Model Checking Controllers with Predicate Inputs

Santhosh Prabhu M and Pallab Dasgupta
Department of CSE, Indian Institute of Technology, Kharagpur, India
{santhosh.prabhu, pallab}@cse.iitkgp.ernet.in

Abstract—Digital controllers sitting at the digital-analog boundary often react to specific analog events that can be modeled in terms of predicates over real variables. The specifications for such controllers are also naturally described in terms of similar events, and can be formally expressed with simple extensions of assertion languages. This paper studies the model checking problem for such controllers, where the inputs represent predicates over real variables. We show that this is a novel problem which is distinct from both model checking hybrid systems and model checking purely digital systems. This paper presents a methodology which enables us to solve this problem using a combination of SMT solvers and existing industrial model checking tools. We establish the theoretical correctness of the approach and present two case studies to demonstrate the proposed tool flow.

Index Terms—Formal Verification, Model Checking, Digital Controllers

I. INTRODUCTION

Digital control is being used in a wide variety of embedded control systems. Typically, a digital controller for an analog or hybrid system samples the real valued variables of the system and reacts by writing its control variables, which in turn serve as actuation commands. Such controllers are broadly divided into those that react continuously (that is, in every sample) and those that react discretely on specific events (such as threshold crossings). For example, a controller which switches on a pump when the water level in a tank falls below a specific lower threshold is a discrete controller. A controller which computes the rate of fuel injection depending on the speed of the vehicle is a continuous controller.

An important difference between discrete and continuous controllers is that the inputs to a discrete controller are Boolean (for example, whether the water level has crossed the threshold) and the output is a decision (for example, whether the pump should be switched on). On the other hand, the continuous controller has to perform arithmetic on the real variables (for example, computing the necessary fuel injection rate from the given speed).

Discrete controllers are quite common at the digital-analog boundary of integrated circuits. For example, most battery charger circuits have a digital brain which controls the switching between the charging modes based on voltage and current conditions in the analog driver circuit. Figure 1 shows a flow chart of a typical charger brain. It may be noted that the charger observes specific predicates over real variables such as $V_{BATT} > 3V$.

Discrete controllers are also used for supervisory control, that is, to orchestrate global control involving multiple continuous controllers. For example, an automatic gear transmission system, the supervisory control dictates the switching between the gear positions depending on specific thresholds the speed of the vehicle, whereas the rate of increase of speed is controlled by continuous control as a function of acceleration. In the water tank example, the supervisory control decides the switching on/off of the pump, where as continuous control decides the rate at which water is being pumped. In an elevator system supervisory control decides the sequence of floors to be visited, whereas continuous control operates the motors for moving between floors and for opening / closing doors.

Discrete controllers which react only to specific events over the real valued variables in the system are essentially digital finite state machines where some of the inputs are Predicates Over Real Variables (PORVs). The specification of such controllers is typically expressed in terms of PORVs as well. For example, the specification of the controller for the water tank may include the following requirements:
1) P1: If the water level is below 10, then the water pump must be ON in the next time instance.
2) P2: If the water level is above 75, the water pump must be OFF in the next time instance.

Since the water level is a real valued variable, the above properties cannot be expressed in propositional logics like Linear Temporal Logic (LTL) \([2]\). It is possible to express such properties by extending the notion of atomic propositions in LTL with predicates over real variables (PORVs). For example, the first of the two properties stated above can be expressed as:

\[ P1 : \quad G((\text{water level} < 10) \Rightarrow X\text{PumpON}) \]

In this property, \((\text{water level} < 10)\) is a PORV. PumpON is an atomic proposition indicating that the pump is ON.

The actual implementation of a controller may not be a verbatim translation of the specification. For example, the implementation of the water tank controller shown in Figure 2 switches the pump ON whenever the water level falls below 15, and switches it OFF whenever the water level rises above 70. This is to take into account possible aberrations in reading the water level and satisfy the specification robustly. Our objective is to formally verify that controller implementations (such as the one in Figure 2) are correct with respect to properties like the one stated above.

It is important to note that the controller is a purely digital finite state machine with the annotation that its inputs represent PORVs. The variant of LTL used for the specification extends the set of atomic propositions with PORVs. To the best of our knowledge the model checking problem formulated in this paper is novel and has not been studied in existing literature. The primary contribution of this paper is in studying this new problem and proposing a model checking algorithm for the problem. We present two case studies from two different domains and demonstrate the tool flow implementing our proposal.

The paper is organized as follows. Section II presents the background and formal statement of the problem. The verification methodology is presented in section III. Section IV describes case studies and experimental results. All proofs are presented in the appendix.

A controller for a hybrid system can be defined as a tuple:

\[ G = (Q, I, \delta, Q_0, AP, var, L) \]

where:

- \( Q \) is the set of states of the controller,
- \( Q_0 \subseteq Q \) is the set of initial states,
- \( AP \) is a set of atomic propositions (labels),
- \( var \) is a set of real valued variables,
- \( I = I_B \cup I_{PORV} \) is the set of inputs of the controller, where \( I_B \) is a set of Boolean signals and \( I_{PORV} \) is a set of PORVs over \( var \),
- \( \delta \subseteq Q \times 2^I \times Q \) is the transition relation,
- \( L : Q \rightarrow 2^{AP} \) is a function for labeling the states in \( Q \) with propositions in \( AP \).

As an example, the constituents of the tuple corresponding to the controller in Figure 2 are as follows. The set of states is \( Q = \{S_0, S_1\} \), with the initial state \( Q_0 = \{S_0\} \). The set of atomic propositions is \( AP = \{\text{PumpON}\} \), and the set of real variables is \( var = \{\text{water level}\} \). There are two PORVs, that is, \( I_{PORV} = \{\text{water level} < 15, \text{water level} > 70\} \). Since there are no other inputs, \( I_B = \emptyset \). The transition relation, \( \delta \), is as illustrated in Figure 2. The labeling function defines \( L(S_0) = \emptyset \) and \( L(S_1) = \{\text{PumpON}\} \).

The semantics of propositional temporal logics like LTL are defined over closed systems called Kripke structures (that is, there is no distinction between inputs and non-inputs). The conversion of an open system (Moore machine) to a non-deterministic Kripke structure is a standard procedure. We extend this procedure to obtain a Kripke structure \( M \) equivalent to the controller:

\[ M = (Q', \delta', Q'_0, AP', L') \]

where:

- \( Q' = Q \times 2^I \). We denote a state \( q'_i \in Q' \) as a pair \( (q_i, a_i) \), where \( q_i \in Q \) and \( a_i \in 2^I \),
- \( Q'_0 = Q_0 \times 2^I \),
- \( \delta' \subseteq Q' \times Q' \) is the transition relation, such that \( (q'_i, q'_j) \in \delta' \) iff \( (q_i, a_i, q_j) \in \delta \),
- \( AP' = AP \cup I \),
- \( L' : Q' \rightarrow 2^{AP} \) is a labeling function, such that \( L'(q'_i) = L(q_i) \cup a_i \)
  where \( q'_i = (q_i, a_i) \).

Intuitively, the input variables of the controller are also treated as state variables in the Kripke structure. If the controller has a transition from a state \( s \) to a state \( s' \) on input \( a \), then in the Kripke structure there is a transition from the state \( \langle s, a \rangle \) to every state of the form \( \langle s', \beta \rangle \), where \( \beta \) is a valuation of the input variables. Since the input variables represent PORVs in our case, the Kripke structure obtained through this transformation has PORVs in its states.

A path, \( \pi \) in the Kripke structure is defined in the standard way. \( \pi = q_0, q_1, \ldots \) is an infinite sequence of states, where

\[ \forall i, q'_i \in Q', q'_0 \in Q'_0, \text{ and } \forall i, (q'_i, q'_{i+1}) \in \delta' \].

Let \( \Sigma = I_B \cup var \cup AP \) denote the set of variables of the controller. \( I_B \cup var \) contains the input variables and \( AP \) represents the set of outputs asserted by the controller.
We define a signal trace or simply trace, $\sigma$, as an infinite sequence $A_0, A_1, \ldots$, where each $A_i \in 2^{I_B} \times \mathbb{R}^{|\text{var}|} \times 2^{A_P}$. In other words, $\sigma$ is an infinite sequence of valuations of the variables in $\Sigma$. Note that the valuations of the variables in $\text{var}$ are real valued, while the rest are Boolean.

A trace $\sigma = A_0, A_1, \ldots$ simulates a path $\pi = q'_0, q'_1, \ldots$ of the Kripke structure iff for every $k$:

1) Set of propositions in $L'(q'_k)$ is exactly equal to the subset of $A_P$ that is true in $A_k$, and
2) the valuation of the variables, $\text{var}$, in $A_k$ satisfies each PORV that labels $q'_k$.

For expressing properties in terms of PORVs, we propose an extension of LTL, which defines formulae over propositions as well as PORVs. Given $\Sigma = I_B \cup \text{var} \cup A_P$, the set of variables, well-formed formulae (wff) over $\Sigma$ in extended LTL may be:

- $p \in I_B \cup A_P$, $\top$, $\bot$
- $q$, a PORV over $\text{var}$
- $\varphi \land \psi$, $\neg \varphi$, $X \varphi$, $\varphi U \psi$ where $\varphi$ and $\psi$ are wff

Extensions of propositional temporal logics with PORVs have been studied in the past. For example, the logic STL proposed in [3] is a dense real time logic, which allows PORVs. Extensions of the industry standard assertions languages PSL and SVA with PORVs have also been studied [4], [5].

The signal trace $\sigma = A_0, A_1, \ldots$ satisfies a formula $\varphi$ ($\sigma \models \varphi$), under the following conditions:

- $\sigma \models \top$
- $\sigma \not\models \bot$
- $\sigma \models p$, where $p$ is a proposition or PORV iff $p$ is made true by $A_0$
- $\sigma \models \varphi \land \psi$ iff $\sigma \models \varphi$ and $\sigma \models \psi$.
- $\sigma \models \neg \varphi$ iff $\sigma \not\models \varphi$.
- $\sigma \models X \varphi$ iff $\exists i \in \mathbb{N}_0, \sigma_i \models \varphi$ and $\forall j < i, \sigma_j \models \varphi$.

Here, $\sigma_i$ is defined as the suffix $A_i, A_{i+1}, \ldots$ of $\sigma$.

In addition, the Boolean connectives $\lor$, $\Rightarrow$, $\Leftrightarrow$ and the derived temporal operators $F$, $G$ and $V$ are also used.

A Kripke structure $M$ is said to satisfy a property $\varphi$, written as $M \models \varphi$, iff there exists no trace $\sigma$ such that $\sigma \not\models \varphi$ and $\sigma$ simulates a path in $M$. The next section presents the proposed decision procedure for this problem.

III. THE VERIFICATION METHODOLOGY

Given a PORV-labeled Kripke structure $M$ and a property $\varphi$, our model checking task is a search for a trace $\sigma$ such that $\sigma$ simulates a path in $M$ and $\sigma \not\models \varphi$. Due to the density of the real valued variables over which the PORVs are defined, there can be infinite number of traces that simulate each path of $M$. Also, due to non-determinism, a trace may simulate multiple paths in $M$.

Let $\text{Traces}(\varphi)$ denote the set of traces satisfying $\varphi$, and let $\text{Traces}(M)$ denote the set of traces that simulate some path in $M$. Our objective is to determine whether $\text{Traces}(M) \cap \text{Traces}(\neg \varphi) = \emptyset$.

Let $\text{True}(\varphi, \sigma, i)$ denote the set of propositions and PORVs of $\varphi$ that are made true by $A_i$ on the trace $\sigma = A_0, A_1, \ldots$. Let $\text{True}(\pi, i)$ denote the set of propositions and PORVs labeling state $q'_i$ in the path $\pi = q'_0, q'_1, \ldots$ of the Kripke structure. We define $\text{True}(\varphi, \sigma, i)$ and $\text{True}(\pi, i)$ to be consistent iff:

1) the set of atomic propositions of $\varphi$ in $\text{True}(\varphi, \sigma, i)$ and the set of atomic propositions of $\varphi$ in $\text{True}(\pi, i)$ are exactly the same, and
2) the conjunction of PORVs in $\text{True}(\varphi, \sigma, i)$ is satisfiable with the conjunction of PORVs in $\text{True}(\pi, i)$, that is, there is some valuation of $\text{var}$ that satisfies all PORVs in $\text{True}(\varphi, \sigma, i)$ and all PORVs in $\text{True}(\pi, i)$.

Given an extended LTL formula $\varphi$, we denote by $A_\varphi$ the set of PORVs and propositions that appear in it. Consider an infinite sequence $\nu = \nu_0, \nu_1, \ldots$ of truth assignments to the elements of $A_\varphi$. $\nu$ is said to be consistent with a path $\pi = q'_0, q'_1, \ldots$ of a Kripke structure $M$, iff $\nu_i$ is consistent with $\text{True}(\pi, i)$, for all $i \geq 0$. Also, since $\varphi$ is just like any LTL formula, its truth can be interpreted over infinite sequences like $\nu$. This interpretation is the same as the one for LTL formulæ. It may be noted that if a trace $\sigma \models \varphi$, and $S$ is the sequence of truth assignments that $\sigma$ gives to the PORVs and propositions in $\varphi$, then $S\models \varphi$. Also, if $S$ is a sequence of truth assignments (to the PORVs and propositions) that satisfies $\varphi$, and $\sigma$ is any trace that defines the same truth assignments as $S$, then $\sigma \models \varphi$.

Lemma 3.1: For a given PORV-labeled Kripke structure $M$ and a property $\varphi$, $M \not\models \varphi$ if and only if there exists a sequence, $\nu = \nu_0, \nu_1, \ldots$ over $A_\varphi$ such that $\nu \models \neg \varphi$ and $\nu$ is consistent with some path $\pi$ in $M$.

Lemma 3.1 forms the basis of the proposed verification method. Each path in a PORV-labeled Kripke structure $M = (Q', \delta', Q'_0, A_P', L')$, represents a sequence over $A_P'$, but since $A_P' \neq A_\varphi$, we cannot interpret the truth of $\varphi$ over this sequence directly. However, we can consider all possible sequences over $A_\varphi$ that are consistent with paths in $M$ and determine (using standard interpretation of LTL) whether any of them (say $\pi$) refutes $\varphi$. If so, then by Lemma 3.1, $M \not\models \varphi$, and the path $\pi$ is simulated by some trace that refutes $\varphi$.

Therefore, for a given PORV-labeled Kripke structure, $M = (Q', \delta', Q'_0, A_P', L')$, and a property $\varphi$, we develop a Kripke structure, $M' = (Q'', \delta'', Q''_0, A_P'', L'')$ where states are labeled with the PORVs of $\varphi$ as well. Intuitively, each state of $M$ is split into $2^k$ states in $M'$ where $k$ is the number of PORVs in $\varphi$ that are not in present in $M$. Each of the split states are labeled with a distinct combination of PORVs imported from $\varphi$.

Formally,

- $Q'' = Q' \times 2^{C_\varphi \setminus C_M}$ where $C_\varphi$ and $C_M$ are the sets of PORVs in $\varphi$ and $M$ respectively.
- $\delta'' = \{(s, a, (s', b))|(s, s') \in \delta', a, b \in 2^{C_\varphi \setminus C_M}\}$
- $Q''_0 = Q'_0 \times 2^{C_\varphi \setminus C_M}$
- $L'': Q'' \rightarrow 2^{C_\varphi \cup C_M}$, is defined as $L''((s, a)) = L'(s) \cup a$.

The states of the Kripke structure $M'$ are labeled with the PORVs of $M$ as well as $\varphi$, hence we can interpret the truth
of $\varphi$ over the paths of $M'$ using the standard semantics of LTL. This enables us to use a standard LTL model checker to check $\varphi$ on $M'$. However this check is not equivalent to checking whether $M \models \varphi$ for the reason explained below.

A state, $s$, of $M'$ may have inconsistent labels, that is, the PORVs labeling $s$ (consisting of the PORVs from $M$ and $\varphi$) may not be satisfiable together. Such states are not really possible, and therefore any counter-example trace that passes through such states is fictitious. In order to prevent the model checker from coming up with such fictitious counter-example paths, we perform model checking with additional constraints, as follows:

1) We take the union, $C = C_\varphi \cup C_M$ and use an SMT solver to compute the set of minimal unsatisfiable cores of $C$.
2) For each unsatisfiable core we add a constraint to express that no state in the path should contain the elements of this core in its labels. This can be expressed in LTL as the property:

$$G(\neg \hat{\psi})$$

where $\hat{\psi}$ is the disjunction of the UNSAT cores, with the PORVs replaced by propositions.
3) Finally, we use a model checker to determine whether $M' \models \varphi$, treating all PORVs as propositions, under the assumption of $G(\neg \hat{\psi})$.

The steps of the proposed model checking methodology is shown in Figure 3.

We will demonstrate this method using the controller $G$ of Figure 2 and the property $G((\text{water\_level} < 10) \rightarrow X\text{Pump\_ON})$. We first represent the controller as a Kripke Structure $M$. This structure does not have the PORV $\text{water\_level} < 10$. Therefore, in $M'$, we have two copies of each state in $M$, one which is labeled with $(\text{water\_level} < 10)$ and one which is not. For every edge from state $q$ to state $q'$ in $M$, we have in $M'$ edges from both copies of $q$ to both copies of $q'$. Using an SMT solver, we compute the set of UNSAT cores of the set $\{(\text{water\_level} < 10), (\text{water\_level} < 15), (\text{water\_level} > 70)\}$. We denote by $\psi$ the disjunction of these cores.

$$\psi = ((\text{water\_level} < 10) \land \neg (\text{water\_level} < 15)) \lor ((\text{water\_level} < 15) \land (\text{water\_level} > 70)) \lor ((\text{water\_level} < 10) \land (\text{water\_level} > 70))$$

Once $\psi$ is computed, we perform the following replacement on $M'$, $\varphi$ and $\psi$:

- $(\text{water\_level} < 10) \equiv w_{lt\_10}$
- $(\text{water\_level} > 15) \equiv w_{gt\_15}$
- $(\text{water\_level} > 70) \equiv w_{gt\_70}$

to obtain $M'$, $\hat{\varphi}$ and $\hat{\psi}$ respectively. Now, using an industrial model checker, we verify $\hat{\varphi}$ over $M'$, under the assume constraint $G\neg \hat{\psi}$.

Lemma 3.2: A trace $\sigma$ simulates a path in $M'$ iff it simulates a path in $M$ (where $M'$ is defined using $M$, as shown in Figure 3).

While verifying $\hat{\varphi}$ over $M$, a model checker reports as a counter-example any path in $M$ that refutes $\hat{\varphi}$ according to LTL semantics. A counter-example so obtained is actually a path in $M$, that refutes $\varphi$. We state the theorem establishing the correctness of the proposed technique, as follows:

Theorem 3.1: Model checking $\hat{\varphi}$ over $M'$ under the constraint $G\neg \hat{\psi}$ will produce a counterexample if and only if $M$ refutes $\varphi$, and any trace that takes $M'$ through the path corresponding to the counterexample will be simulated by $M$, and will violate $\varphi$, and hence will be a valid counterexample for $\varphi$ in $M$.

IV. CASE STUDIES AND RESULTS

We have applied the approach presented in this paper for verifying two digital controllers, both of which control analog plants.

A. Steam boiler controller

This controller is based on the one described in [6]. It controls two pumps that feed water into the boiler, and also a valve that releases water from the boiler. The controller has 7 states, as shown in Figure 4. Among these, the states BOTH_ON, ONE_ON and BOTH_OFF are the normal operating modes. The transitions between states are labeled with PORVs. We have verified this controller against several properties. The properties that were reported to hold are:

- If a pump is on currently, and water level remains above 27 for 3 consecutive time instants, in the state after that, both pumps will be off.
- If pump $p_1$ is on currently, and water level remains above 77 for 3 consecutive time instants, in the state after that, both pumps will be off.
- If the steam boiler is in operation but not in a normal operating mode, and is waiting, the steam rate is zero and the water level is between 16 and 19 for 3 time instants, normal operation is entered.
\[ \varphi_3 = G((n_{op}) \land \neg w > 10, \land t > 90) \Rightarrow (XF_2(p_{on} \lor \text{stopped})) \]

(We use \( F \psi \) to specify that \( \psi \) holds within \( t \) time steps, including the current one.)

Counter-examples are obtained in the following cases:

- If the boiler is not in \( \text{STOP} \), and the water level falls below 5, the boiler operates within 2 steps.

\[ \varphi_4 = G(\neg \text{stopped}) \land (w < 5) \Rightarrow XF_2(\text{stopped}) \]

- If both pumps are on, and the water level stays above 25 for 3 time instants, at least one pump is turned off.

(Counter-example obtained if pumps are not turned off in \( \text{STOP} \) mode).

\[ \varphi_5 = G((p_{on} \land p_{on}) \land G_3(w \geq 25)) \Rightarrow X_3 \neg p_{on} \]

**B. Automatic Transmission Control**

This controller (Figure 5), based on an embedded controller implemented in Simulink/Stateflow, controls the gear positions of a vehicle. There are four different gear positions, and the transitions between positions are labeled with PORVs. The controller computes the upper and lower thresholds of speed for the current gear position and throttle value, compares them with the current speed, and the gear is shifted up or down as necessary. The functions used for computing the thresholds are linear, and hence, we can represent the result of the comparisons as linear predicates over the variables speed and throttle. The properties that hold include:

- If the controller is in one of the normal operating modes and the water level falls below 7, within 2 transitions, the controller goes to \( \text{STOP} \), or turns both pumps on.

\[ \varphi_3 = G((n_{op}) \land w \leq 7) \Rightarrow (XF_2(p_{on} \lor \text{stopped})) \]

(Counter-example obtained if pumps are not turned off in \( \text{STOP} \) mode).

- If both pumps are on, and the water level stays above 5, the boiler stops within 2 steps.

\[ \varphi_6 = G((n_{op}) \land w \leq 5) \Rightarrow X_2 \neg p_{on} \]

The following properties are falsified:

- If the speed is in \( [35,50] \) and throttle is in \( [40,50] \), the gear doesn’t change.

\[ \varphi_8 = G((35 < s \leq 50) \land (40 < t \leq 50) \Rightarrow ((g1 \Rightarrow (})

Fig. 4: Controller for steam boiler pumps

Fig. 5: Automatic Transmission Controller
<table>
<thead>
<tr>
<th>Controller</th>
<th>Property</th>
<th>Count</th>
<th>Time</th>
<th>Property</th>
<th>Count</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boiler Controller</td>
<td>$\varphi_1$</td>
<td>7 + 2</td>
<td>25</td>
<td>$\varphi_3$</td>
<td>7 + 2</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>$\varphi_2$</td>
<td>2</td>
<td>56</td>
<td></td>
<td>$\varphi_4$</td>
<td>2</td>
</tr>
<tr>
<td>Transmission Control</td>
<td>$\varphi_5$</td>
<td>2</td>
<td>56</td>
<td></td>
<td>$\varphi_6$</td>
<td>2</td>
</tr>
</tbody>
</table>

Tab. 1: Results

$$X(g_1) \land (g_2 \Rightarrow X(g_2)) \land (g_3 \Rightarrow X(g_3)) \land (g_4 \Rightarrow X(g_4))$$

- If the speed is greater than or equal 40 and also twice the throttle value for 5 consecutive time instants, then the gear will be at position 4.

$$\varphi_0 = G(G_0(s \geq 40 \land s \geq 2t) \Rightarrow X_5(g_4))$$

We developed a toolkit implementing the methodology illustrated in Fig 3. We use Mall[7], a tool that uses Ip_solve package in the back-end, for computing the unsatisfiable cores and the Synopsys Magellan tool [8] as the model checker. Table [1] present the results obtained for the two case studies discussed in this paper. These results were taken on a 2.0GHz 8-core Intel Xeon with 64GB RAM.

REFERENCES


APPENDIX

Proof of Lemma 3.2

Part 1: Consider a trace $\sigma = \sigma_1, \sigma_2, \ldots$ that simulates some path $\pi_i$ in $M$. Let $s_i$ and $s_{i+1}$ be any two successive states in $\pi_i$. $\sigma_1$ and $\sigma_{i+1}$ make true only those propositions, and at least those PORVs, that are present in $\pi_i$, $\sigma_1$ and $\sigma_{i+1}$ shall make true the labels of $q_i$, $s_1 = (s_i, X)$ and $s_{i+1} = (s_{i+1}, Y)$ are present in $Q_i$. Also, by definition of $\delta^*$, since $(s_i, s_{i+1})$ is in $\delta^*$, $(s_i, s_{i+1})$ is in $\delta^{*\pi_i}$, and by definition of $Q_i$, $s_i$ is in $Q_i$. So, since states $s_i$ constitute a path in $M$. Since the elements of $L(\sigma_1)$ and $X$ are exactly the propositions and PORVs made true by $\sigma_1$, $\sigma_1$ simulates a trace $\pi$. Let $\sigma$ be a trace that simulates a path $\pi_1$ in $M$. Consider any two states $p, a$ and $q, b$ that occur in $\pi$ at positions $i$ and $i+1$ respectively. From the definition of $Q^*$, we know that there are states $a$ and $q$ in $Q^*$, whose labels define the same truth assignment as $(a, b)$ respectively, for the propositions and the PORVs present in $M$. So, $\sigma_i$ and $\sigma_{i+1}$ shall make true the labels of $p$ and $q$ in $M$. Also, since $(p, a), (q, b)$ is in $\delta^*$, $(p, q)$ is in $\delta^{*\pi_i}$. Since $(a, b) \in Q^*$, $(q, b) \in Q^*$, and $(q, a) \in Q^*$, there is a path in $M$, simulated by $\sigma$.

Proof of Theorem 3.1

Part 1: The labels of any counterexample path in $M'$ would define a sequence $\nu$ over $A_\varphi$, such that $\nu \not\models \varphi$. Due to the constraint $G^-\psi$, for any path that is found, the corresponding path $\pi$ in $M'$ cannot contain inconsistently labeled states. For any state $s_i$ in $\pi$, let $\eta_i$ denote a value assignment to the elements of $var$ such that, among the elements of $A_\varphi^\pi$, only the labels of $s_i$ are true made. We define a trace $\sigma = \eta_1, \eta_2, \ldots$. Since $\sigma$ simulates $\pi$ in $M'$, it simulates a path in $M$ too(by Lemma 3.2). Since $\eta$ is the sequence defined by $\sigma$ over $A_\varphi$, and $\sigma$ simulates $\pi$ and $\pi$ and $\nu$ are consistent. And since $\nu \not\models \varphi$, by Lemma 3.1 $M \not\models \varphi$, and $\sigma$ is the counterexample trace that simulates a path in $M$ and refutes $\varphi$.

Part 2: Assume that $M'$ refutes $\varphi$, and $\sigma$ is the counterexample trace. By Lemma 3.2 $\sigma$ simulates a path (say $\pi'$) in $M'$. Since $\pi$ is simulated by a trace, it cannot contain any inconsistent states. So, the constraint $G^-\psi$ is satisfied by the path $\pi'$ in $M'$ corresponding to $\pi$. Also, since $\sigma \not\models \varphi$, the sequence $\nu$ that $\sigma$ defines over $A_\varphi$ shall also refute $\psi$. Thus, $\pi$ shall refute $\varphi$, and $\pi'$ shall refute $\hat{\varphi}$, according to LTL semantics. Hence, the model checker will report the path $\pi'$, which corresponds to $\pi$, the path simulated by $\sigma$ in $M'$. □