Formal Verification of Statecharts Using Finite-State Model Checkers

Qianchuan Zhao and Bruce H. Krogh, *Fellow, IEEE*

***Abstract*—This brief concerns the formal verification of properties of statecharts, a hierarchical state machine formalism for designing control-system logic. Various semantics have been defined for statecharts in terms of the microsteps that determine the transitions between statechart configurations. We show how computation tree logic (CTL) specifications for a statechart can be verified using a finite-state model checker such as SMV. The inputs to the model checker are a model and an associated expansion of the CTL formula that reflect the semantics. A Kripke structure with marked states provides the formal relationship between the expanded model and the original statechart structure and CTL specification. The results apply to a general class of semantics and statecharts with bounded behaviors. The approach is illustrated with a small example.**

***Index Terms*—Control logic, discrete control, formal verification, statecharts.**

I. Introduction

THIS brief presents a method for using existing model-checking tools to verify properties of statecharts [10], one of the most popular representations for control logic design. Model checkers, which implement algorithms to perform formal verification of properties of finite-state systems, have been effective in finding design errors for digital circuits and protocols [2], [17]. The growing use of tools for computer-aided control system design increases the possibilities for using these tools to verify the correctness of control logic before it is realized in software and implemented on a target processor.

Statecharts are graphical representations of discrete-state transition systems based on hierarchical state machines. The state of a statechart structure is commonly referred to as the configuration and a state transition is referred to as a step. Steps occur in response to input events. The statechart step-semantics defines the admissible steps, usually in terms of microsteps that define the sequences of intermediate configurations and variable values leading from one statechart state to the next. Various step-semantics have been introduced to precisely define the admissible steps for a given statechart structure [1], [14], [18]. Multiple semantic interpretations are possible for statecharts because of the possibility of having multiple states enabled at one time in the chart, and because changes in variables and configurations can introduce new enabled transitions.

To use a model checker to verify properties of a statechart, it is necessary to build a finite-state model of the statechart transition system [12], [13], [21]. Fig. 1 illustrates the proposed approach to verifying properties of statecharts using an existing model checker. In this figure, the shaded constructs are conceptual. The upper section of the figure illustrates the verification one wants to perform. A specification in computation tree logic (CTL) for the statechart describes desired behaviors in terms of statechart steps and states. If one were available, the statechart and the statechart CTL specification would be the input to a statechart model checker. Such a statechart model checker would analyze the possible sequences of statechart steps represented (conceptually) by an associated state transition system called a statechart Kripke structure [7]. To take advantage of existing model checking tools, the statechart step-semantics is used to construct a microstep model for the statechart. This microstep model is constructed in terms of the input language for the model checker. For example, for the model checker SMV [3], [4], [19], the microstep model would be a collection of concurrent state machines described using the SMV input language. The statechart CTL expression is expanded into a CTL expression for this microstep model based on the statechart step-semantics. The model checker then analyzes the possible sequences of statechart microsteps represented (conceptually) by an associated microstep Kripke structure. In this brief, we provide rules for constructing the expanded microstep CTL specifications and demonstrate the equivalence of the results obtained by the microstep model verification and the desired results that would be obtained by a statechart model checker.

Verification of statecharts has been studied by a number of researchers [9], [20], [6]. The idea of formula expansion and structure shrinking developed in this brief is similar to the approach used for analysis of the StateMate variant of statecharts based on the step-semantics defined in [8]. Recently, Bienmüller studied the efficiency of model checking StateMate statecharts using alternative approaches to microstep models [5]. The objective in this brief is to define an approach to statechart verification that is applicable for a general class of step-semantics.

This brief is organized as follows. Section II presents the definitions and notation for statecharts, and defines step-semantics for statecharts. Section III introduces Kripke structures as transition system semantics for formal verification of statechart properties. Section IV describes the verification of statechart CTL specifications via specification expansion. An example in Section V, illustrates the verification procedure and shows that specification expansion is necessary. The concluding section summarizes the contributions of this brief.

Manuscript received May 9, 2003. Manuscript received in final form March 14, 2006. Recommended by Associate Editor S. Kowalewski. This work was supported in part by the National Science Foundation of China under Grant 60274011 and Grant 60574067, by Fundamental Research Funds from Tsinghua University and the Chinese Scholarship Council, by Ford Motor Company, by General Electric Transportation Systems, and by the Pennsylvania Infrastructure Technology Alliance, a partnership of Carnegie Mellon, Lehigh University, and the Commonwealth of Pennsylvania's Department of Economic and Community Development.

Q. Zhao was with the Department of Electronic and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213 USA. He is now with the Center for Intelligent and Networked Systems, Department of Automation, Tsinghua University, Beijing 100084, China (e-mail: zhaoqc@tsinghua.edu.cn).

B. H. Krogh is with the Department of Electronic and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: krogh@ece.cmu.edu).

Digital Object Identifier 10.1109/TCST.2006.876921

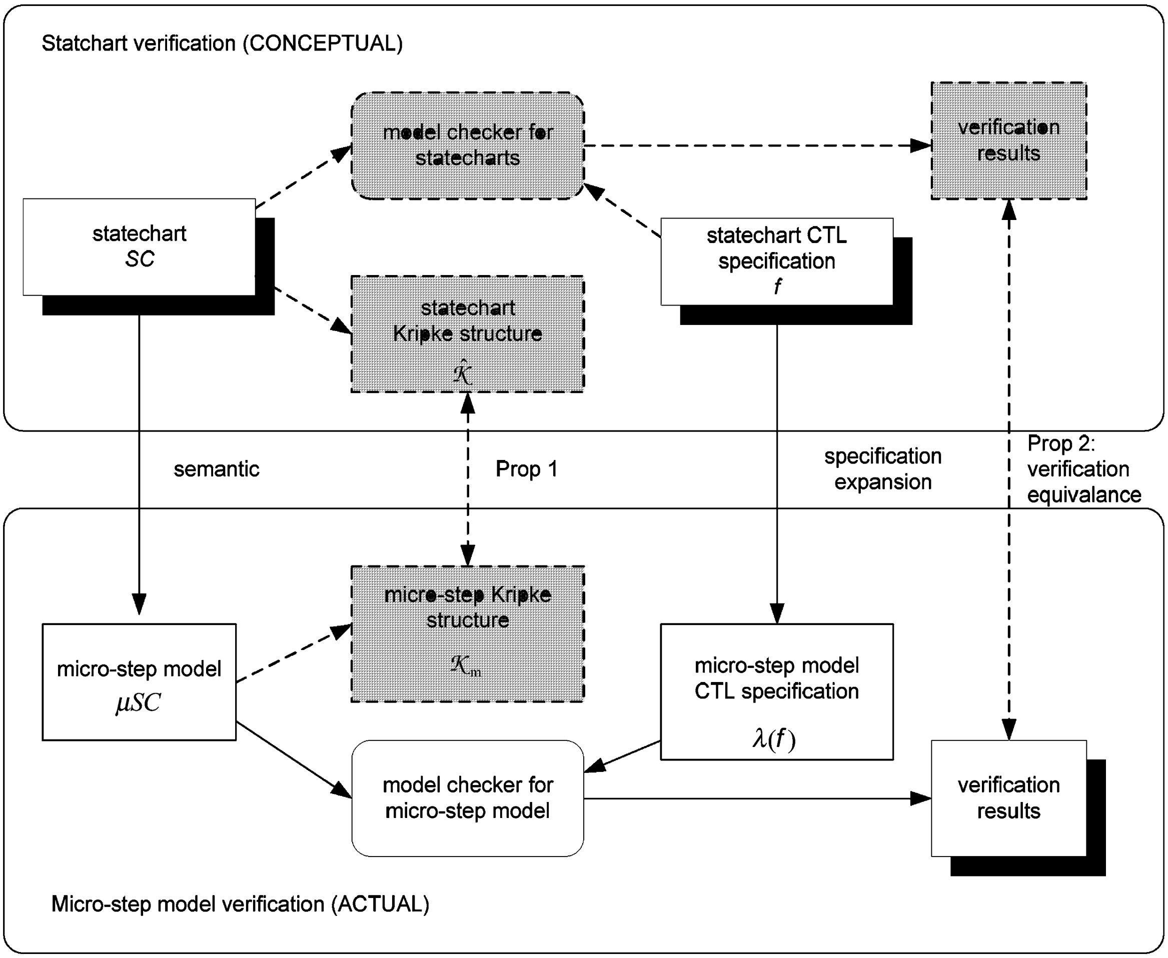
Statchart verification (CONCEPTUAL)
model checker for statecharts
verification results
statechart SC
statechart Kripke structure $\hat{\mathcal{K}}$
statechart CTL specification f
semantic
Prop 1
specification expansion
Prop 2: verification equivalance
micro-step Kripke structure $\mathcal{K}_m$
micro-step model μSC
micro-step model CTL specification λ(f)
model checker for micro-step model
verification results
Micro-step model verification (ACTUAL)

Fig. 1. Overview of proposed approach for formal verification of statecharts using a finite-state model checkers.

II. Statecharts

This section describes statechart structures and statechart step-semantics. We follow the standard definitions of [18] augmented with Boolean variables to capture important features of most practical implementations of statecharts [16], [11].

A statechart structure is a tuple $\mathcal{S} = \langle Q, \rho, r, \delta, T \rangle$ specifying the topological aspects of the statechart graph. Q is the set of chart-states and T is the set of chart-transitions. The children function ρ specifies a tree-like hierarchical relation on Q with a special chart-state $r \in Q$ as the root of the tree. The root state r is an ancestor of all other chart-states. The function δ specifies a default chart-state for each OR-state whose meaning is made clear below.

A statechart is a tuple $\mathrm{SC} = \langle \mathcal{S}, \mathcal{V}, \mathcal{E}, \chi, B, A \rangle$, where: $\mathcal{S}$ is a statechart structure (defined above); $\mathcal{V} = v^1, \ldots, v^m$ is the set of m Boolean variables;[1] $\mathcal{E}$ is a set of events partitioned into $\mathcal{E}_I$, the set of input events, and $\mathcal{E}_B$, the set of broadcasted events; $\chi : T \rightarrow P_{\mathcal{E} \times \mathcal{V}}$ defines the enabling conditions for the chart-transitions, where $P_{\mathcal{E} \times \mathcal{V}}$ is the set of predicates on the sets of events and Boolean variables; $B : T \rightarrow 2^{\mathcal{E}_B}$ associates broadcast events with chart-transitions; and $A : T \times \mathcal{B}^{|\mathcal{V}|} \rightarrow \mathcal{B}^{|\mathcal{V}|}$ associates actions with chart-transitions, where $\mathcal{B} = \{0, 1\}$.

In the graphical presentation of statecharts, chart-states are boxes and chart-transitions are directed arcs, as illustrated in Fig. 2. If a chart-state s is immediately contained in a chart-state s' (no other chart-state in between), s is called a child of s'. In this case, $s \in \rho(s')$ and s' is called a superstate. In Fig. 2, r and P are examples of superstates, with $P \in \rho(r)$ and $P0 \in \rho(P)$. Chart states that do not contain other chart-states are called basic chart-states. Superstates are either as AND-states or OR-states, which represent concurrent or sequential components of the statechart, respectively. The children of AND-states, shown as dotted boxes, are active simultaneously. The children of OR-states and basic chart-states are shown as solid boxes. In Fig. 2, r is an AND-state, P and Q are OR-states. Default chart-states of OR-states are indicated by arrows with open tails.

The state of a statechart structure is determined by a set of active chart-states, called the configuration of the statechart and a valuation of the statechart variables. A statechart responds to a set of input events by firing a sequence of chart-transitions and the system configuration and variables change accordingly. A complete transition in the statechart state in response to an input event is called a step.

[1] Valuations of all variables will be denoted by vectors $v \in \{0, 1\}^m$, and the ith component of a valuation vector v will be denoted by $v(i)$, which is a valuation of variable v^i.

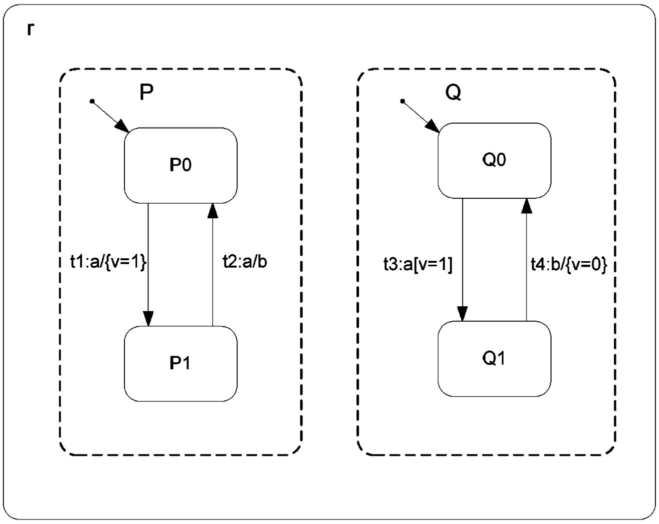

Fig. 2. Statechart with variables.

Configurations are defined as maximally consistent sets of chart-states, which are sets of chart-states satisfying the following conditions: the root state is included; each AND-state included in the configuration also includes all of its children; and each OR-state included in the configuration also includes exactly one of its children. The set of valid configurations is denoted by $\mathcal{C}$. For the statechart in Fig. 2, the set of configurations is $\mathcal{C} = \{C_i, i = 1,2,3,4\}$, where $C_1 = \{r,P,Q,P0,Q0\}, C_2 = \{r,P,Q,P0,Q1\}, C_3 = \{r,P,Q,P1,Q0\}$, and $C_4 = \{r,P,Q,P1,Q1\}$.

The logical behaviors of a statechart are specified by associating conditions and actions involving events and variables with the chart-states and chart-transitions. For each chart-transition $t \in T$, the associated enabling condition is specified by a Bachuss normal form (BNF) expression [15] of the form

$$\chi(t)[''/''B(t)][''\{''A(t,v)''\}'']$$

where optional items are enclosed by [and], and v is a valuation of the Boolean variables. Only those events and variable values, such that the condition is true, enable the chart-transition. In Fig. 2, the set of events is $\mathcal{E} = \{a,b\}$, there is one variable $\mathcal{V} = \{v^1\}$, and the condition of chart-transition $t1$ is $\chi(t1) = \{a\}$. The action of $t1$ on the variable is $A(t1,v) = \{v' := 1\}$, where v denotes the valuation of the variable before the action and v' denotes the valuation after the action. In general, the actions on variables can be a function of their current valuations.

When input events occur, the chart-state and outputs are updated instantaneously with respect to the clock of the environment. A chart-transition is said to be enabled if 1) its source chart-state is active (which means the chart-state is contained in the current configuration) and 2) the predicate of the chart-transition is true for the current values of Boolean variables and the active events (including input events and events broadcasted so far from the microsteps given by the step-semantics for the statechart, as defined below). Only enabled chart-transitions can be fired. The set of enabled chart-transitions for a given configuration $C \in \mathcal{C}$, values of variables $v \in \mathcal{B}^{|\mathcal{V}|}$, and set of input events $\Sigma \subset \mathcal{E}$ is denoted by $\text{enabled}(C,v,\Sigma)$. In Fig. 2, $\text{enabled}(C_1,0,\{a\}) = \{t1\}, \text{enabled}(C_3,1,\{a\}) = \{t3,t2\}, \text{enabled}(C_4,1,\{a\}) = \{t2\}$, and $\text{enabled}(C_2,1,\{a,b\}) = \{t4,t1\}$.

The firing of a chart-transition t changes the configuration C to a new configuration C' which is determined by a configuration transition function $\Omega(C,t)$. The basic idea of obtaining C' from C is to deactivate the source chart-states affected by the chart-transitions and to activate their destination chart-states. In Fig. 2, $\Omega(C_1,t1) = C_3, \Omega(C_3,t3) = C_4, \Omega(C_4,t2) = C_2$, and $\Omega(C_2,t4) = C_1$.

When a set of enabled chart-transitions is in conflict, that is, when the firing of one chart-transition in the set will disable other chart-transitions in the set, there can be different ways to define the next configuration. The rules for determining the firing sequences for sets of enabled chart-transitions is known as defining step-semantics of statecharts. Let T^* be the set of chart-transition sequences with finite length (including the "empty sequence" ε containing no transition).[2] All definitions for step-semantics of statecharts satisfy the following basic properties. To produce step-semantics that have specific properties, such as global consistency or compositionality, more restrictions are needed (see [18], [21], and [14] for details).

Definition 1—Step-Semantics of Statecharts: A function $\alpha : \mathcal{C} \times \mathcal{B}^{|\mathcal{V}|} \times 2^{\mathcal{E}_I} \to T^*$ is a step-semantics of SC if α satisfies:

1) $\alpha(C,v,\Sigma) = \varepsilon$ (the empty string); or
2) $\alpha(C,v,\Sigma) = t_1 t_2 \ldots t_k$, where $t_1, t_2, \ldots, t_k \in T$, such that there exists a sequence of configurations $C_0, C_1, \ldots, C_k$, a sequence of values of variables $v_0, v_1, \ldots, v_k$, and a sequence of event sets $\Sigma_0, \ldots, \Sigma_k$ satisfying:
   a) $C_0 = C, v_0 = v, \Sigma_0 = \Sigma$;
   b) $t_i \in \text{enabled}(C_{i-1}, v_{i-1}, \Sigma_{i-1})$;
   c) $C_i = \Omega(C_{i-1}, t_i), v_i = A(t_i, v_{i-1})$, and $\mathcal{E}_i = \mathcal{E}_{i-1} \cup B(t_i)$.

The function α defined above is a formal representation of the rules for determining the next configuration and variable values for each possible state of the statechart. The ordered set of chart-transitions $\alpha(C,v,\Sigma)$ that is taken by the statechart in response to the external inputs Σ is called a step, and the firing of each chart-transition in $\alpha(C,v,\Sigma)$ is referred to as a microstep. SC_α will denote a statechart SC with a given step-semantics α. The result of applying α from a give statechart state (C,v) and for an input event set Σ will be denoted by the next-step-map $\beta_\alpha(C,v,\Sigma)$.

To illustrate the concepts of steps and microsteps, consider the situation in Fig. 2, where $\mathcal{E}_I = \{a\}$. At the state $(C_3, 1)$, where $C_3 = \{r,P,Q,P1,Q0\}$, if event a occurs, both $t3$ and $t2$ are enabled. According to the MATLAB Stateflow step-semantics, chart-transition $t3$ is fired first in this situation and then $t2$ is fired. The chart-transition sequence in microsteps is $\alpha(C_3,1,\{a\}) = t3t2t4$. Note that chart-transition $t4$ is enabled due to the broadcast of event b after firing of chart-transition $t2$. An alternative step-semantics could be $\alpha'(C_3,1,\{a\}) = t2t3$, where $t2$ is fired before $t3$. For this step-semantics, $t4$ is not enabled.

[2] Note that while there is no bound on the lengths of the sequences in T^*, T^* contains only finite-length sequences.

III. Kripke Structures for Statecharts

Formal verification requires a formal semantics for the system behaviors and a formalism for specifications to be verified. As a formal semantics, we use Kripke structures [7]. A Kripke structure consists of a set of states, a set of transitions between states (represented by a Boolean relation), and a function that labels each state with a set of atomic propositions that are true in the state. Although they are simple, Kripke structures are sufficiently expressive to capture many aspects of temporal behaviors that are important for reasoning about reactive systems.

Definition 2—Statechart Kripke Structure: Given a statechart with step-semantics, SC_α, its associated statechart Kripke structure is a tuple $\hat{\mathcal{K}}_{\mathrm{SC}_\alpha} = \langle \hat{M}, \hat{m}_0, \hat{R}, \widehat{\mathrm{AP}}, \hat{L} \rangle$, where

- $\hat{M} \subset \mathcal{C} \times \mathcal{B}^{|\mathcal{V}|} \times 2^{\mathcal{E}_I}$ is a finite set of states;
- $\hat{m}_0 = (C_0, v_0, \emptyset)$ is the initial state, $C_0 \in \mathcal{C}$ is the initial configuration of $\mathcal{S}$, and v_0 is the initial value of the variables;
- $\hat{R} \subset \hat{M} \times \hat{M}$, the state transition relation, is defined as: for $(C, v, \Sigma), (C', v', \Sigma') \in \hat{M}, ((C, v, \Sigma), (C', v', \Sigma')) \in \hat{R}$, iff $(C', v') = \beta_\alpha(C, v, \Sigma')$;
- $\widehat{\mathrm{AP}} = Q \bigcup \mathcal{V}$ is the set of atomic propositions (where Q is the set of statechart states in SC_α) and;
- $\hat{L} : \hat{M} \rightarrow 2^{\widehat{\mathrm{AP}}}$ is the labeling function identifying the atomic propositions that are true for each state $x = (C, v, \Sigma)$, defined by $L(x) = C \cup \{v^j \in \mathcal{V} \,|\, v^j(x) = 1\}$, where $v^j(x)$ is the value of variable v^j at state x.

The labels for the states in the statechart Kripke structure are the active statechart states and the variables that are true for the given statechart configuration and variable valuations. We now introduce an additional type of Kripke structure to represent the microsteps given by α. This provides semantics for the microstep model used in the model checker $\mu\mathrm{SC}_\alpha$, as illustrated in Fig. 1. The microstep Kripke structure uses the notion of a standard Kripke structure equipped with marked states. These marked states provide the connection between the model checking results for the microstep model and the desired verification of the specification given for the original statechart.

Definition 3: A marked Kripke structure is a tuple $\mathcal{K}_m = \langle M, m_0, M_m, R, \mathrm{AP}, L \rangle$, where:

- M is a finite set of states;
- m_0 is the initial state;
- $M_m \subset M$ is a set of marked states, with $m_0 \in M_m$;
- $R \subset M \times M$ is a set of transitions;
- AP is a set of atomic propositions, with $\mathrm{marked} \in \mathrm{AP}$;
- $L : M \rightarrow 2^{\mathrm{AP}}$ is a labeling function that satisfies $\mathrm{marked} \in L(x)$ iff $x \in M_m$.

The Kripke structure for the microstep statechart model is a marked Kripke structure defined as follows.

Definition 4—Microstep Kripke Structure: Given a statechart with step-semantics $\mathrm{SC}_\alpha = \langle \mathcal{S}, \mathcal{V}, \mathcal{E}, \chi \rangle$, the microstep Kripke structure of SC_α is defined as $\mathcal{K}_{\mathrm{SC}_\alpha, m} = \langle M, m_0, M_m, R, \mathrm{AP}, L \rangle$, where:

- $M \subset \mathcal{C} \times \mathcal{B}^{|\mathcal{V}|} \times T^* \times 2^{\Sigma_I}$ is a finite set of states;
- $m_0 = (C_0, v_0, \varepsilon, \emptyset)$ is the initial state, $C_0 \in \mathcal{C}$ is the initial configuration of $\mathcal{S}$, and v_0 is the initial value of the variables, where ε is the empty string in T^*;
- $R \subset M \times M$ is defined as: for $x = (C, v, \tau, \Sigma)$ and $x' = (C', v', \tau', \Sigma') \in M, (x, x') \in R$, iff there exists a sequence of states $x_1, \ldots, x_l$, with $x_i = (C^i, v^i, \tau^i, \Sigma^i)), i = 1, \ldots, l$ and $l > 0$[3] satisfying the following conditions:
  1) $x_i = x$ and $x_{i+1} = x'$ for some i such that $1 \leq i \leq l - 1$;
  2) $x_1 = (C^1, v^1, \tau^1, \Sigma^1)$ is such that $\tau^1 = \varepsilon$ and $\alpha(C^1, v^1, \Sigma') \neq \varepsilon$;
  3) for $j = 1, \ldots, l - 1, x_{j+1} = (C^{j+1}, v^{j+1}, \tau^{j+1}, \Sigma^{j+1})$ is such that $\tau^1 = \alpha(C^1, v^1, \Sigma')$, and

$$(C^{j+1}, v^{j+1}, \tau^{j+1}) = \begin{cases} (\Omega(C^j, t_1), A(t_1, v^j), t_2, \ldots, t_k), & \text{if } \tau^j = t_1, \ldots, t_k, k > 1 \\ (\Omega(C^j, t_1), A(t_1, v^j), \varepsilon), & \text{if } \tau^j = t_1 \end{cases}$$

  and $\Sigma^{j+1} = \Sigma'$, where $|\tau|$ is the number of chart-transitions in τ and Ω is the configuration transition function for SC_α. It is straightforward to verify that $x_l = (C^l, v^l, \tau^l, \Sigma^l)$ is such that $(C^l, v^l) = \beta_\alpha(C, v, \Sigma')$.
- $M_m \subset M$ is defined recursively as follows:
  1) $(C_0, v_0, \varepsilon, \emptyset) \in M_m$;
  2) $x' = (C', v', \varepsilon, \Sigma') \in M_m$ if there is a $x = (C, v, \varepsilon, \Sigma) \in M_m$ such that $(C', v') = \beta_\alpha(C, v, \Sigma')$;
  3) all marked states are obtained by 1) and 2).
- $\mathrm{AP} = \widehat{\mathrm{AP}} \cup \{\mathrm{marked}\}$;[4]
- $L : M \rightarrow 2^{\mathrm{AP}}$ is defined for $x = (C, v, \tau, \Sigma)$ as $L(x) = \hat{L}(C, v, \Sigma) \cup \{\mathrm{marked}\}$ if $\tau = \varepsilon$; otherwise, $L(x) = \hat{L}(C, v, \Sigma)$.

The states in the above structure are all states reachable from the initial state of the statechart SC_α. The arcs are microsteps of SC_α. The structure is well defined since from the definition of step-semantics of statecharts $\alpha(C, v, \Sigma)$ is finite for all $(C, v, \Sigma) \in \mathcal{C} \times \mathcal{B}^{|\mathcal{V}|} \times 2^{\mathcal{E}_I}$. The labelling function identifies the marked states with the atomic proposition marked. We use simply the notation $\mathcal{K}_m$ to indicate the microstep Kripke structure for the statechart under consideration. The following lemmas follow directly from the definition of marked states in Definition 4.

Lemma 1: Suppose M and M_m are the state and marked state sets of $\mathcal{K}_m$. For a state $x = (C, v, \tau, \Sigma) \in M, x \in M_m$ iff $\tau = \varepsilon$.

Lemma 2: For a state $x \in M - M_m$, there is only one $x' \in M$ such that $(x, x') \in R$.

To make the connection between the statechart Kripke structure and the microstep Kripke structure, we introduce the following shrink operation for a marked Kripke structure.

Definition 5—Shrink of a Marked Kripke Structure: The shrink of $\mathcal{K}_m = \langle M, m_0, M_m, R, \mathrm{AP}, L \rangle$, denoted by $\tilde{\mathcal{K}}_m$ is a Kripke structure $\langle \tilde{M}, \widetilde{m_0}, \tilde{R}, \widetilde{\mathrm{AP}}, \tilde{L} \rangle$, where:

1) $\tilde{M} = \{(C, v, \Sigma) \,|\, (C, v, \varepsilon, \Sigma) \in M_m\}$;
2) $\tilde{m}_0 = (C_0, v_0, \emptyset)$;
3) $\tilde{R}$ is a Boolean relation defined as: for $x, x' \in M, (x, x') \in \tilde{R}$, iff there is a sequence of states $\pi = x_1, \ldots, x_l$ in M such that $x_1 = x, x_l = x', (x_i, x_{i+1}) \in R, i = 1, \ldots, l - 1$, and $x_j \notin M_m$, for $j = 2, \ldots, l - 1$, but $x_l \in M_m$;
4) $\widetilde{\mathrm{AP}} = \mathrm{AP} - \{\mathrm{marked}\}$;
5) The labeling function $\tilde{L}(x) = L(x) - \{\mathrm{marked}\}$.

[3] From the assumption that $\alpha(C^1, v^1, \Sigma') = \varepsilon$, we know that in fact $l > 2$.

[4] We assume that $\mathrm{marked} \notin \widehat{\mathrm{AP}}$.

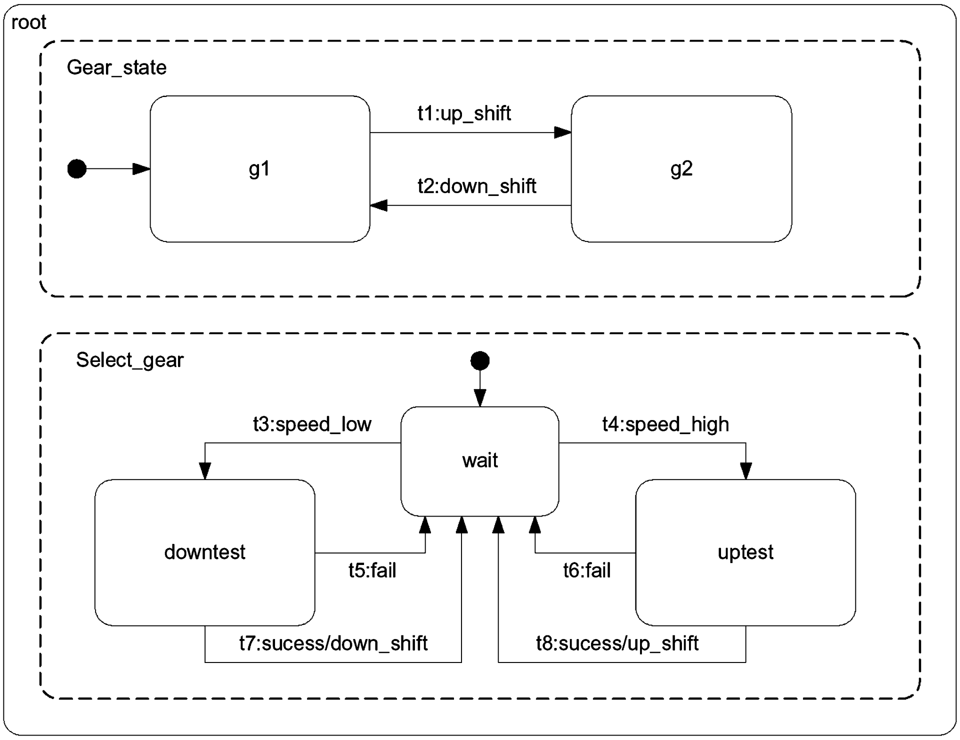

Fig. 3. Statechart for part of the gear shift controller.

In words, the shrink of a marked Kripke structure is a Kripke structure considering only transitions between marked states. To verify statecharts, it is not necessary to build either the microstep Kripke structure or the shrink of the microstep Kripke structure. These constructs are introduced as the semantics for establishing the correctness of the specification expansion and verification defined in the following section. A microstep statechart model is built only for the target model checker, as shown in Fig. 1. The following lemma follows from the definition of the shrink operation.

Lemma 3: Suppose M and M_m are the state and marked state sets of $\mathcal{K}_m$, respectively, R is the state transition relation of $\mathcal{K}_m$ and $\tilde{R}$ is the state transition relation of $\widetilde{\mathcal{K}}_m$. For two states $x, x' \in M_m$, if $(x, x') \in \tilde{R}$, then (x, x') is a step of SC_α.

This lemma provides the basis for the following proposition, which states that the shrink of the marked microstep Kripke structure is equivalent to the Kripke structure for the statechart, where equivalence here means the two structures are identical: they have the same sets of states, transitions, atomic propositions, and identical labeling functions. Proofs of this proposition and subsequent results are given in the Appendix.

Proposition 1: $\widetilde{\mathcal{K}}_m \equiv \hat{\mathcal{K}}$.

IV. Statechart Verification via Specification Expansion

Given the statechart Kripke structure as the semantics for statecharts, it is straightforward to introduce temporal logic specifications called CTL formulas [7].[5] CTL formulas describe properties of computation trees for Kripke structures in terms of the atomic propositions that are true at states along paths in the computation tree. The computation tree is formed (conceptually) by unwinding the Kripke structure from its initial state into an infinite tree with the designated state at the root. The computation tree shows all of the possible executions starting from the initial state. For our statechart Kripke structure, CTL formulas describe properties of sets of possible executions of a statechart starting from the initial configuration. The syntax for statechart CTL expressions is given as follows.

Definition 6—Statechart CTL Specification: Given a statechart SC_α, the set of CTL specifications defined on the statechart Kripke structure $\hat{\mathcal{K}}$ of the statechart, denoted $\mathrm{CTL}(\mathrm{SC}_\alpha)$, is given by:

1) $\widehat{\mathrm{AP}} \subset \mathrm{CTL}(\mathrm{SC}_\alpha)$;
2) if $\sigma_1, \sigma_2 \in \mathrm{CTL}(\mathrm{SC}_\alpha)$, then the following are in $\mathrm{CTL}(\mathrm{SC}_\alpha)$:
   - $\neg\sigma_1, \sigma_1 \vee \sigma_2, \sigma_1 \wedge \sigma_2$;
   - $\{\mathbf{A} \,|\, \mathbf{E}\} \,|\, \{\mathbf{X}(\sigma_1) \,|\, \mathbf{F}(\sigma_1) \,|\, \mathbf{G}(\sigma_1) \,|\, \sigma_1 \mathbf{U} \sigma_2\}$.
3) all expressions in $\mathrm{CTL}(\mathrm{SC}_\alpha)$ are defined by 1) and 2).

In the expressions above, $\mathbf{A}$ and $\mathbf{E}$ are universal and existential path quantifiers, respectively, and $\mathbf{X}, \mathbf{F}, \mathbf{G}$, and $\mathbf{U}$ are the temporal operators next, future, global, and until. Detailed information about CTL specifications can be found in [7]. The statechart verification problem now can be stated as follows.

Definition 7—Statechart Verification Problem: For a given statechart SC_α, and a given specification $f \in \mathrm{CTL}(\mathrm{SC}_\alpha)$, determine whether f is true for all possible executions of SC_α from the initial state; i.e., determine if $\hat{m}_0 \models_{\hat{\mathcal{K}}} f$, where $\hat{\mathcal{K}}$ is the Kripke structure for SC_α and $\hat{m}_0$ is the initial state of $\hat{\mathcal{K}}$.

[5] Other temporal logics could also be used, such as CTL* or linear time logic (LTL). We use CTL since it is the specification formalism for the SMV model checker.

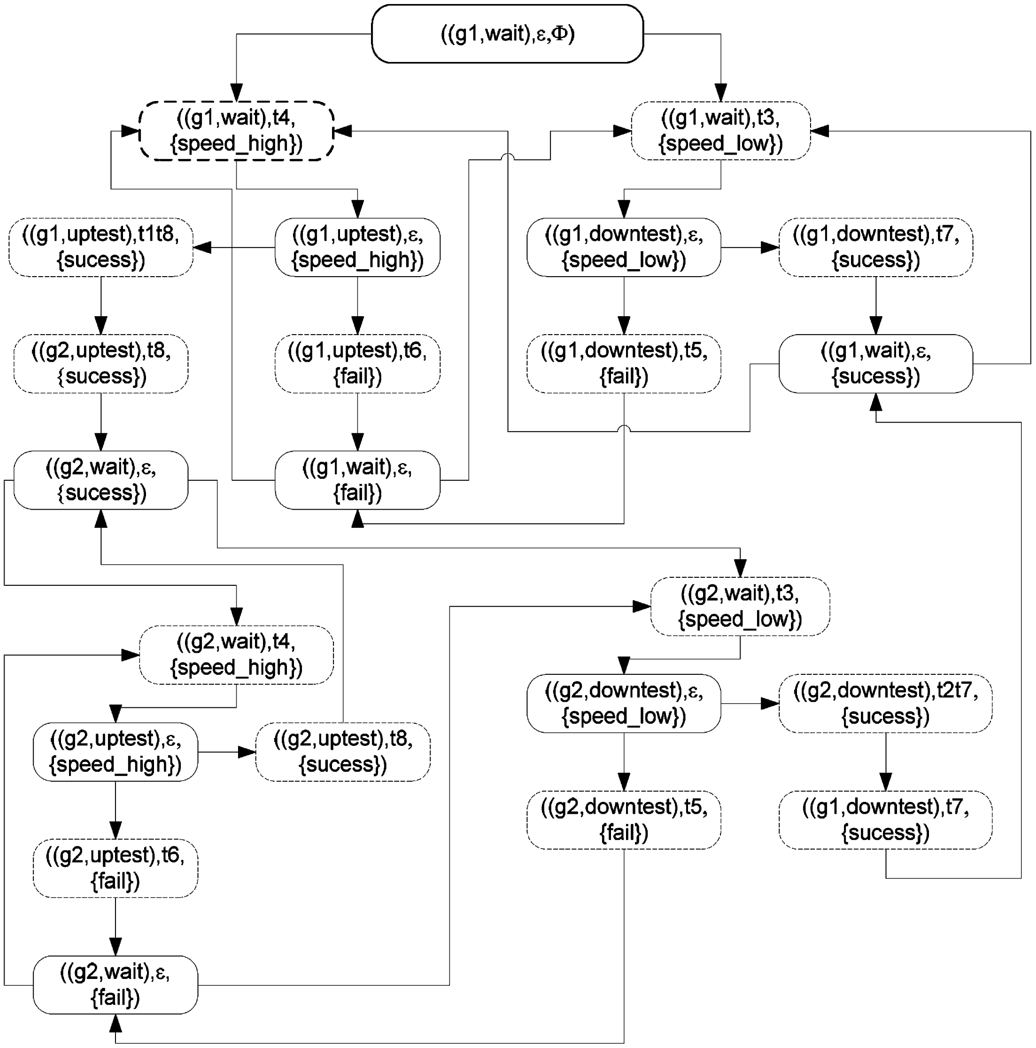
((g1,wait),ε,Φ)
((g1,wait),t4, {speed_high})
((g1,wait),t3, {speed_low})
((g1,uptest),t1t8, {sucess})
((g1,uptest),ε, {speed_high})
((g1,downtest),ε, {speed_low})
((g1,downtest),t7, {sucess})
((g2,uptest),t8, {sucess})
((g1,uptest),t6, {fail})
((g1,downtest),t5, {fail})
((g1,wait),ε, {sucess})
((g2,wait),ε, {sucess})
((g1,wait),ε, {fail})
((g2,wait),t3, {speed_low})
((g2,wait),t4, {speed_high})
((g2,downtest),ε, {speed_low})
((g2,downtest),t2t7, {sucess})
((g2,uptest),ε, {speed_high})
((g2,uptest),t8, {sucess})
((g2,downtest),t5, {fail})
((g1,downtest),t7, {sucess})
((g2,uptest),t6, {fail})
((g2,wait),ε, {fail})

Fig. 4. Marked Kripke structure for the statechart in Fig. 3; marked states are indicated with solid borders.

As illustrated in Fig. 1, we solve the statechart verification problem by expanding the given CTL expression so that it applies to the microstep Kripke structure for the expanded microstep model $\mu\mathrm{SC}_\alpha$ delivered to the model checker. CTL specifications for the microstep model, denoted $\mathrm{CTL}(\mu\mathrm{SC}_\alpha)$, are defined using the set of atomic predicates for the microstep Kripke structure $\mathcal{K}_m$ for the statechart SC_α; that is, using the atomic predicates AP. The expansion of a statechart CTL expression into a CTL specification for the microstep model is defined as follows.

Definition 8—Expansion Operator for Statechart CTL Specifications: The expansion operator $\lambda : \mathrm{CTL}(\mathrm{SC}_\alpha) \rightarrow \mathrm{CTL}(\mu\mathrm{SC}_\alpha)$ is defined by the following production rules. For $p \in \widehat{\mathrm{AP}}, f_1, f_2 \in \mathrm{CTL}(\mathrm{SC}_\alpha)$:

- $\lambda(p) = p$;
- $\lambda(f_1 \wedge f_2) = \lambda(f_1) \wedge \lambda(f_2)$;
- $\lambda(\neg f_1) = \neg\lambda(f_1)$;
- $\lambda(\mathbf{EX}f_1) = \mathbf{EX}\,\mathbf{E}(\neg\ \mathrm{marked}\,\mathbf{U}(\mathrm{marked}\ \wedge \lambda(f_1)))$;
- $\lambda(\mathbf{EG}f_1) = \mathbf{EG}(\mathrm{marked}\ \rightarrow \lambda(f_1))$;
- $\lambda(\mathbf{E}f_1\mathbf{U}f_2) = \mathbf{E}[(\mathrm{marked} \rightarrow \lambda(f_1))\mathbf{U}(\mathrm{marked} \wedge \lambda(f_2))]$.

The image $\lambda(f)$ for $f \in \mathrm{CTL}(\mathrm{ST}_\alpha)$ is called the expansion of f.

The expansion function λ is well defined since all the elements in $\mathrm{CTL}(\mathrm{SC}_\alpha)$ can be built recursively by sequentially applying the two logic operators $\wedge, \neg$, the path quantifier $\mathbf{E}$, and three temporal operators $\mathbf{X}, \mathbf{G}, \mathbf{U}$ [7].

The following proposition states that the verification of a given CTL specification for a statechart can be accomplished by verifying the expanded specification for the associated microstep model.

Proposition 2: For each statechart CTL formula $f, \hat{m}_0 \models_{\widehat{\mathcal{K}}} (f)$ if and only if $m_0 \models_{\mathcal{K}_m} \lambda(f)$.

V. Example

In this section, we illustrate the concepts in the previous sections using the MATLAB Stateflow step-semantics for a statechart model of a gear shift controller, shown in Fig. 3. This example is part of a demonstration example in the Stateflow toolbox. In Fig. 3, two gear levels `g1` and `g2` are considered. The gear shift controller works as follows. When the gear state is `g1` the low gear, and the event `speed_high` occurs, the controller enters a testing mode uptest, to make sure the up_shift command should be issued. The test is performed by an external function that issues the fail or success input to the Stateflow diagram. If the success input is received, the up_shift event is broadcasted from the transition from the uptest to wait state and this triggers the transition to g2. On the other hand, if fail is received, no gear shift is triggered by the transition to the wait state. A similar transition sequence occurs when the gear state is g2 and the speed_low event is received, passing in this case through downtest state rather than uptest. We have for this example, the input event set $\Sigma_I = \{\texttt{speed\_high},\texttt{speed\_low},\texttt{success},\texttt{fail}\}$ and broadcasted event set $\Sigma_{\rm BC} = \{\texttt{up\_shift},\texttt{down\_shift}\}$. We assume that the input events are mutually exclusive, i.e., only one event can happen at a given time instant.

Fig. 4 shows the microstep Kripke structure $\mathcal{K}_m$ corresponding to MATLAB Stateflow step-semantics. In $\mathcal{K}_m$, we use the pairs of basic states in Gear_state and Select_gear to represent the configurations. Thus, for example, the initial configuration given by the default states is $(\texttt{g1},\texttt{wait}) = \{\texttt{root},\texttt{Gear\_state},\texttt{Select\_gear},\texttt{g1},\texttt{wait}\}$.

Consider a specification $f = \mathbf{EF}((\texttt{g1},\texttt{uptest}) \wedge \mathbf{EX}(\texttt{g2},\texttt{uptest}))$. In words, this specification checks whether there is a path with two configurations (g1, uptest) and (g2, uptest) in sequence in the Kripke structure of SC_α. If it were true, then there would be a fault in the gear shift controller, since it would not respond to the speed_low event when the gear shifts to state g2. This could be a dangerous situation that would block the automatic downshift action.

This specification (without expansion) is true for marked Kripke structure $\mathcal{K}_m$ in Fig. 4, but this is an incorrect result because the specification is true only during the execution of the microsteps. To verify f on $\hat{\mathcal{K}}$, we need to check the following expanded specification $\lambda_{\mathcal{K}_m}(f)$ on $\mathcal{K}_m$:

$$\begin{aligned}\lambda_{\mathcal{K}_m}(f) = \mathbf{EF}(&\text{marked } \wedge (\texttt{g1},\texttt{uptest},\text{marked})\\ &\wedge \mathbf{EXE}(\neg \text{ marked } \mathbf{U}(\texttt{g2},\texttt{uptest},\text{marked}))).\end{aligned}$$

Here, we have used the fact that $\mathbf{EF}f \equiv \mathbf{E}(\text{true } \mathbf{U} f)$ [7] and the initial state of $\mathcal{K}_m$ is marked. This specification is false when we check it on $\mathcal{K}_m$, which means our design in Fig. 3 is correct in the sense that the gear shift controller will not stay at uptest after gear state transitions from g1 to g2.

VI. Conclusion

This brief presents a Kripke structure and CTL temporal logic specification language for a broad class of statecharts. To perform verification using a standard finite-state model checker, we show how the CTL expression can be expanded so that it applies to the microstep model of the statechart semantics. We developed a tool *sf2smv* that generates the microstep model of Stateflow diagrams based on the MathWorks step-semantics for the symbolic model checker SMV [7]. The work presented here formalizes the operation of *sf2smv* for general statechart step-semantics.

Verification using a finite-state model checker relies on two results in this brief. First, we show that the statechart Kripke structure representing the macro behavior of a statechart is identical to the Kripke structure involving only the marked states of the microstep Kripke structure for the detailed microsteps defined by the statechart step-semantics. Second, we show that the CTL specifications for the statechart are true if and only if the expanded specifications are true for the microstep model. These results hold for statecharts with a finite number of input events, a finite number of variable values, inter-level transitions, and internal event broadcasting. We assume deterministic step-semantics, since determinism is usually desired for control applications, particularly when real-time control software will be generated directly from the statechart model.

With a slight increase in notational and computational complexity, the approach in this brief can be extended to handle state history (that is, the ability to reactivate a state transition structure in an OR state that was active when the OR state was previously deactivated) and nondeterministic step-semantics, albeit nondeterministic step-semantics are useful when statecharts are used for requirements specifications. We assume the number of microsteps required to execute any possible step in the possible behaviors for the given statechart is bounded. As with nondeterminism, the assumption of bounded behavior is not restrictive for models of real-time control logic. Even if the step-semantics admits unbounded behaviors, statecharts that would lead to an infinite-length microstep sequence for a single step are normally disallowed through rigorous style specifications that prohibit such behaviors. We assume the statecharts being verified adhere to such guidelines.

Appendix
Sketch of the Proofs

Proof of Proposition 1: According to definition, the state set $\hat{M}$ of $\hat{\mathcal{K}}$ consists of those states that can be reached by steps from the initial state. It is clear that $\hat{M}$ and M_m have a one-to-one correspondence. From Lemma 1 and the construction of marked state set M_m of $\mathcal{K}_m$, we know that the set of marked states $(C, v, \varepsilon, \Sigma)$ consists of those states that can be reached from the initial state and satisfy the common condition $\tau = \varepsilon$. So $\tilde{M}$ and M_m have a one-to-one correspondence.

Next, we show that $((C, v, \Sigma), (C', v', \Sigma')) \in \hat{R}$ iff $((C, v, \varepsilon, \Sigma), (C', v', \varepsilon, \Sigma')) \in \tilde{R}_m$. It is straightforward to show $(C, v, \Sigma), (C', v', \Sigma') \in \hat{R}$ implies $((C, v, \varepsilon, \Sigma), (C', v', \varepsilon, \Sigma')) \in \tilde{R}_m$. To show the reverse, suppose $((C, v, \varepsilon, \Sigma), (C', v', \varepsilon, \Sigma') \in \tilde{R}_m$. It follows that $(C', v') = \beta_\alpha(C, v, \Sigma')$. This implies $((C, v, \Sigma), (C', v', \Sigma')) \in \hat{R}$. Hence, $\hat{M}$ and $\tilde{M}$ have a one-to-one correspondence. To complete the proof, observe that $\tilde{L}(C, v, \varepsilon, \Sigma) = L(C, v, \varepsilon, \Sigma) - \{\text{marked}\} = \hat{L}(C, v, \Sigma)$. ■

Proof of Proposition 2: From Proposition 1, we only need to prove that for every f for the Kripke structure $\widetilde{\mathcal{K}}_m, \hat{m}_0 \models_{\widetilde{\mathcal{K}}_m} f$ is true if and only if $m_0 \models_{\mathcal{K}_m} \lambda(f)$. Since each CTL formula can be expressed in terms of three CTL operators $\mathbf{EX}$, $\mathbf{EG}$ and $\mathbf{EU}$ with the help of logic operators $\neg$ and $\wedge$, we perform induction on the nested layer f of CTL operators. Let $\mathrm{CTL}(\mathrm{SC})_k$ denote the set of formulas f with k layer nested CTL operators

$\mathbf{EX}, \mathbf{EU}$, and $\mathbf{EG}$ and logic operations $\neg$ and $\wedge$. Assume that $\mathrm{CTL(SC)}_0 = \{p \,|\, p \in \widehat{\mathrm{AP}}\}$.

For $k = 0, f = p$, with $p \in \widehat{\mathrm{AP}}$. For all $x \in M_m, x \vDash_{\widetilde{\mathcal{K}}_m} f$ iff $p \in \tilde{L}(x) = L(x) - \{\mathrm{marked}\}$ since marked $\notin \widehat{\mathrm{AP}}$. It follows from $\lambda(f) = \lambda(p) = p$ that $x \vDash_{\mathcal{K}_m} f$.

Now suppose that for a given $k \geq 0$, for all $x \in M_m, x \vDash_{\widetilde{\mathcal{K}}_m} f$ iff $x \vDash_{\mathcal{K}_m} \lambda(f)$ holds for every $f \in \mathrm{CTL(SC)}_k$. To prove that this holds for every $f \in \mathrm{CTL(SC)}_{k+1}$, let $f', f_1, f_2 \in \mathrm{CTL(SC)}_k$. There are five cases: 1) $f = \neg f'$; 2) $f = f_1 \wedge f_2$; 3) $f = \mathbf{EG}f'$; 4) $f = \mathbf{E}(f_1 \mathbf{U} f_2)$; 5) $f = \mathbf{EX}f'$. Case 1) and Case 2) are straightforward. We give details for Case 3); the proofs for Cases 4 and 5 are similar.

Case 3) Let $f = \mathbf{EG}f', f' \in \mathrm{CTL(SC)}_k.x \vDash_{\widetilde{\mathcal{K}}_m} f$ iff there is a path $\pi = x_1, x_2, \ldots,$ in $\widetilde{\mathcal{K}}_m$, s.t., $x_i \vDash_{\widetilde{\mathcal{K}}_m} f', i = 1, 2, \ldots$, where $x_1 = x$. From the induction assumption, $x_i \vDash_{\mathcal{K}_m} \lambda(f'), i = 1, 2, \ldots$. Since there is exactly one segment $\pi_i = x_{i_1} x_{i_2}, \ldots, x_{i_k}$ in $\mathcal{K}_m$ such that $x_{i_1} = x_i, x_{i_k} = x_{i+1}, x_{i_j} \notin M_m, j = 2, \ldots, k-1$, we have marked $\in L(x_i)$ and for each segment π_i, marked $\notin L(x_{i_j}), j = 2, \ldots, k-1$. Thus, $x_{i_j} \vDash_{\mathcal{K}_m} \mathrm{marked} \rightarrow \lambda(f'), i = 1, 2, \ldots, j = 2, \ldots, k$. Thus, $x \vDash_{\mathcal{K}_m} \mathbf{EG}(\mathrm{marked} \rightarrow \lambda(f'))$. Conversely, if $x \vDash_{\mathcal{K}_m} \mathbf{EG}(\mathrm{marked} \rightarrow \lambda(f'))$, there is a path $\pi = x_1, x_2, \ldots,$ in $\mathcal{K}_m$, s.t., $x_i \vDash_{\mathcal{K}_m} \mathrm{markded} \rightarrow \lambda(f'), i = 1, 2, \ldots$, where $x_1 = x$. That means if $x_i \in M_m$ then $x_i \vDash_{\mathcal{K}_m} \lambda(f')$. By the induction assumption, $x_i \vDash_{\widetilde{\mathcal{K}}_m} f'$. Since marked $\in L(x_i)$ iff $x_i \in M_m$, we have that $\pi' = x_{i_1}, x_{i_2}, \ldots,$ where $i_1 < i_2 <, \ldots$, s.t., $x_k \notin M_m$ for $k \notin \{i_1, i_2, \ldots, \}$ is a path in $\widetilde{\mathcal{K}}_m$. This implies $x \vDash_{\widetilde{\mathcal{K}}_m} \mathbf{EG}(f')$. ■

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers and the Associate Editor for many helpful suggestions. They would especially like to thank an anonymous reviewer for pointing out the work in [5] and [8].

REFERENCES

[1] M. von Der Beek, "A comparison of statecharts variants," in *Proc. Formal Techniques Real Time Fault Tolerant Syst. Lecture Notes Comp. Sci. (LNCS) 863*, 1994, pp. 128–148.

[2] S. Bensalem, "A methodology for proving control systems with Lustre and PVS," in *Dependable Computing for Critical Applications 7*. San Jose, CA: Comp. Soc. Press, 1999, pp. 89–107.

[3] C. Banphawatthanarak, B. H. Krogh, and K. Butts, "Symbolic verification of executable control specifications," in *Proc. IEEE Conf. Comput.-Aided Contr. Syst. Des.*, 1999, pp. 581–586.

[4] C. Banphawatthanarak, "Verification of stateflow diagrams using SMV," M.S. thesis, Dept. Elect. Comput. Eng., Carnegie Mellon Univ., Pittsburgh, PA, 2000.

[5] T. Bienmüller, "Reducing complexity for the verification of statemate designs," Ph.D. dissertation, Berichte aus dem Deartment für Informatik, Nr. 6/2003, Oldenburg Univ., Oldenburg, Germany, 2003.

[6] W. Chan, R. J. Anderson, P. Beame, and D. H. Jones, "Decoupling synchronization from local control for efficient symbolic model checking of statecharts," in *Proc. Int. Conf. Softw. Eng. (ICSE)*, 1999, pp. 142–151.

[7] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA: MIT Press, 1999.

[8] W. Damm, B. Josko, H. Hungar, and A. Pnueli, , W.-P. de Roever, H. Langmaack, and A. Pnueli, Eds., "A compositional real-time semantics of STATEMATE designs," in *Proc. COMPOS'99, Lecture Notes in Computer Science 1536*. Berlin, Germany: Springer–Verlag, 1998, pp. 186–238.

[9] N. ay, "An example of linking formal methods with CASE tools," in *Proc. Conf. Centre Adv. Studies Collaborative Research: Softw. Eng.*, 1993, vol. 1, pp. 97–107.

[10] D. Harel, "Statecharts: A visual formalism for complex systems," *Sci. Comput. Program.*, vol. 8, pp. 231–274, Jul. 1987.

[11] D. Harel and A. Naamad, "The STATEMATE semantics of statecharts," *ACM Trans. Software Eng. Technol.*, vol. 5, no. 4, pp. 293–333, 1996.

[12] J. Helbig and P. Kelb, "An OBDD-Representation of statecharts," in *Proc. Euro. Des. Test Conf.*, 1994, pp. 142–129.

[13] A. Maggiolo Schettini, A. Peron, and S. Tini, "Equivalences of statecharts," in *Proc. Conf. Concurrency Theory, LNCS 1119*, 1996, pp. 687–702.

[14] A. Maggiolo Schettini and S. Tini, "Projectable semantics for statechart," in *Proc. Elect. Notes Theoretical Comp. Sci.*, 1998, pp. 134–146.

[15] M. Marcotty and H. Ledgard*, The World of Programming Languages*. Berlin, Germany: Springer-Verlag, 1986.

[16] *"Stateflow User's Guide,"* The Mathworks Inc., Natick, MA, Sep. 1999.

[17] S. Nadjm Tehrani and J.-E. Strömberg, "Formal verification of dynamic properties in an aerospace-application," *Formal Methods In System Design*, vol. 14, pp. 135–169, Mar. 1999.

[18] A. Pnueli and M. Shalev, "What is in a step: On semantics of statecharts," in *Proc. LNCS*, 1991, pp. 244–264.

[19] M. Rausch and B. H. Krogh, "Symbolic verification of stateflow logic," in *Proc. Int. Workshop Discrete Event Syst. (WODES98)*, 1998, pp. 37–42.

[20] A. Sowmya and S. Ramesh, "Extending statecharts with temporal logic," *IEEE Trans. Softw. Eng.*, vol. 24, no. 3, pp. 216–231, Mar. 1998.

[21] A. C. Uselton and S. A. Smolka, "A compositional semantics for statecharts using labeled transition systems," in *Proc. LNCS 836*, 1994, pp. 2–17.