On Modelling and Reasoning About Hybrid Systems

R. C. Sekar, Yow-Jian Lin and Sanjai Narain
Bellcore, 445 South Street, Morristown, NJ 07962
E-mail: {sekar,yowjian,narain}@thumper.bellcore.com

Abstract

This paper concerns the development of formal approaches for specification, simulation and validation of real-time reactive systems. Most previous approaches to this problem have focussed on *discrete systems* wherein the behavior of the system is described as a sequence of discrete events or state changes; they do not provide a satisfactory solution for a wide range of *hybrid systems* whose behavior depends on discrete as well as continuous parameters. Several approaches have been proposed to model such hybrid systems, but they are not amenable to efficient simulation. We also show that these approaches cannot conveniently express a variety of triggering and preemption policies. We then present our approach to overcoming these problems, based on our model TRACE(H). Our approach is aimed at retaining the (relative) simplicity of discrete systems as far as possible, while still permitting descriptions of continuous behavior. Moreover, our approach can express many different triggering, enabling and preemption policies so as to conveniently model diverse hybrid systems. We also illustrate how simulation and mechanical reasoning can be accomplished for systems represented in TRACE(H). Trade-offs between expressive power and computational complexity are also discussed.

Keyword Codes: F.3.1, I.6.0, I.6.4
Keywords: Logics and Meanings of Programs, Specifying and Verifying and Reasoning about Programs; Simulation and Modelling, General; Model Validation and Analysis.

1 Introduction

This paper concerns the development of formal approaches for specification and analysis of real-time systems such as communication protocols and process control programs. Most previous approaches to this problem [2, 3, 5, 6, 7, 10, 12, 18] have focussed on *discrete systems* wherein the behavior of the system is described as a sequence of discrete events or state changes , each of which take zero time. Although this approach is adequate for modelling systems that consist only of discrete components such as digital computers, it is not satisfactory for a wide range of systems whose behavior parameters vary continuously. For instance, in modelling the communication between an airplane pilot and an air-traffic controller, we must take into account continuous parameters such as the position and velocity of the airplane. We also need to take into account discrete components such as the computers coordinating or controlling various activities of the pilot and the air-traffic controller. This paper proposes an approach to specify and analyze such *hybrid systems* that consist of a combination of discrete and continuous processes.

Hybrid systems occur naturally in many applications such as motion planning and control systems (e.g., guidance systems and robots) and concurrent/real-time systems (e.g., communication protocols, process control programs and operating systems). Our interest in hybrid systems arose in the context of communication systems. In particular, we are using such models for:

Containment of fault propagation in packet networks: An analysis of recent outages in packet networks suggests that many failures may be related to network elements developing hardware or software faults, which are then propagated to other connected nodes. We are currently developing models of SS7 (Signalling System No.7) [15], a widely used protocol in telephone networks, to study this problem. We aim to develop techniques to localize or prevent fault propagation. In the context of this research we need to model several continuous parameters such as the level of traffic or congestion in various nodes and links.

Analysis of Feature Interactions: One of the critical obstacles to rapid deployment of intelligent features in telephone systems is that new features may interact in an undesirable way with existing features — a phenomenon called *feature interaction* [7, 8]. We are currently developing a high-level model of telecommunication systems to study and identify such feature interactions. In this context, we need to take into account that some of the network elements such as mobile phones are not stationary and their behavior is related to the continuous parameter, viz., position. We may also need to model resource usages (such as cost of a call) which are often functions of time.

Interoperability analysis of SONET: SONET (Synchronous Optical NETwork) [4] is a new standard for fiber optics communications proposed by Bellcore. In this work we model the hardware and software developed by different vendors and use simulation techniques (augmented with some reasoning capabilities) to determine if they are interoperable. At the hardware level, we need to model continuous quantities such as propagation of optical signals in a fiber-optic cable and waveforms of these signals.

In all these cases, the system being modelled consists predominantly of discrete components. However, their behaviors are also affected by some continuous parameters which must be taken into account for obtaining accurate results. Even in the case of parameters that change discretely, it is often simpler for the purposes of simulation and reasoning to model their change by a continuous function rather than as a number of discrete steps. In this context, it is desirable to retain the (relative) simplicity of reasoning and efficiency of simulation permitted by discrete event techniques as far as possible, while still permitting descriptions of continuous behavior. Furthermore, these systems are often very large and complex, making complete validation infeasible with existing technology. Therefore we have to rely on a combination of simulation and validation techniques in order to understand the behavior of complex communication systems. Although techniques for modelling hybrid systems (such as in [13, 14]) have been proposed before, their methods do not attempt to exploit the (relative) simplicity of discrete systems. Furthermore they do not address several important issues such as the ability to conveniently express various policies for enabling and preempting state transitions. In this paper we present a way of modelling hybrid systems that addresses all of these issues. Our model, called TRACE(H)[1], is inspired by the earlier discrete modelling works of Cameron and Lin [7] and the approach of DMOD [16, 17]. We wish to point out that the main contribution of this paper is in presenting an approach to modelling and reasoning hybrid systems that retains the simplicity of discrete systems and in identifying issues in conveniently expressing enabling and preemption policies.

The rest of this paper is organized as follows. In Section 1.1 we provide a brief overview of some of the previous approaches to hybrid modelling and describe their drawbacks. Following

---

[1] TRACE(H) stands for **T**emporal **R**easoning for **A**nalyzing **C**ausality and **E**ffects (for **H**ybrid systems).

this, we provide an overview of our approach and summarize its salient features in Section 1.2. Section 2 presents our model and illustrates our modelling technique through examples. We then present a simulation algorithms for TRACE(H) specifications in Section 3. Some of our reasoning techniques are described and illustrated in Section 4. Finally, we conclude the paper in section 5 with a discussion some of the issues in modelling hybrid systems.

1.1 Previous Work and Motivation for a new Model

Maler, Manna and Pnueli have proposed a simple and elegant approach for modelling hybrid systems in [13]. In their approach, the state of a system is modelled by a combination of discrete and continuous state variables. The behavior of the system is governed by a set of transition relations of the form $C \wedge E$ (we will refer to C as the *cause* of the transition and E as the *effect.*) where C is a condition on the current state of the system and E is a conjunction that specifies the values of state variables in the next state (as a function of the current state). For continuous variables, the values are specified as a time-dependent function using differential equations. Each transition relation is associated with a time interval $[l, h]$. This transition relation specifies that if C holds continuously for any period of time t, $l \leq t \leq h$, then the system will make a transition to a state that satisfies E. The computation of a hybrid system is given by a *hybrid trace* that describes a behavior of the system as a sequence of discrete transitions interspersed with intervals during which the values of continuous variables may change. Although this model clarifies modelling hybrid systems, it does not address some important issues:

Mechanical Computation of Behaviors: Although they specify a computational model, this specification only determines whether a *given hybrid trace* is a computation of the system; no algorithm is provided to *actually compute* any such trace. A mechanical solution to the computation of traces is very difficult in their system since they allow conditions on continuous variables such as $x = y$ in the cause part of a transition. Since continuous variables x and y are specified as functions of time, say $f(t)$ and $g(t)$, identifying when this transition will be enabled requires computation of the smallest time T such that $f(T) = g(T)$. The problem is actually even more complex since the system may undergo transitions before T that change the functions f and g governing the values of the continuous variables x and y.

Enabling Policies: In their model, suppose that there is a transition with time interval $[l, l]$, cause $x = 1$ and effect $y = y + 1$. If the value of x continues to remain at 1 for a long period of time, then the value of y will keep on getting incremented. (Depending on the interpretation, the increments may occur infinitely often or every l units of time.) If the intention of a programmer is that the increment of y must be enabled only by the value changes of x to 1, then the programmer must use an auxiliary control variable in the following manner to disable this transition as soon as it is taken once:

$$cause :: ((x = 1) \wedge (ControlVar = 0)) \qquad effect :: ((y = y + 1) \wedge (ControlVar = 1))$$

Furthermore, every time this transition is to be enabled, we must explicitly reset the value of the control variable to zero. Thus, in their model, it is cumbersome to model "edge-triggered transitions", i.e., those transitions that take place when the cause becomes true after being false. Such transitions arise very often in descriptions of systems, e.g., edge-triggered hardware circuits. Furthermore, if we use a "once-enabled[2]" policy on the cause then the "level-triggered"

[2]This means that a transition will take place after l units of time following any instant when the cause holds. See [7] for definitions and examples of "once-enabled" contrasted with "continuously-enabled".

transition mechanism (i.e., transitions take place as long as the cause condition holds) used in [13] may not be meaningful: In the presence of real-valued time, observe that the transition will take place an infinite number of times whenever the cause holds for any finite duration.

Modelling Preemption: Preemption of a state change by another is common in real-time systems. In the approach of [13], we may need to use control variables in the cause of a transition T whose effect needs to be preempted. A state change may preempt T by assigning a value to this control variable that falsifies the cause of T. Note that this mechanism will work only if the continuously-enabled policy is used. Alternative policies such as once-enabled policies cannot be made use of even when such a policy reflects the behavior of a system more naturally. Use of control variables may make modelling cumbersome. Since the role of such variables is not immediately obvious, they can make it difficult to understand or reason about such specifications.

In summary, communication systems are often very large and complex. This means that we have to rely on simulation to a large extent for understanding their behavior and hence efficient simulation algorithms are crucial. At the same time, we also like to have a framework based upon which some mechanical checking and reasoning are possible. Secondly, these systems often consist of many different kinds of subsystems. A triggering mechanism or enabling policy or preemption mechanism that is appropriate for one subsystem is often inappropriate for another subsystem. Therefore we need to be able to conveniently express a rich set of enabling and triggering policies (or preemption mechanisms) in the specification. All these requirements have motivated the development of a new model TRACE(H) for modelling hybrid systems.

Several other approaches such as [1, 3, 19] also address the modelling of continuous time behavior, but do not directly address the problem of modelling continuous state parameters. For example, in Zeigler's DEVS (Discrete EVent Specified) model [19] each system state s is associated with a non-negative real number as the maximum time the system is allowed to stay in s. The system will make a transition from state s to another state either if the time has expired (an *internal* transition) or if an event has arrived (an *external* transition). Simulations can be done based on event-driven or fixed step time-advance strategies. Although the internal transition naturally models a time-out (a special kind of preemption), the model cannot easily express many other kinds of preemption conditions. Aggarwal et al. extended the discrete time IC* model to its continuous time semantics [1]. With the notion of relativized descriptions, they are able to model and simulate systems by converting discrete time systems of various pulse frequencies into a single continuous time system operating on a finest common pulse. However, the continuous time IC* model does not have the notion of continuous state variables. As a result, it is difficult to model continuous behavior such as the position of a moving object.

1.2 Overview of Our Approach and its Salient Features

We propose a new model TRACE(H) that addresses the above-mentioned issues. In our model, we allow state variables as well as events. We believe that this allows a more natural modelling of a system than using either state variables or events alone. For instance, the state of a communicating finite state machine can be modelled by state variables whereas its communication activities can be modelled by events. The transition rules have five distinct components: *causing trigger, causing condition, effect action, effect condition* and *transition time*. These are denoted respectively by CT, CC, EA, EC and TT. The transition is enabled whenever CT becomes true (after being false), provided that CC holds at that time. The effect of the transition EA will take place a duration t (specified in TT) after it is enabled, provided the temporal formula EC

holds in between.[3] Separating the transition rule into these components not only enables us to conveniently express different enabling and triggering policies but also makes our intentions explicit in the specification. This makes it more convenient for a reasoning procedure to analyze our specifications as compared to one that makes use of many control variables. We now summarize the salient features of our work and explain how it resolves some of the issues raised in the previous section.

• In our model, we do not permit continuous variables in the trigger. This enables us to develop an efficient algorithm for simulating (or computing with) a specification. In Section 2.1 we explain via examples how we can rewrite transitions that are to be triggered based on values of continuous variables into transitions that are triggered by discrete changes.

• Separating the cause into a cause trigger and a cause condition enables us to naturally specify edge-triggering conditions as well as level-triggering conditions. Both types of conditions occur naturally in systems such as hardware.

• The effect condition enables us to conveniently model preemption without having to use control variables. Furthermore, it enables us to separate the enabling policy from modelling itself. Our model is flexible enough to naturally describe both "continuously enabled" and "once enabled" policies or any other policy in between the two extremes [7]. This makes it more convenient since different policies may be natural under different circumstances.

• We present two simple mechanisms called *backward analysis* and *forward analysis* for reasoning about hybrid systems. Although these techniques cannot establish all interesting properties of a system, we show how they can be used in conjunction to prove several interesting properties of hybrid systems. Furthermore, both these techniques are amenable to mechanization. We believe that reasoning in TRACE(H) is simplified because control variables are not frequently used.

• Our work clarifies many of the relationships and differences between discrete systems and hybrid systems. It identifies the increase in complexity brought in by the presence of continuous variables. In particular, we discuss (see Section 5) possible extensions to our model and describe the capabilities that would be required for mechanical simulation or reasoning about the system. This brings out some of the trade-offs between expressive power and computational complexity.

2 A Model of Real-Time Systems

A real-time hybrid reactive system M in our model is represented by $\langle E, V, \Sigma, \mathcal{T} \rangle$:

- E is a set of observable system *events*. $e(t)$ denotes the occurrence of an event e at time t.
- V is a set of *variables* constituting the representation of system states. The set V is further divided into two disjoint sets V_{dis} (*discrete* variables) and V_{con} (*continuous* variables). All value changes of a discrete variables are discrete (i.e., take zero time for change) whereas values of continuous variables can change continuously with time and specified as a (continuous) function of time. The value of any variable v at a time t is denoted $v(t)$.
- Σ is a set of *states* of M. Each state $\sigma \in \Sigma$ is an acceptable combined interpretation of variables in V. $\sigma[x]$ represents the value assigned to the variable x in state σ. Any state $\sigma \in \Sigma$ can be designated as the initial state *IS*.

[3] Currently, we do not use time intervals. Some of the issues in doing so are discussed in Section 5.

- $\mathcal{T}$ is a set of transition rules that describe when state changes (or events) can occur in the system. They are of the form

  $CT(T)/CC \to EA(T')/EC$ **where** TT

  where CT, CC, EA, EC and TT stand for *causing trigger, causing condition, effect action, effect condition*, and *transition time*, respectively. For convenience, elements of a transition rule τ are referred to as $CT(\tau), CC(\tau), EA(\tau), EC(\tau)$, and $TT(\tau)$. If $CC(\tau)$ (or $EC(\tau)$) is always true, we can omit CC (or EC) in the transition rule.

  The meaning of a transition rule τ is as follows. If $CT(\tau)$ is true of the state of a system at time T but does not hold for some nonzero interval of time preceding T then we say that this transition is *triggered* at T. (In other words, the transition is edge-triggered.) If $CC(\tau)$ also holds at this time then we say that the transition is *enabled* at T. The effect of an enabled transition will take place at a time T' (with the relationship between T and T' specified in $TT(\tau)$), provided that the temporal formula $EC(\tau)$ holds in between T and T'. When the effect takes place, if $EA(\tau)$ is an event e then this event will take place at T'. Note that a transition with one or more events in CT will be triggered at a time T if all these events occur at T and the rest of CT evaluates to true at T.

  The following assumptions apply to components of any transition rule τ.

  - We require that $CT(\tau)$ consists of a conjunction of atomic formulas each of which is either an event occurrence or a condition on the value of a state variable. To be able to easily compute the occurrence time of triggers, the formula must not refer to any continuous variable.[4]
  - $CC(\tau)$ is also a conjunction of atomic formulas, each of which specifies a condition on the value of a state variable at the time of occurrence of the causing trigger.
  - $EA(\tau)$ is either an event or an assignment to a state variable that will take place at T'. Each assignment to a discrete variable needs to be computable at T', i.e., the right-hand side of the assignment should result in a constant value at T'. Any assignment to a continuous variable x should result in a function of time that is continuous at any time $\geq T'$ until further changes. Using λ-notation, the assignment is represented as $x = \lambda t.\ e$ wherein e consists only of constants, time or references to values of state variables at T'. The continuity condition at T' also translates into:

    $$\lim_{t \to T'} e = x(T')$$

    For convenience, we refer to transition rules that affect values of continuous variables as *activity* rules (cf. [13] for a similar categorization).
  - $EC(\tau)$ is a temporal formula to be validated for a finite interval between T and T'. In order that the formula be validated mechanically, we require that it refers only to events and discrete variables. References to continuous variables are permitted only if the references are to values of continuous variables at specified points of time between T and T'.
  - $TT(\tau)$ specifies the value of T' for a given T. We may often omit $TT(\tau)$ by directly substituting its value (in terms of T) in $EA(\tau)$.

[4] See Section 5 for further discussion.

Behaviors of a system in our model are described in the form of finite or infinite *timed situation sequences*

$$S = s_0, s_1, \ldots, s_i, \ldots$$

where s_i can be either an event $e \in E$ or a state $\sigma \in \Sigma$ for all $i \geq 0$. Each s_i is time-stamped with a time t_i and we assume that $\forall i, j : t_j \geq t_i$ if $j > i$. We also assume that s_0 specifies the initial values of the state variables and s_1 is the distinguished event *Start*[5].

Each state s_i in a timed situation sequence actually records the instance of state change (or phase change, cf. [13]). Let s_i and s_j, $i < j$, be two adjacent states (i.e., $s_{i+1}, \ldots, s_{j-1}$ are all events). Then we assume the system state remains the same as s_i between t_i and t_j and then makes a zero-time change to state s_j at t_j. Some clarifications are needed for the handling of continuous variables. A continuous variable v may be assigned a constant or a time-dependent function in s_i. If it is a constant, then v will get the new value at t_i. Otherwise, the value of v for $t > t_i$ will be computed based upon the new function, but its value at t_i will be the one derived from its most recent assignment. When both constant and function assignments are present at t_i, we assume the constant assignment will take place first: v will first get the new constant value at t_i, then behaves like the function onward.

A legal behavior is one wherein state transitions or events take place only when they are caused by one or more of the transition rules. For legality, we also require that time always progresses in any infinite timed-situation sequence (i.e, there is no time t such that all the situations in the sequence have a time-stamp less than t).

Events and state changes are actually dual representations. We can introduce an event variable v_e to encode the occurrence of events: an event e occurs when the assignment $v_e = e$ takes place. In that case, the situation sequence becomes a timed state sequence. On the other hand, we can also name assignments in the effect action portion of each transition rule as events: a state change at the time t_i means all corresponding events must occur at the same time. Therefore, the situation sequence becomes a partial order event sequence. We allow both in our model for the convenience. For example, to model the communicating finite state machine framework, we view sending and receiving messages as events, and state transitions as changes in the interpretation of state variables.

2.1 Examples

In this section we illustrate modelling in TRACE(H) through two examples. The first example is that of a thermostat controlled heater. This example (abstractly) models a simple flow control protocol [9]. The second example models a cat chasing a mouse and is taken from [13]. This example is an abstract model of the producer-consumer problem.

2.1.1 Thermostat Controlled Heater

The behavior of the thermostat is governed by two parameters $HiTh$ and $LoTh$ that denote respectively the higher and lower temperature thresholds at which the thermostat will switch the heater off or on. There may be a delay of δ seconds between the time when the thermostat decides to turn the heater off (or on) and the time when the heater actually goes off (or on). If

[5] We must have a *Start* event that sets the system in motion. A start state is required since we use edge-triggering and since we can detect an edge only if we knew previous values of the state variables.

the heater is turned on (or off) at a time T_0 then its temperature at a later time T is given by $t_0 + k*(T - T_0)$ (or $t_0 + k'*(T - T_0)$), where t_0 is the temperature of the thermostat at time T_0.

To see the relationship between this model and a flow-control protocol, we identify the temperature with the number of packets in the receiver's buffer. The rate of temperature increase is identified with the rate at which the receiver's buffer is filled up, and the rate of temperature decrease with the rate at which the receiver processes packets. When this buffer is close to full, the receiver sends out a message asking the sender to stop and when it is close to empty, it asks the sender to resume. The delay δ models the time between when the receiver sends out a "stop" (or "start") message and the time the sender receives and responds to it.

To model this system, we first introduce the events $SwOn$ and $SwOff$. These events denote the switch being turned on or off. Our initial specification is as follows, with the initial state $IS = \{(temp(0) = LoTh)\}$.

$$\begin{aligned}
temp(T) = HiTh &\rightarrow SwOff(T+\delta) &(1)\\
temp(T) = LoTh &\rightarrow SwOn(T+\delta) &(2)\\
Start(T) &\rightarrow SwOn(T) &(3)\\
SwOn(T) &\rightarrow temp = \lambda t.\, temp(T) + k*(t-T) &(4)\\
SwOff(T) &\rightarrow temp = \lambda t.\, temp(T) + k'*(t-T) &(5)
\end{aligned}$$

The first two transition rules in this specification refer to continuous variables in the trigger and are hence illegal. Therefore we must identify a discrete trigger that causes the temperature to be $HiTh$ (or $LoTh$) in future. It is easily seen that $SwOn(T)$ will cause the temperature to increase to $HiTh$, provided that the current temperature is less than $HiTh$ and that $SwOff$ does not occur till that time. Furthermore, this temperature will be reached after a duration $(HiTh - temp(T))/k$. (Note that $SwOff(T)$ can also lead to temperature reaching $HiTh$ if the current temperature is above $HiTh$. However, this is not significant since the only result of temperature reaching $HiTh$ is to cause the switch to go off, which is already the case.) We therefore introduce the following rules instead of (1) and (2). Herein we use the notation $P(T_1, T_2)$ to denote the temporal formula $\forall t \ [T_1 < t < T_2 \Rightarrow P(t)]$.

$$\begin{aligned}
SwOn(T)/temp(T) < HiTh &\rightarrow SwOff(T'+\delta)/\neg SwOff(T,T') \textbf{ where } T' = (HiTh - temp(T))/k &(6)\\
SwOff(T)/temp(T) > LoTh &\rightarrow SwOn(T'+\delta)/\neg SwOn(T,T') \textbf{ where } T' = (temp(T) - LoTh)/k' &(7)
\end{aligned}$$

Note that references to continuous variables in the trigger has now been eliminated. To achieve this, the programmer explicitly encodes the calculation to identify the time instance when the condition on the continuous variable will become true.

2.1.2 Cat and Mouse

In this example, we model a cat chasing a mouse. At the beginning, the mouse starts running from one end of a room to the other end with a velocity v_m. The cat, which is also resting at the same end, notices the mouse after a time δ and chases it with a velocity v_c. At the end, either the cat wins and catches the mouse or the mouse reaches the safety of a hole at the other end.

We can think of the mouse as a producer process and the cat as a consumer process. The length of the room represents the total number of messages sent by the producer to the consumer. The consumer starts consuming the messages δ units of time after the producer starts producing them. The event of cat catching the mouse represents the situation when the consumer "catches up" with the producer and hence must wait for subsequent messages. Therefore an interesting question here is what δ should be if the consumer does not have to wait. To model this system we introduce the following events:

$M.r$ — Mouse starts running
$C.r$ — Cat starts running
$M.s$ — Mouse is safe, i.e., reaches the other wall
$C.w$ — Cat wins

Letting d, x_m and x_c denote respectively the length of the room and position of the mouse and cat, our initial model of the system is as follows:

$$\begin{aligned}
Start(T) &\rightarrow M.r(T) && (8)\\
M.r(T) &\rightarrow C.r(T+\delta) && (9)\\
x_c(T) = x_m(T) / x_m(T) \neq d &\rightarrow C.w(T) && (10)\\
x_m(T) = d &\rightarrow M.s(T) && (11)\\
M.r(T) &\rightarrow x_m = \lambda t.\ v_m(t-T) && (12)\\
C.r(T) &\rightarrow x_c = \lambda t.\ v_c(t-T) && (13)\\
C.w(T) &\rightarrow x_m = x_m(T) && (14)\\
C.w(T) &\rightarrow x_c = x_c(T) && (15)\\
M.s(T) &\rightarrow x_m = x_m(T) && (16)\\
M.s(T) &\rightarrow x_c = x_c(T) && (17)
\end{aligned}$$

Rule (10) specifies that the cat wins if there is a time when the positions of cat and mouse are the same, provided that the mouse has not already reached the safety of the wall. The last four rules capture the fact that both the cat and the mouse stop running once $C.w$ or $M.s$ happens.

To refine our initial model, we must eliminate references to continuous variables in the triggers of rules (10) and (11). Now suppose that the mouse starts running at T. Then the condition $x_m(T') = x_c(T')$ will be satisfied whenever T' is such that

$$v_c(T' - T - \delta) = v_m(T' - T) \qquad \text{i.e.,} \qquad T' = T + v_m\delta/(v_c - v_m)$$

provided both the cat and the mouse continue to run between T and T'. We also see that they will both be running, unless $M.s$ or $C.w$ happens. Similarly, $x_m = d$ will be satisfied when $T' = T + d/v_m$, provided the mouse is running at T', i.e., $C.w$ has not happened. Therefore we replace rules (10) and (11) by

$$\begin{aligned}
M.r(T) &\rightarrow C.w(T')/\neg M.s(T,T') \ \textbf{where}\ T' = T + v_m\delta/(v_c - v_m) && (18)\\
M.r(T) &\rightarrow M.s(T')/\neg C.w(T,T') \ \textbf{where}\ T' = T + d/v_m && (19)
\end{aligned}$$

3 Simulation Algorithm

In this section we present an algorithm for simulating specifications written in TRACE(H). In the algorithm, *Scheduled* is a list of transitions that have been enabled, but have not yet completed. It is a list of quadruplets of the form $\langle T, T', EA, EC \rangle$, where T and T' (called *cause time* and *effect time*) denote respectively the times when the transition was enabled and when its effect will take place, EA and EC denote the effect and effect condition of this transition. The quadruplets are sorted in increasing order of effect time. We distinguish between discrete quadruplets (whose effect component is either an event or assignment of a value to a state variable) and activity quadruplets (whose effect is an assignment of a function to a continuous state parameter).

1. $T := 0; S := [IS, Start(0)];$ $\quad$ $Scheduled = nil;$
2. For each transition $Start(T)/CC \rightarrow EA(T')/EC$ do
3. Compute T' and insert $\langle 0, T', EA, EC\rangle$ into $Scheduled$ $\quad$ {Schedule EA at T'.}

{The following Main loop exits when no more events or state changes can occur.}

4. While $Scheduled$ is nonempty do

{ The following loop alternately performs discrete effect actions that are scheduled to take place at T and checks for triggers of discrete transitions (or activities). }

5. While there is a discrete quadruplet in $Scheduled$ with effect time T do
6. $TS := nil; TE := nil$ $\quad$ {TS and TE record state changes and events that occur within this loop.}
7. For each discrete quadruplet $\langle T_1, T_2, EA, EC\rangle$ in $Scheduled$ with $T_2 = T$ do
8. If $S \Rightarrow EC$ then
9. Append events in EA to TE $\quad$ {Register the effect actions in the temporary event buffer.}
10. Update TS with assignments in EA $\quad$ {Perform state changes locally in TS.}
11. end
12. append TE and TS to S
13. For each $CT(T)/CC \rightarrow EA(T')/EC \in \mathcal{T}$ such that $triggers(S, CT, T)$ and $L \Rightarrow CC$ do
14. Compute T' and insert $\langle T, T', EA, EC\rangle$ into $Scheduled$ $\quad$ {Schedule EA at T'.}
15. end

{Now perform the activities scheduled for T.}

16. For each activity quadruplet $\langle T_1, T_2, EA, EC\rangle$ in $Scheduled$ with $T_2 = T$ do
17. If $S \Rightarrow EC$ then
18. add $x := \bar{f}(T')$ to S $\quad$ {Store the expression to be evaluated to determine the value of x.}
19. For each $CT(T)/CC \rightarrow EA(T')/EC \in \mathcal{T}$ such that $triggers(S, CT, T)$ and $L \Rightarrow CC$ do
20. Compute T' and insert $\langle T, T', EA, EC\rangle$ into $Scheduled$ $\quad$ {Schedule EA at T'.}
21. $T := first(Scheduled).effecttime$
22. end

Each continuous variable x is represented internally as the expression that determines the current value of x. In particular, if the value of x is given by the activity rule

$$CT/CC \rightarrow x = \lambda t.\ e$$

then we store the expression $\bar{e}$, which is obtained by substituting references to state variables (and possibly T and T') in e with their values in $\sigma(T')$. Whenever the value of a continuous variable is needed at a time $t > T'$, it will be computed from this expression $\bar{e}$ by substituting the value of t.

In the following algorithm, IS denotes the initial state, T the current time and S the timed situation sequence. We also use the function $trigger$ which takes three arguments, viz., a situation sequence S, a trigger condition CT and a time T and returns whether CT will be triggered at T w.r.t. S. The algorithm is based on the event-driven discrete event simulation techniques. Quadruplets corresponding to enabled transitions are put in the queue $Scheduled$. These are removed in increasing order of time and the effects allowed to take place, provided the effect condition is satisfied. In order to make sure that discrete assignments to a continuous variable occur before assignment of continuous functions, all discrete quadruplets are processed (in lines 5–15) before all activity quadruplets. To deal with "zero-time" transitions, our algorithm may need to make several discrete steps to progress through a single time point, similar to *stepping* through fictitious points in the continuous time IC* model. Finally, note that due to continuity requirements, performing the effect of activity quadruplets does not lead to any zero-time transitions. Therefore the effects of activity quadruplets take place outside the loop at lines 5–15.

- This algorithm does not specify how triggers are checked. A simple approach to this would involve checking if the trigger condition was false in the previous state and true in the current state. For efficiency, we may maintain with each discrete variable a list of all triggers in which the variable appears. Whenever the value of any variable changes we may check all the triggers in its list and identify those that just became true. If the trigger contains one or more events then checking for the trigger involves determining whether those events occurred at time T and if so, we need only check that the rest of the trigger condition holds.

- This algorithm implements a partial-order semantics, i.e., at any time, all enabled effects take place concurrently. If an interleaved semantics is desired then the algorithm should be modified.

- If multiple assignments to a state variable are to be detected and flagged as potential conflicts then it should be checked in lines 5–15. We can also detect "glitches" that may occur in the triggers when two state variables that occur in a trigger are both updated at the same time. For instance, suppose that there is a trigger $x = 1 \wedge y = 1$ and that the value of x changes from 1 to zero at the same time as the value of y changes from zero to 1. The above algorithm may not recognize a trigger in such a case. If we need to detect such glitches and flag them, this can also be done in lines 5–15. Finally, if it is desired to detect simultaneous assignment of multiple functions to a continuous variable then it can be done at line 16–18.

4 Reasoning in TRACE(H)

In this section we present proofs of safety properties of the thermostat and the Cat and Mouse. These proofs make use of mainly two techniques called *backward analysis* and *forward analysis*. Forward analysis is an extension of simulation. We explore all possible behaviors of the system and show that the safety or liveness property holds in any configuration that can be reached by the system. In particular, given

$$CT(T)/CC \rightarrow EA(T')/EC$$

and $CT(T)$, it concludes $CC(T) \wedge EC \Rightarrow EA(T')$. The real problem here is that the number of reachable configurations may be very large or infinite and so this analysis may be expensive.

Backward analysis uses the rules in backward manner to conclude that whenever any event or state change occurs, one of its causing triggers must have occurred earlier. In particular, given $EA(T')$ and

$$CT_1(T_1)/CC_1 \rightarrow EA(T')/EC_1, \ldots, CT_n(T_n)/CC_n \rightarrow EA(T')/EC_n$$

it concludes

$$\bigvee_{i=1}^{n} CT_i(T_i) \wedge CC_i(T_i) \wedge EC_i(T_i, T') \wedge (\exists t \neg CT_i(t, T_i))$$

This analysis can be used to prove a safety property by starting with the assumption that the property is violated and then using backward analysis to show that a state wherein the property is violated is unreachable. The search space of backward analysis may be much smaller since it is goal oriented. But this technique is harder than forward analysis since it is typically nontrivial to use a rule backward and because of the branching introduced by $\vee$.

Although forward analysis and backward analysis may not be very powerful by themselves, a combination of the two techniques appears powerful enough to establish many interesting properties of hybrid systems. We now show this by proving certain safety properties of the two examples discussed before using these analyses. In general, however, these techniques have to be combined with others such as induction and case analysis to prove the desired properties.

4.1 Reasoning about Thermostat

In this section we will prove a safety property of the thermostat that asserts that the temperature will never become "too high", say $t_b > HiTh$. A simple approach to proving this would involve associating an event $BadT$ that occurs whenever the temperature reaches t_b. Then we can use backward and forward analysis to show that this event can never occur, or more generally, identify a necessary condition for $BadT$ to occur. We introduce the following causality rule for occurrence of $BadT$.

$$SwOn(T)/temp(T) < t_b \quad \rightarrow \quad BadT(T')/\neg SwOff(T,T') \textbf{ where } T' = T + (t_b - temp(T))/k \quad (20)$$

We first outline a derivation of a necessary condition for $BadT$ to occur using forward and backward analysis. We use rule (20) to derive that $SwOn(T_1)$ must have occurred and $\neg SwOff(T_1,T)$. From $SwOn(T_1)$ we derive $SwOff(T_2)$ where $T_2 = T_1+(HiTh-temp(T_1))/k+\delta$ using rule (6). For this fact not to contradict $\neg SwOff(T_1,T)$, we require that $t_b < HiTh + k\delta$.

In order to formalize the above proof, we first establish the following lemma that allows forward application of rule (6), which has a causing condition that the temperature be less than $HiTh$. This lemma can be established by induction on length of a timed-situation sequence that describes a legal behavior. Since our main intention here is to demonstrate the use of backward and forward analysis, we omit the details of the induction proof here.

Lemma 1 $SwOn(T) \Rightarrow temp(T) < HiTh$

Based on this lemma, we now present a formal derivation of the above mentioned necessary condition. From $BadT(T)$ we derive by backward application of rule (20) that

$$T_1 = T - (t_b - temp(T_1))/k,\ temp(T_1) < t_b,\ SwOn(T_1),\ \neg SwOff(T_1,T) \quad (21)$$

(Here ',' is used to denote conjunction.) Using rule (6) in the forward direction, we get

$$[temp(T_1) < HiTh, \neg SwOff(T_1,T_2)] \Rightarrow SwOff(T_2+\delta) \textbf{ where } T_2 = T_1+(HiTh-temp(T_1))/k (22)$$

The above lemma implies the first of the two conditions in the antecedent of this implication. For establishing the second condition, since $t_b > HiTh$, $T = T_1 + (t_b - temp(T_1)/k > T_2 = T_1 + (HiTh - temp(T_1))/k$. Now note that

$$T_2 < T \wedge \neg SwOff(T_1,T) \Rightarrow \neg SwOff(T_1,T_2)$$

hence we conclude $SwOff(T_2 + \delta)$ from (22). Together with $\neg SwOff(T_1,T_2)$, this yields the condition $T_2+\delta \geq T$. Since $T_2 = (HiTh-temp(T_1))k+T_1$ (by 22) and $T = (t_b-temp(T_1))/k+T_1$ (by 21), the above constraint simplifies to

$$HiTh + k\delta \geq t_b$$

which is what we would have expected.

4.2 Reasoning about Cat and Mouse

In this section we will use backward analysis to obtain the conditions for the cat to win. We start with the event $C.w(T)$ that denotes this. Using rule (18) we get

$$M.r(T_2), \neg M.s(T_2, T) \text{ where } T_2 = T - v_m\delta/(v_c - v_m)$$

Now we use rule (8) to replace $M.r(T_2)$ and obtain

$$Start(T_2), \neg M.s(T_2, T)$$

A necessary condition for $M.s$ not to occur, by rule (19), is

$$\neg M.r(T_2 - d/v_m, T - d/v_m)$$

Using backward analysis with rule (8), we derive $\neg Start(T_2 - d/v_m, T - d/v_m)$. Unifying this with $Start(T_2)$ we obtain

$$(T_2 \leq T_2 - d/v_m) \vee (T_2 \geq T - d/v_m)$$

Since the first of these two condition is not possible, the second part is the desired condition. Substituting $T_2 = T - v_m\delta/(v_c - v_m)$ we get

$$\frac{d}{v_m} \geq \frac{v_m\delta}{v_c - v_m}$$

which is a necessary (and actually a sufficient) condition for the cat to win the race.

5 Discussion

In this section we first discuss the possibilities of relaxing some of the restrictions in our model and the issues involved. We then mention some capabilities that would be needed in order to develop machine-aided verification or reasoning procedures.

5.1 Relaxing the Restrictions in our Model

In this section we bring out the issues in relaxing some of the restrictions we have imposed in our model. In particular, we point out the complications in developing simulation algorithm that arise when the restriction are relaxed.

5.1.1 Continuous Variables in Trigger

Our initial model required that the trigger component of a transition made no reference to continuous variables. This because the truth of a condition on continuous variables may change at any instance of time whereas that of a condition on discrete variables (or events) can change only immediately after an effect (which may be an assignment to discrete variables or occurrence of an event). For this reason, it suffices to check for triggering of transitions immediate after any effect takes place. If we permitted continuous variables in the trigger then this is no longer sufficient. We have to compute the time point when the transition will be triggered, possibly by solving equations that specify the value of the continuous variable as a function of time. These solutions have to be recomputed whenever any one of the functions governing the values of the continuous variables in the trigger is changed. Thus the relaxation comes at the expense of a more complex (and less efficient) simulation procedure. Furthermore, since it may not always be possible to mechanically solve all kinds of equations that arise when computing the time of triggers, the whole process may not be completely mechanizable.

5.1.2 Continuous Variables in Effect Condition

In order that effect condition be mechanically checked, we required that there be no continuous variables in EC (unless the references are to values of continuous variables at specified points of time between T and T'). When continuous variables are permitted, we need to be able to check conditions such as $x(T, T') < 10$ where x is a continuous variable. Such a condition is difficult to verify mechanically. An even more complex case concerns effect conditions of the form $x = y$ where both x and y are continuous parameters. Checking this condition amounts to establishing that the functions governing x and y are the same in the interval between T and T' and may not be mechanizable at all.

5.1.3 Time Intervals

Another possible extension is to specify that the effect may occur at any point of time within a specified interval. With this approach, a simulation of the system may yield different results depending upon the time when an effect occurs within the specified interval. If we are interested in obtaining one such timed sequence then our simulation algorithm can be readily modified as follows. Before an effect is put in the *Schedule* queue, its time stamp is chosen to be a random value within the interval of time during which the effect is specified to occur.

The problem of computing all possible timed sequences is harder. Observe that there can be uncountably many timed sequences if we use the current interpretation of a timed sequence: the sequences will differ in the time when an event (or state variable assignment) occurred. So we need to extend the notion of timed sequences so that each event or state change is time-stamped with an interval rather than a specific point of time.

Transition rules will specify constraints on the time interval during which each event or variable assignment occurs. Often we may need to determine whether a particular event may occur or not depending upon the value of a continuous variable. In each such case, the number of sequences double: One sequence in which the event occurs and one in which it does not. Solving these constraints to eliminate sequences that contain any unsatisfiable set of constraints can be quite hard. Approaches such as those mentioned in [11] or [5] may be very useful here.

5.2 Capabilities Required for Reasoning

Constraint solving capability is central to both forward and backward reasoning. In the forward analysis, it is useful for performing symbolic simulation wherein we only know the range of a state parameter, not its exact value. In backward analysis, it is useful so that we can accumulate information such as $x < 5$ where x is a state parameter. This may later be used to eliminate another transition that requires x to be greater than 5, thereby decreasing the search-space. To deal with continuous variables, the constraint solver must be able to solve algebraic equations. Another capability is to be able to invert functions, i.e., given a function $f(x)$, identify another function g such that $f(g(x)) = g(f(x)) = x$. This problem of function inversion becomes more complex if f takes more than a single argument or if f is a one-to-many mapping.

Finally, we would like to remark that in some cases, it may be possible to eliminate references to continuous variables for the purpose of reasoning. The cat and mouse example is one such case wherein we made use of only discrete transition rules to establish the proof. This is possible mainly because continuous variables were not permitted in the trigger.

Acknowledgements

We would like to thank Jane Cameron and Hugo Velthuijsen for their insightful comments and helpful suggestions on earlier drafts of this paper. The authors are also grateful to anonymous reviewers for many useful comments.

References

[1] S. Aggarwal, B. Gopinath and P. Uppaluru, Specification and Composition of Relativized Models of Real-Time Systems, *Third International Conference on CAD/CAM, Robotics, and Factories of the Future, Springer-Verlag, 1989.*

[2] S. Aggarwal and R.P. Kurshan, Modeling elapsed time in protocol specification, *PSTV III, North Holland, 1983.*

[3] R. Alur and T.A. Henzinger, A really temporal logic, *FOCS 1989.*

[4] Bellcore, Synchronous Optical Network (SONET) Transport Systems: Common Generic Criteria, *Bellcore TA-NWT-000253, 1990.*

[5] B. Berthomieu and M. Diaz, Modeling and verification of time dependent systems using time petri nets, *IEEE Transactions on Software Engineering, March 1991.*

[6] E.J. Cameron, D. Cohen, L. Ness, and H. Srinidhi, L.0: A language for modeling and prototyping communications software, *Proceedings of the Third International Conference on Formal Description Techniques, November 1990.*

[7] J. Cameron and Y.-J. Lin, A Real-Time Transition Model for Analyzing the Behavioral Compatibility of Telecommunication Systems, *SIGSOFT '91 – Software for Critical Systems, 1991.*

[8] F.S. Dworak, T.F. Bowen, C.H. Chow, G.E. Herman, N. Griffeth, and Y.-J. Lin, Feature interaction problem in telecommunication systems, *Proceedings of the Seventh International Conference on Software Engineering for Telecommunication Switching Systems, July 1989.*

[9] M. Gerla and L. Kleinrock, Flow Control: A Comparative Survey, *IEEE Transactions on Communications, April 1980.*

[10] E. Harel, O. Lichtenstein, and A. Pnueli, Explicit clock temporal logic, *LICS 1990.*

[11] Y.-J. Lin and G. Wuu, A Constrained Approach for Temporal Intervals in the Analysis of Timed Transitions, *PSTV XI, 1991.*

[12] N. Lynch and H. Attiya, Using mappings to prove timing properties, *PODC 1990.*

[13] O. Maler, Z. Manna and A. Pnueli, From Timed to Hybrid Systems, *REX 91.*

[14] K. Marzullo, F.B. Schneider and N. Budhiraja, Derivation of sequential, real-time, process-control programs, *Foundations of Real-Time Computing: Formal Specification and Methods.*

[15] A. R. Modarressi and R. A. Skoog, Signaling System No. 7: A Tutorial, *IEEE Communications, July 1990.*

[16] S. Narain, An Axiomatic Basis for General Discrete Event Modelling, *Bellcore, 1991.*

[17] S. Narain, An approach to Reasoning about Hybrid Systems, *International Journal of General Systems, 1991.*

[18] J.S. Ostroff, Deciding properties of timed transition models, *IEEE Transactions on Parallel and Distributed Systems, April 1990.*

[19] B.P. Zeigler, *Multifaceted Modelling and Discrete Event Simulation, Academic Press, 1984.*