{aligned}$$

Through the formal verification with PHAVer, the above four properties are totally satisfied in our PSDS model.

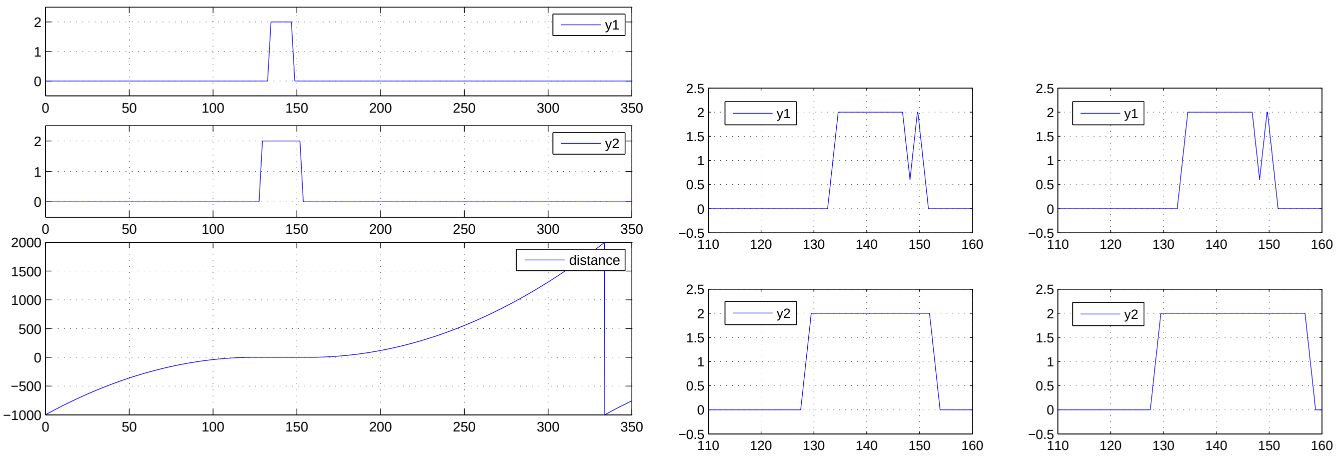
B. Simulation

The four liveness properties are verified and no error are found in PHAVer. Next, we consider the simulation of our models in the Matlab toolset. As the models have been built in Section V, we can analyze activities of train doors, screen doors and train with the *Scope* block. Fig. 8a displays one run of the train from *near* to *leave*.

In Fig. 8a, *y1* represents the activities of train doors, *y2* represents the activities of platform screen doors. Here *distance* represents the activities of the train. The *distance* ranges over $[-1000, 2000]$, and the train stops at the station when *distance* equals zero. After the train stops, *y2* increases to 2, which means that the screen doors are opened. And then, *y1* also increases to 2, which means that the train doors are opened. A few seconds later, *y1* decreases to 0, and then *y2* also decreases to 0. This means that the train doors have closed before screen doors begin to close.

In Fig. 8a, we choose the running of the system within about 350 seconds period, in this simulation, the train can leave and stop, thus, the first property *Leaving and Stopping* is satisfied. The variable *y2* equals 2 when *y1* decreases from 2 to 0. So, the first part of third property *Ordering* is satisfied. Moreover, the fourth property *Operation of Doors* is also satisfied, because all the variations of *y1* and *y2* are taken place when *distance* equals 0. The other properties can be checked by the analysis of the Stateflow chart with simulation or debug.

As mentioned before, we can simulate the situation that someone is sandwiched between closed doors. The process is described as: first, we start the Stateflow debugger and continue simulating until the variable *y1* begins to decrease. After that, we change the closing speed of train doors from -1 to 0 by double click the *train door velocity Manual Switch*, then the train doors re-open and then re-close. As



(a) One normal running.

(b) Sandwich simulation.

(c) Simulation after the refinement.

Figure 8. Simulation results of PSDS.

depicted in Fig. 8b, screen doors begin to close almost at the moment the train doors closed, so the sandwich situation occurs. We can prolong the closing time of train doors by changing the closing speed for many times to simulate the sandwich situation.

The sandwich situation and normal situation are depicted in Fig. 8b and Fig. 8a, respectively. The good point is that the platform screen doors do not begin to close before the train doors are closed. But the weak point is that the time interval named T_{dc} between the moments when train doors are closed and screen doors begin to close is smaller than it is in the normal situation. Furthermore, the platform screen doors may close immediately when the train doors are closed. Therefore, it is required to set a number of seconds (i.e., T_{dc}) for passengers to return to platform when train doors are closed. And, the PSDS hybrid automata also do not consider this time interval. As a result, we need to refine our models, not only hybrid automata but also Simulink/Stateflow charts.

VII. Refinement

In this section, we will modify the models of the controller, aiming to avoid the sandwich situation with proper time interval T_{dc}.

A. Simulation Refinement

For Stateflow chart (see Fig. 7), the self-loop transition of the state $about\_to\_close2$ contains the transition condition:

$$z >= 5 \ \&\& \ in(Train\_door.closed) == 0.$$

Let's consider the following situation. If the train doors are closed at the moment $z == 4.9$, then the predicate $in(Train\_door.closed) == 0$ is false. Thus, the transition from $about\_to\_close2$ to $start$ will occur when $z == 5$, and the time interval is 0.1s which is smaller than 5s. On the contrary, if we remove $z >= 5$ from ($z >= 5 \ \&\& \ in(Train\_door.closed) == 0$), then we just check whether the train doors are closed in the transition condition. The new transition condition makes the controller reset the variable z to 0 if the train doors are not closed. Moreover, the variable z is needed to increase from 0 to 5 (will take 5 seconds) after the train doors are closed, and then the controller starts to close platform screen doors. Fig. 8c is the simulation result. Compared with Fig. 8b, the time interval in the refined model is much larger.

However, we cannot conclude that the time interval is at least 5 seconds definitely, because we cannot simulate every operation of train doors and controller. We can solve this (i.e., verify time interval property) by formal verification tools like PHAVer in the next subsection.

B. Formal Verification Refinement

For the controller automaton (see Fig. 4), the clock named t will be placed in the control mode *about_to_close*$_2$. When the controller switches from *about_to_close*$_2$ to *start*, we keep the value of the clock t unchanged, and then we can check this value in the mode *start*. The right time to start this clock is the moment when the train doors are closed. Therefore, we add a self-loop control switch of mode *about_to_close*$_2$. The jump condition is ($z' = 0 \wedge c_1' = 0 \wedge t' = 0$), and the synchronization label of the switch is $shut\_closed_1$ which is the same as the label of the control switch from mode $shut$ to $closed$ in the train doors automaton (see Fig. 3). Thus the two control switches will be taken simultaneously. We are sure that, Similarly, for the control of screen doors, a self-loop control switch of mode *start* is added with the synchronization label $shut\_closed_2$.

Now, we need to verify the time interval property. Consider the states denoted by $sys.\{start \sim \$ \sim \$ \sim \$\&t < 5\}$. The above states should not be reachable from initial state. This means that the time interval would not be smaller than 5. On the contrary, the following states must be reachable:

$$sys.\{start \sim \$ \sim \$ \sim \$\&t \geq 5\}.$$

And, the verification results by PHAVer show that this time interval property is satisfied. The following states given by PHAVer illustrate that in the control mode $start$, the predicate $t \geq 5$ is always true:

$$\begin{array}{r}
controller \sim traindoor \sim screendoor \sim train.\{ \\
start \sim closed \sim \boldsymbol{shut} \sim stop\& \\
-1 <= dy2 <= 0\&0 <= y2 <= 2\&x == 0\&y1 == 0\& \\
close2\_flag == 1\&close1\_flag == 0\& \\
\boldsymbol{t >= 5}\&z + y2 >= 2, \\
start \sim closed \sim \boldsymbol{open} \sim stop\& \\
x == 0\&y2 == 2\&y1 == 0\&close2\_flag == 1\& \\
close1\_flag == 0\&\boldsymbol{t >= 5}\&z >= 0, \\
start \sim closed \sim \boldsymbol{closed} \sim stop\& \\
x == 0\&y2 == 0\&y1 == 0\&close2\_flag == 0\& \\
close1\_flag == 0\&\boldsymbol{t >= 5}\&z >= 0, \\
start \sim closed \sim \boldsymbol{part} \sim stop\& \\
0 < y2 <= 2\&x == 0\&y1 == 0\&close2\_flag == 1\& \\
close1\_flag == 0\&\boldsymbol{t >= 5}\&z + y2 >= 2\}.
\end{array}$$

VIII. Conclusions

In this paper, we have presented the models of Platform Screen Doors System (PSDS) which is a subsystem of subway control systems. Formal verification approaches and simulation based techniques have been applied. We proposed the methodology to integrate formal verification with industrial simulation. For the verification of PSDS, we combined the formal verification tool PHAVer with simulation tool Simulink/Stateflow. Benefited from the interactive simulation and powerful visualization features of Matlab toolset, we have simulated the sandwich situation by using Simulink blocks in the model, and explored the reaction of the system in the Scope window with respect to simulation time. To verify the correctness of the refinement, we checked the time interval property by PHAVer. In base phase, we focused on verification and simulation for four fundamental properties, and found the sandwich problem with simulation. Then in the exploration phase, we refined our controller's model and took simulation again, then modified the hybrid automaton of controller and checked the correctness by formal verification with PHAVer. As a result, we acquired the ultimate models of PSDS for our case. As future work, we plan to continue the verification on subway control systems.

Acknowledgments

This work is supported in part by National Basic Research Program of China (No. 2011CB302904), National High Technology Research and Development Program of China (No. 2011AA010101 and No. 2012AA011205), National Natural Science Foundation of China (No. 61061130541 and No. 61021004).

References

[1] R. Alur, C. Courcoubetis, T. Henzinger, and P. Ho, "Hybrid automata: An algorithmic approach to the specification and analysis of hybrid systems," in *Hybrid Systems*, ser. LNCS. Springer-Verlag, 1993, vol. 736, pp. 209–229.

[2] T. Henzinger, P. Kopke, A. Puri, and P. Varaiya, "What's decidable about hybrid automata?" *Journal of Computer and System Sciences*, vol. 57, no. 1, pp. 94–124, 1998.

[3] C. Le Guernic and A. Girard, "Reachability analysis of linear systems using support functions," *Nonlinear Analysis: Hybrid Systems*, vol. 4, no. 2, pp. 250–262, 2010.

[4] E. Asarin, O. Bournez, T. Dang, and O. Maler, "Approximate reachability analysis of piecewise-linear dynamical systems," in *Hybrid Systems: Computation and Control*, ser. LNCS. Springer-Verlag, 2000, vol. 1790, pp. 20–31.

[5] A. B. Kurzhanski and P. Varaiya, "Ellipsoidal techniques for reachability analysis," in *Hybrid Systems: Computation and Control*, ser. LNCS. Springer-Verlag, 2000, vol. 1790, pp. 202–214.

[6] A. Girard and C. Le Guernic, "Zonotope/hyperplane intersection for hybrid systems reachability analysis," in *Hybrid Systems: Computation and Control*, ser. LNCS. Springer-Verlag, 2008, vol. 4981, pp. 215–228.

[7] E. Asarin, T. Dang, O. Maler, and R. Testylier, "Using redundant constraints for refinement," in *Automated Technology for Verification and Analysis*, ser. LNCS. Springer-Verlag, 2010, vol. 6252, pp. 37–51.

[8] X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "An approach to the description and analysis of hybrid systems," in *Hybrid Systems*, ser. LNCS. Springer-Verlag, 1993, vol. 736, pp. 149–178.

[9] R. Alur, T. Henzinger, and P. Ho, "Automatic symbolic verification of embedded systems," *Software Engineering, IEEE Transactions on*, vol. 22, no. 3, pp. 181–201, 1996.

[10] T. Henzinger, P. Ho, and H. Wong-Toi, "Hytech: A model checker for hybrid systems," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 1, no. 1, pp. 110–122, 1997.

[11] G. Frehse, "PHAVer: Algorithmic verification of hybrid systems past HyTech," in *Hybrid Systems: Computation and Control*, ser. LNCS. Springer-Verlag, 2005, vol. 3414, pp. 258–273.

[12] R. Bagnara, E. Ricci, E. Zaffanella, and P. Hill, "Possibly not closed convex polyhedra and the Parma Polyhedra Library," in *Static Analysis*, ser. LNCS. Springer-Verlag, 2002, vol. 2477, pp. 299–315.

[13] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "SpaceEx: Scalable verification of hybrid systems," in *Computer Aided Verification*, ser. LNCS. Springer-Verlag, 2011, vol. 6806, pp. 379–395.

[14] C. Bonnett, *Practical Railway Engineering*. Imperial College Press, 2005.

[15] T. Granlund and K. Ryde, "The GNU Multiple Precision Arithmetic Library Version 4.0," 2001.