Modeling and Analysis of Hybrid Control Systems

An Elevator Case Study

Ying Zhang[1] and Alan K. Mackworth[2]

[1] Xerox Palo Alto Research Center,
3333 Coyote Hill Road,
Palo Alto, CA, USA,
yzhang@parc.xerox.com

[2] Department of Computer Science,
University of British Columbia,
Vancouver, B.C., Canada,
mack@cs.ubc.ca

Abstract. We propose a formal approach to the modeling and analysis of hybrid control systems. The approach consists of the interleaved phases of hybrid dynamic system modeling, requirements specification, hybrid control design and overall behavior verification. We have developed Constraint Nets as a semantic model for hybrid dynamic systems. Using this model, continuous, discrete and event-driven components of a dynamic system can be represented uniformly. We have developed timed ∀-automata as a requirements specification language for dynamic behaviors. Using this language, many important properties of a dynamic system, such as safety, stability, reachability and real-time response can be formally stated. We have also proposed a verification method for checking whether a constraint net model satisfies a timed ∀-automaton specification. The method uses the induction principle and generalizes both Liapunov stability analysis for dynamic systems and monotonicity of well-foundedness in discrete-event systems. The power of these techniques is demonstrated with a simple elevator system. An elevator system is a typical hybrid system with continuous motion following Newtonian dynamics and discrete event control responding to users' request. We model the complete elevator system using Constraint Nets and verify the overall behavior of the system against the requirements specification in timed ∀-automata.

1 Introduction

Hybrid dynamic systems are systems that consist of coupled discrete and continuous components. Any electromechanical system with a computerized controller is a hybrid system in general. In the past, the modeling and analysis of a hybrid system have been done separately for its discrete and continuous components. The overall system is designed in a rather empirical fashion. Since computer-aided control is becoming more and more significant in modern system design practice, we face a major challenge: the development of intelligent, reliable, robust and safe computer-controlled systems. The foundation for modeling and analysis of hybrid systems must be established.

1.1 Our approach

We advocate a formal modeling and analysis framework for the development of hybrid control systems. The framework consists of the interleaved phases of hybrid system modeling, requirements specification, hybrid control design and behavior verification. System modeling can also be called system specification, which precisely defines how a system is structured in terms of its components and inter-connections, and how each of its component works. Requirements specification expresses global properties of a system such as safety (i.e., a bad state will never be reached), stability (i.e., a final state will be reached), reachability (i.e., a state is reachable) and real-time response (i.e., a state will be reached in bounded time). Control system design takes a system model (including the plant and its envoronment), which in general is continuous or hybrid, and its requirements specification, produces a control system, which in general is discrete or hybrid. Behavior verification ensures that the behavior of the coupled overall system (i.e. plant+environment+control) satisfies the specified requirements.

For hybrid system modeling, we have developed the Constraint Net (CN) model [37, 34]. CN provides a formal syntax and semantics of a hybrid dynamic system so that its continuous and discrete (fixed sampling time or event-driven) components can be modeled uniformly. CN provides aggregation operators so that a complex system can be modeled hierarchically. Therefore, a system and its control can be modeled individually in CN and then be composed to generate an overall system model. The overall hybrid system is defined mathematically so that it can be analyzed without ambiguity. CN also supports multiple levels of abstraction, based on abstract algebra and topology; therefore, a system can be modeled and analyzed at different levels of detail.

For requirements specification, we have developed timed $\forall$-automata [36, 35]. Timed $\forall$-automata are essentially finite automata modified from $\forall$-automata [24] by generalizing to continuous time and adding timing constraints; yet they are powerful enough to specify global system properties of sequential and timed behaviors of hybrid dynamic systems, such as safety, stability, reachability and real-time response.

For control design, we have proposed constraint-based techniques [38, 39]. In this paper, we advocate a two-level design architecture. A hybrid control system can be developed in a two-level structure, with the lower level as a continuous component guided by an analog control law and the higher level as a discrete logic component driven by events from the lower level or from the environment.

For behavior verification, we have proposed a formal model checking method [35, 34]. The method uses the induction principle and combines Liapunov stability analysis for dynamic systems and monotonicity of well-foundedness in discrete-event systems. This verification method can be semi-automated for discrete time systems and further automated for finite domain systems.

1.2 Related work

Much work has been done on the modeling and analysis of hybrid control systems [10]. Various formal models/languages have been developed for this purpose.

Roughly speaking, formal models/languages for modeling and analysis can be characterized as belonging to one of the four categories: (1) state transition models, (2) algebraic processes, (3) block diagrams/nets/dataflow/equations, and (4) temporal logics/ω-languages.

For example, Phase Transition Systems [23, 26] are a typical state transition model, where computation consist of alternating phases of discrete transitions and continuous activities. Similarly, Nerode and Kohn's Hybrid Automata [25] consist of a digital control automaton and a plant automaton. The plant automaton can be modeled as a state transition system over intervals. Alur et al. [1] develop a model for hybrid systems which generalizes timed automata [3, 2]. Lynch's group at MIT has been using Timed Automata [22] for modeling and verification of automated transit systems [21].

In the formalism of algebraic processes, Gupta et al. [11] proposes Hybrid cc, a generalization (or a new member) of the cc family [28], for modeling systems with timers and continuous activities.

The Constraint Net model we have developed belongs to the category of block diagrams / nets / dataflow / equations. In this formalism, a system is a network of processes, processes are represented by blocks, and interactions between processes are represented by connections between blocks. All processes are considered to be running in parallel and data flow through these processes where operations on the data are performed. An equational representation of the model can also be obtained, with each process corresponding to an equation. For example, SIGNAL [6] is a typical language in this formalism. Conventional continuous or discrete control structures are best modeled in block diagrams. Krogh [17] develops condition/event signal interfaces for block diagrams, which extends block diagrams with logic operators. Brockett [8] studies motion control systems with an equational event-driven model. Some commercial products for control simulation are also belong to this category, such as Simulink [14] and SystemBuild [13].

Timed $\forall$-automata we have developed are closely related to temporal logics/ω-languages. Many extensions of temporal logics/regular languages to timed and/or hybrid systems have been proposed. For example, Lamport [18] develops TLA$^+$, an extension to TLA (Temporal Logic of Action), for modeling hybrid systems. Alur and Henzinger [4] propose a really temporal logic with metric time. Duration Calculus [33], a generalization of interval temporal logics, has also been applied to the formal design of hybrid systems. From the formal automata/language point of view, TBA (Timed Buchi Automata) [2] are developed to generate timed ω-languages, and its properties and decision procedures are also studied.

Related work is also been done in software engineering for safety issues. For example, the Requirements State Machine (RSM) [16] provides semantic analysis of real-time process-control software requirements, and the Requirements State Machine Language (RSML) [19] is then designed to write requirements specifications for an industrial aircraft collision avoidance system. Time Petri Nets are applied to modeling and verification of time dependent systems [7].

From a methodological point of view, there are two schools: one uses a single general model/language (e.g. state transition systems or temporal logics) for both system modeling and requirements specification; the other proposes different formalisms for modeling and specification: e.g., state transition systems for modeling and temporal logics for specification. Our approach here belongs to the second school.

1.3 Our contributions

Our contributions to the modeling and analysis of hybrid control systems are three-fold.

First of all, we develop Constraint Nets for modeling complex structured hybrid dynamic systems. Like all the net-based models, CN is modular and hierarchical with a formal syntax and semantics for simulation. Unlike all the other existing models, CN is developed on abstract time and domain structures. Instead of combining models of discrete and continuous dynamic systems, we start from a general model of dynamic systems, of which both discrete and continuous systems are special cases. As a result, CN provides a powerful and formal model for complex hybrid dynamic systems.

Secondly, we develop timed $\forall$-automata for specifying simple global properties of hybrid dynamic systems. Unlike most existing timed automata, timed $\forall$-automata have finite state without clock or data variables. Timed $\forall$-automata are similar to propositional temporal logics. However, timed $\forall$-automata are defined on abstract time structures which can be either discrete or dense. With notions of both local and global timeout, timed $\forall$-automata can represent real-time responses as well. As a result, timed $\forall$-automata provide a simple and useful specification language for timed behaviors.

Thirdly, we develop a formal behavior verification method that uses the induction principle and generalizes both Liapunov stability analysis for dynamic systems and monotonicity of well-foundedness in discrete-event systems.

1.4 A simple elevator system

We use a simple elevator system here to demonstrate the use of our approach. Elevator systems have been used in various communities as benchmark examples of methodologies for software engineering and real-time systems [15, 27, 12, 5, 9].

However, most previous examples of elevator systems focus on discrete-event structures. In this paper, we model an elevator system as a hybrid system with continuous motion following Newtonian dynamics and discrete control responding to users' request. In particular, we show how the coupling of the discrete and continuous components can affect the behavior of the overall system.

A simple elevator system for an n-floor building consists of one elevator. Inside the elevator there is a board with *n floor buttons*, each associated with one floor. Outside the elevator there are two *direction buttons* for service call on each floor, except the first floor and the top floor where only one button is needed (see Figure 1). Any button can be pushed at any time. After being

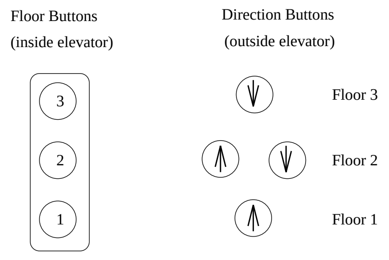


Fig. 1. A simple 3-floor elevator

pushed, a floor button will be on until the elevator stops at the floor, and a direction button will be on until the elevator stops at the floor and is going to move in the same direction. (Note that a more complex elevator would have open and close door buttons, and alarm or emergency buttons which, for simplicity, we do not model in this paper.)

Using constraint nets, we first model this elevator system at two levels of detail. At the lower level, the dynamics of continuous motion is modeled; and at the higher level, the abstraction of the desired discrete system is represented. Then we model the overall hybrid system, the elevator with control in black boxes, in Constraint Nets.

A well-designed elevator system should satisfy the property of real-time response, i.e., any request will be served within some bounded time. Using timed ∀-automata, we explore the meaning of such requirements and specify them precisely.

Then we design a continuous controller and a discrete controller by analyzing the requirements.

Finally, we explore approaches to verifying the behavior of a dynamic system against its requirements specification. For the elevator case study, we propose and answer the question: is the elevator system well-designed?

1.5 Outline of this paper

The rest of this paper is organized as follows. Section 2 briefly introduces Constraint Nets, and demonstrates constraint net modeling using the elevator example. Section 3 presents timed ∀-automata and gives the requirements specification of the real-time behavior of the elevator system. Section 4 designs the control system for the elevator by analyzing the requirements specification. Section 5 develops a model checking method which determines whether the constraint net model of the elevator system satisfies the timed ∀-automaton specification of the desired behavior. Section 6 draws some conclusions.

2 System Modeling in Constraint Nets

A complex dynamic system should be modeled in terms of its components and interconnections, at multiple levels of abstraction. We model the elevator system this way using Constraint Nets.

2.1 Concepts of dynamic systems

We start by introducing some general concepts of dynamic systems which we use later. For formal definitions of these concepts, the reader is referred to [34].

- *Time* $\mathcal{T}$ is a linearly ordered set with a minimal element as the start time point, for example, the set of natural numbers or the set of non-negative real numbers with the arithmetic ordering. The former is called *discrete time* and the latter is called *continuous time*.
- A *trace* $v^A_{\mathcal{T}} : \mathcal{T} \rightarrow A$ is a function from time $\mathcal{T}$ to a domain A of values.
- An *event trace* $e_{\mathcal{T}} : \mathcal{T} \rightarrow B$ is a special type of trace whose domain B is boolean. An *event* in an event trace is a transition from 0 to 1 or from 1 to 0. An event trace characterizes some *event-based time* where the set of events in the trace is the time set. The time domain of an event trace is called the *reference time* of the event-based time.
- A *transduction* $F : \mathcal{V}_I \rightarrow \mathcal{V}_O$ is a mapping from a tuple of input traces to a tuple of output traces, which satisfies the causality constraint between the inputs and the outputs, i.e., the output values at any time depend only on the inputs up to that time. For instance, a state automaton with an initial state defines a transduction on discrete time; a temporal integration with a given initial value is a typical transduction on continuous time. Just as nullary functions represent constants, nullary transductions represent traces.
- A transduction F is called a *transliteration* if it is a pointwise extension of a function f, i.e. $F(v)(t) = f(v(t))$. Intuitively, a transliteration is a transformational process without memory (internal state), such as a combinational circuit.
- States or memories are introduced by delays. A *delay* transduction is a sequential process where the output value at any time is the input value at a previous time, i.e., $\delta(v)(t) = v(t - \delta)$. Normally, *unit* delays are for discrete time and *transport* delays are for continuous time.
- The linkages between discrete and continuous components are modeled by event-driven transductions. An *event-driven transduction* is a transduction augmented with an extra input which is an event trace; the event-driven transduction operates at every event and its output value holds from each event to the next.

2.2 Constraint net model

The Constraint Net model is built upon these general concepts of dynamic systems. It is a net/dataflow- or equation-based model with formal syntax and semantics. It also provides modular structures with composition hierarchy.

Syntax and semantics A constraint net consists of a finite set of locations, a finite set of transductions and a finite set of connections. Intuitively, locations represent states, memories, variables or communication channels; transductions represent processes, operating according to a global reference time or activated by external events; and connections represent the interaction structures or data flows of the modeled system. Formally, a *constraint net* is a triple $CN = \langle Lc, Td, Cn \rangle$, where Lc is a finite set of *locations*, Td is a finite set of labels of *transductions*, each with an *output port* and a set of *input ports*, Cn is a set of *connections* between locations and ports, with the following restrictions: (1) there is at most one output port connected to each location, (2) each port of a transduction connects to a unique location and (3) no location is isolated.

A location is an *output* of the constraint net if it is connected to the output of some transduction; otherwise it is an *input*. A constraint net is *open* if there is an input location; otherwise it is *closed*.

A constraint net represents a set of equations, with locations as variables and transductions as functions. The *semantics* of the constraint net, with each location denoting a trace, is the least solution of the set of equations. The semantics is defined on abstract data types and abstract reference time which can be discrete or continuous. For detailed formal semantics, the reader is referred to [37, 34].

Graphically, a constraint net is depicted by a bipartite graph where locations are depicted by circles, transductions by rectangular blocks and connections by arcs.

For example, the graph in Figure 2, where f is a transliteration and δ is a unit delay, depicts a state transition system. This open net, with discrete time, can be represented by the equation: $s(n) = f(u(n-1), s(n-1)), s(0) = s_0$. We can also simply write $s' = f(u, s), s(0) = s_0$ where s' denotes the next state of s.

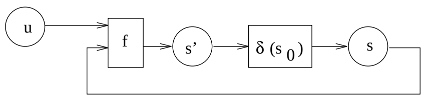


Fig. 2. The constraint net representing a state transition system

Similarly, the net depicted by the graph in Figure 3, with continuous time, models the differential equation $\dot{s} = f(u, s), s(0) = s_0$.

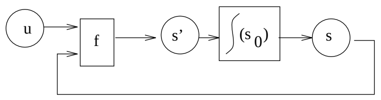


Fig. 3. The constraint net representing a differential equation

Modules and composition A complex system is generally composed of multiple components. We define a *module* as a constraint net with a set of locations as its interface. A module is depicted by a box with rounded corners. For example, the state transition system in Figure 2 is grouped to a module in Figure 4 where u and s are the input and output interface, respectively. The double circles depict its interface.

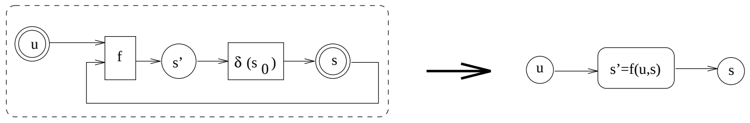


Fig. 4. A module and its representation

A constraint net can be composed hierarchically using modular and aggregation operators on modules. There are three basic operations — union, coalescence and hiding — that can be applied to obtain a new module from existing ones. The *union* operation generates a new module by putting two modules side by side. The *coalescence* operation coalesces two locations in the interface of a module into one, with the restriction that at least one of these two locations is an input location. The *hiding* operation deletes a location from the interface. We can model non-determinism with hidden inputs and model internal states with hidden outputs.

A hybrid system can be modeled by a composition of continuous components with event-driven discrete components. For example, Figure 5 depicts a hybrid system consisting of a state transition system and a differential equation. The event input port of the event-driven state transition is illustrated by a small circle. Events can be generated by clock signals from outside of the net as shown in (a) or by outputs within the net itself as shown in (b) (where events at e are generated by module `event` with input from y).

2.3 Models of the elevator system

An elevator system is a typical hybrid system with continuous motion following Newton's dynamics and event-based control responding to users' request. In the

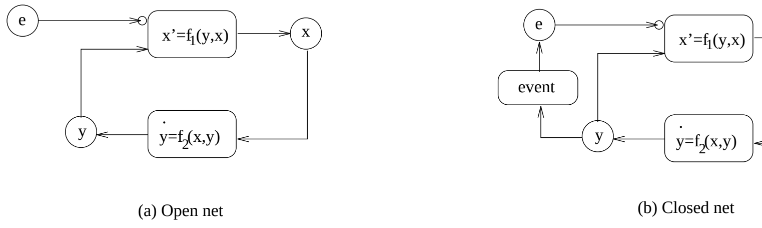


Fig. 5. Hybrid systems in Constraint Nets

rest of this section, we will model the elevator system in Constraint Nets. The controllers are left as black boxes and will be designed and modeled in Section 4, after the section on requirements specification.

Continuous model We model the elevator body by a second order differential equation following Newton's Second Law

$$F - K\dot{h} = \ddot{h} \tag{1}$$

where F is the motor force, K is the coefficient of friction and h is the height of the elevator (Figure 6). Here we assume that the mass of the body is 1 since it

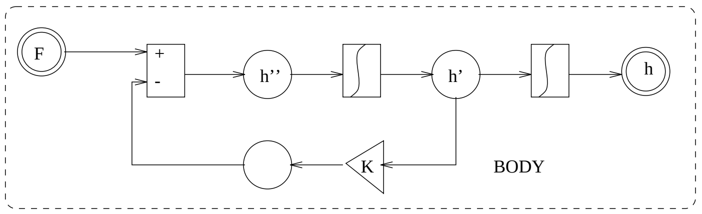


Fig. 6. The BODY module

can be scaled by F and K. We also ignore gravity since it can be added to F to compensate its effect.

Let the separation between floors be H. Given the current height h, the current floor number can be obtained as

$$f = [h/H] + 1 \tag{2}$$

where $[x]$ denotes the integer closest to x, and the distance to the nearest floor is

$$d_s = h - (f - 1)H. \tag{3}$$

Furthermore, we say that the elevator is in a home position if

$$e_h : |d_s| \leq \epsilon \tag{4}$$

for some $\epsilon > 0$. In real life, f, d_s and the home event e_h can be either sensed directly or calculated from h.

The continuous component of the elevator system is depicted in Figure 7 where `Com` is the higher level command which can be $1, -1$ and 0, denoting `up`, `down` and `stop`, respectively, and `CONTROL0` is an analog controller that generates the signal that determines the force to drive the elevator body, `BODY` is the module in Figure 6, `FLOOR` is Equation 2 and `HOME` is Condition 4.

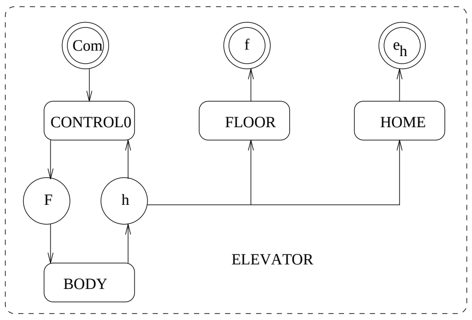


Fig. 7. The continuous components of the elevator system

Discrete model For an n-floor elevator system, the relationship between the command `Com` and the current floor f modeled in the continuous component of the elevator system can be abstracted by a state transition system with state transition function: $f' = NextFloor(f, Com)$ where

$$NextFloor(f, Com) = \begin{cases} \min(f+1, n) & \text{if } Com = 1 \\ \max(f-1, 1) & \text{if } Com = -1 \\ f & \text{if } Com = 0 \end{cases}$$

provided that `CONTROL0` works correctly.

We model push buttons as an array of flip-flops. A button will be set to 1 when it is pushed by a user and be reset to 0 when the request is served.

Formally, let Ub, Db and Fb denote up, down and floor buttons, respectively. For an n-floor elevator, $Ub, Db, Fb \in \{0,1\}^n$ are boolean vectors of n-elements with $Ub(n) = 0$ and $Db(1) = 0$. The request state of a push button is determined by two factors: the users' input and the reset signal when the request has been served. Let the users' inputs, the states of the push buttons and the reset signals be $\langle Ub_i, Db_i, Fb_i \rangle$, $\langle Ub_s, Db_s, Fb_s \rangle$ and $\langle Ub_r, Db_r, Fb_r \rangle$, respectively. If a button is pushed by a user, the state of the button is set (to 1); or, if the request has been served, the state is reset (to 0); otherwise, the state is unchanged. The state transition function of the flip-flop is a logical expression:

$$\begin{aligned} b'_s &= FlipFlop(b_i, b_r, b_s) \\ &= b_i \vee (\neg b_r \wedge b_s) \end{aligned}$$

Note that this flip-flop has higher priority for set than for reset, i.e., if a user pushes a button for service while the elevator is just finishing the service, the elevator should still consider the new request at that time.

Let s be the serving state of the elevator which can be `up, down` or `idle`. The reset signal b_r indicates which requests have been served, formally:

$$b_r = ResetSignal(f, Com, s)$$

such that $\forall 1 \leq k \leq n$,

$$\begin{aligned} Ub_r(k) &= (f = k) \wedge (Com = 0) \wedge (s = up) \\ Db_r(k) &= (f = k) \wedge (Com = 0) \wedge (s = down) \\ Fb_r(k) &= (f = k) \wedge (Com = 0) \wedge (s \neq idle). \end{aligned}$$

Summarizing, the push button module is shown in Figure 8.

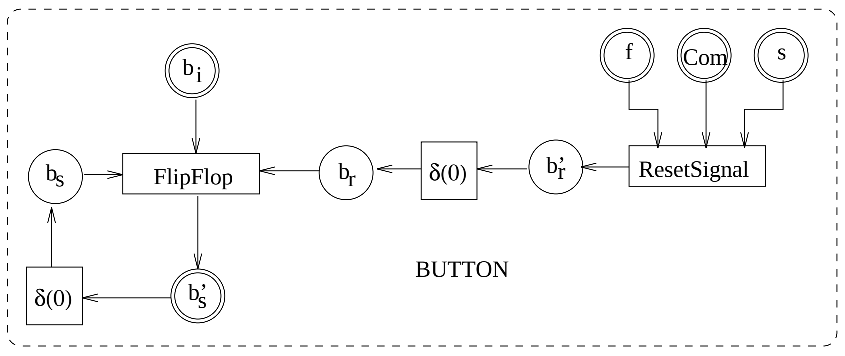


Fig. 8. The BUTTON module

For a continuous reference time, we assume that the event-driven time (which could be generated by a built-in clock) of the push button module has much higher frequency than users' inputs.

Hybrid model Let **CONTROL1** be a discrete control with the current floor number f and the current button states b_r as inputs and with command **Com** and serving state s as outputs. **CONTROL1** is driven by event e which is the "event-or" (for the concepts of event logics, the reader is referred to [30]) of the following three events: (1) a user pushes a button at the elevator's idle state ($s = idle \wedge \bigvee_{1 \leq k \leq n}(Ub_s(k) \vee Db_s(k) \vee Fb_s(k))$), (2) the elevator comes to a home position ($|d_s| \leq \epsilon$) and (3) a request has been served ($Com = 0 \wedge \neg(Fb_s(f) \vee (Ub_s(f) \wedge s = up) \vee (Db_s(f) \wedge s = down)$) for certain time. If it takes τ seconds to serve a request, a transport delay of τ will be used.

Finally, the hybrid model of the elevator and its control is shown in Figure 9, where **EVENT** implements the event logic that produces the events for triggering the discrete control, **BUTTON** is Figure 8 and **ELEVATOR** is Figure 7.

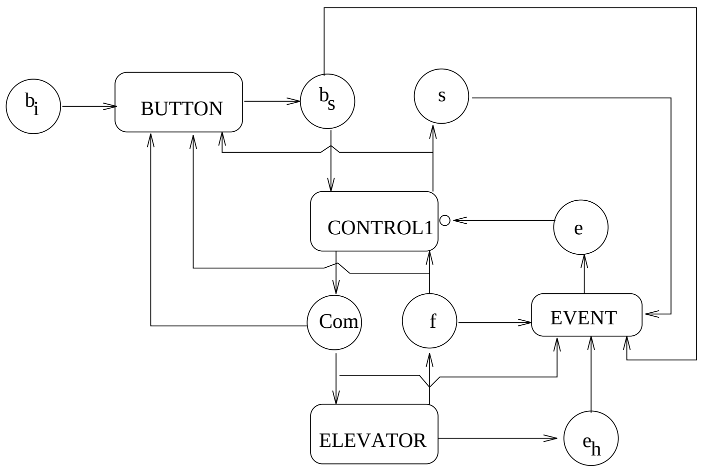


Fig. 9. The hybrid model of the elevator system

3 Requirements Specification in Timed ∀-Automata

While modeling focuses on the underlying structure of a system, the organization and coordination of components or subsystems, the overall behavior of the modeled system is not explicitly expressed. However, for many situations, it is important to specify some global properties and guarantee that these properties hold in the system being designed. For example, a well-designed elevator system should service any request within some bounded waiting time. Requirements specification in timed ∀-automata provides a formal method for this purpose.

3.1 Timed $\forall$-automata

Discrete $\forall$-automata are non-deterministic finite state automata over infinite sequences. These automata were originally proposed as a formalism for the specification and verification of temporal properties of concurrent programs [24]. We augment discrete $\forall$-automata to timed $\forall$-automata by generalizing time from discrete to continuous and by specifying time constraints on automaton-states.

A $\forall$-*automaton* $\mathcal{A}$ is a quintuple $\langle Q, R, S, e, c\rangle$ where Q is a finite set of *automaton-states*, $R \subseteq Q$ is a set of *recurrent states* and $S \subseteq Q$ is a set of *stable states*. With each $q \in Q$, we associate an assertion $e(q)$, which characterizes the *entry condition* under which the automaton may start its activity in q. With each pair $q, q' \in Q$, we associate an assertion $c(q, q')$, which characterizes the *transition condition* under which the automaton may move from q to q'. R and S are the generalization of *accepting* states to the case of infinite inputs. We denote by $B = Q - (R \cup S)$ the set of *non-accepting (bad)* states.

A $\forall$-automaton can be depicted by a labeled directed graph where automaton-states are depicted by nodes and transition relations by arcs. Furthermore, some automaton-states are marked by a small arrow, an *entry arc*, pointing to it. Each recurrent state is depicted by a diamond inscribed within a circle. Each stable state is depicted by a square inscribed within a circle. Nodes and arcs are labeled by assertions. A node or an arc that is left unlabeled is considered to be labeled with *true*. The labels define the entry conditions and the transition conditions of the associated automaton as follows.

- Let $q \in Q$ be a node in the diagram corresponding to an automaton-state. If q is labeled by ψ and the entry arc is labeled by φ, the entry condition $e(q)$ is given by $e(q) = \varphi \wedge \psi$. If there is no entry arc, then $e(q) = false$.
- Let q, q' be two nodes in the diagram corresponding to automaton-states. If q' is labeled by ψ, and arcs from q to q' are labeled by $\varphi_i, i = 1 \cdots n$, the transition condition $c(q, q')$ is given by $c(q, q') = (\varphi_1 \vee \cdots \vee \varphi_n) \wedge \psi$. If there is no arc from q to q', $c(q, q') = false$.

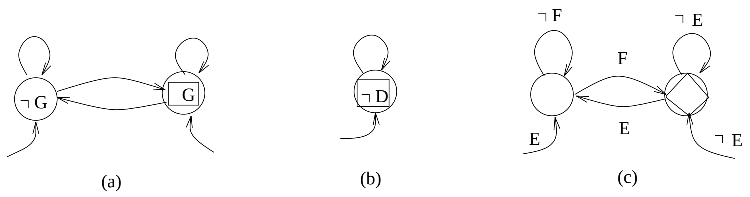


Fig. 10. $\forall$-automata: (a) reachability (b) safety (c) bounded response

Some examples of $\forall$-automata are shown in Figure 10: (a) states that the system should finally satisfy G; (b) states that the system should never satisfy D and

(c) states that whenever the system satisfies E, it will satisfy F in some bounded time.

The formal semantics of $\forall$-automata is defined as follows. Let A be a domain of values. An assertion α on A corresponds to a subset $V(\alpha)$ of A. A value $a \in A$ satisfies an assertion α on A, written $a \models \alpha$, iff $a \in V(\alpha)$. Let $\mathcal{T}$ be the time domain and $v : \mathcal{T} \rightarrow A$ be a trace. Given a $\forall$-automaton $\mathcal{A}$ as $\langle Q, R, S, e, c\rangle$, a *run* of $\mathcal{A}$ over v is a mapping $r : \mathcal{T} \rightarrow Q$ such that the following two conditions are satisfied:

1. *Initiality*: Let $\mathbf{0} \in \mathcal{T}$ be the start time point, $v(\mathbf{0}) \models e(r(\mathbf{0}))$; and
2. *Consecution*: If $\mathcal{T}$ is discrete time, then for all $t > \mathbf{0}$, $v(t) \models c(r(pre(t)), r(t))$, where $pre(t)$ is the previous time point of t. If $\mathcal{T}$ is continuous time, the following two conditions must be satisfied:
   (a) *Inductivity*: $\forall t > \mathbf{0}, \exists q \in Q, t' < t, \forall t'', t' \leq t'' < t, r(t'') = q$ and $v(t) \models c(r(t''), r(t))$ and
   (b) *Continuity*: $\forall t, \exists q \in Q, t' > t, \forall t'', t < t'' < t', r(t'') = q$ and $v(t'') \models c(r(t), r(t''))$.

   These two conditions are derived from continuous induction principle. Condition (a) corresponds to a right-closed transition and condition (b) corresponds to a left-closed transition.

If r is a run, let $Inf(r)$ be the set of automaton-states appearing infinitely many times in r, i.e., $Inf(r) = \{q | \forall t \exists t_0 \geq t, r(t_0) = q\}$. A run r is defined to be *accepting* iff:

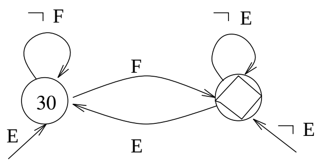
1. $Inf(r) \cap R \neq \emptyset$, i.e., *some* of the states appearing infinitely many times in r belong to R, or
2. $Inf(r) \subseteq S$, i.e., *all* the states appearing infinitely many times in r belong to S.

A $\forall$-automaton $\mathcal{A}$ *accepts* a trace v, written $v \models \mathcal{A}$, iff *all* possible runs of $\mathcal{A}$ over v are accepting.

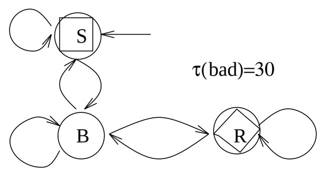
For example, Figure 10(a) accepts the trace $x(t) = Ce^{-t}$ for $G =_{def} |x| < \epsilon$. Figure 10(b) accepts the trace $x(t) = \sin(t)$ for $D =_{def} |x| > 1$. Figure 10(c) accepts the trace $x(t) = sin(t)$ for $E =_{def} x \geq 0$ and $F =_{def} x < 0$.

Timed $\forall$-automata are $\forall$-automata augmented with timed automaton-states and time bounds. Let $\mathcal{R}^+$ be the set of non-negative real numbers. A *timed* $\forall$-*automaton* $\mathcal{TA}$ is a triple $\langle \mathcal{A}, T, \tau\rangle$ where $\mathcal{A} = \langle Q, R, S, e, c\rangle$ is a $\forall$-automaton, $T \subseteq Q$ is a set of *timed* automaton-states and $\tau : T \cup \{bad\} \rightarrow \mathcal{R}^+ \cup \{\infty\}$ is a *time function*. A $\forall$-automaton is a special timed $\forall$-automaton with $T = \emptyset$ and $\tau(bad) = \infty$. Graphically, a T-state is denoted by a nonnegative real number indicating its time bound. Figure 11 shows two examples of timed $\forall$-automata.

The formal semantics of timed $\forall$-automata is defined as follows. Let $I \subseteq \mathcal{T}$ be a time interval and $\mu(I) \in \mathcal{R}^+$ be the real-time measurement of I. Let $r : \mathcal{T} \rightarrow Q$ be a run of $\mathcal{A}$. For any $P \subset Q$, let $Sg(P)$ be the set of consecutive P-state segments of r, i.e., $r_{|I} \in Sg(P)$ for some interval I iff $\forall t \in I$, $r(t) \in P$. A run r *satisfies the time constraints* iff



(a) Local timing



(b) Global timing

Fig. 11. Examples of timed automata

1. (local time constraint) for any $q \in T$ and any interval I of $\mathcal{T}$, if $r_{|I} \in Sg(\{q\})$ then $\mu(I) \leq \tau(q)$ and
2. (global time constraint) let $B = Q - (R \cup S)$ and $\chi_B : Q \rightarrow \{0,1\}$ be the characteristic function for set B; for any interval I of $\mathcal{T}$, if $r_{|I} \in Sg(B \cup S)$ then $\int_I \chi_B(r(t))dt \leq \tau(bad)$.

Condition 1 says if τ is the local timing bound for q, then the time duration of any segment of q's must be no more than τ. Condition 2 says if τ is the global timing bound, then the total time on bad states must be no more than τ.

Let $v : \mathcal{T} \rightarrow A$ be a trace. A *run* r of $\mathcal{TA}$ over v is a run of $\mathcal{A}$ over v; r is *accepting* for $\mathcal{TA}$ iff

1. r is accepting for $\mathcal{A}$ and
2. r satisfies the time constraints.

A timed $\forall$-automaton $\mathcal{TA}$ *accepts* a trace v, written $v \models \mathcal{TA}$, iff *all* possible runs of $\mathcal{TA}$ over v are accepting. For example, Figure 11 (a) specifies a real-time response property meaning that any event (E) will be responded to (F) within 30 time units. Figure 11 (b) specifies a conditioned real-time response property meaning that R will always be reached within 30 time units of B, i.e., the time in S is not counted.

It has been shown [24] that discrete $\forall$-automata have the same expressive power as Buchi automata [31] and the extended temporal logic (ETL) [32], which are strictly more powerful than the propositional linear temporal logic (PLTL) [31,32]. However, timed $\forall$-automata are not directly comparable with TBA [2]. A local timing constraint in (discrete) timed $\forall$-automata can also be specified in TBA. However, global timing constraints cannot be specified within TBA, since it is not possible to stop a clock except by resetting it. On the other hand, there are properties of timed behaviors that can be specified by TBA but cannot be specified by timed $\forall$-automata.

3.2 Requirements specification for elevator systems

A well-designed elevator system should guarantee that any request will be served within some bounded time. We can specify such requirements in timed $\forall$-automata.

There are three kinds of requests: to go to a particular floor after entering the elevator, or to go up or down when waiting for the elevator. The following are some examples of assertions.

- $R2 : Fb_s(2) = 1$ denotes that "there is a request to go to the second floor."
- $R2S : Fb_s(2) = 0$ denotes that "the request to go to the second floor is served."
- $RU2 : Ub_s(2) = 1$ denotes that "there is a request to go up at the second floor."
- $RU2S : Ub_s(2) = 0$ denotes that "the request to go up at the second floor is served."

Notice that an elevator may stop at a floor forever given that a request button in that floor is pushed continuously. The real-time response specification in Figure 11 (a) will not be satisfied in general (e.g. given $E =_{def} R2$ and $F =_{def} R2S$). Instead, a conditioned real-time response specification in Figure 11 (b) should be adopted. For example, let $S =_{def} Com = 0$, $B =_{def} RU2 \wedge (Com \neq 0)$ and $R =_{def} RU2S$, the specification in Figure 11 (b) means that a request to go up in the second floor will be served within 30 time units of elevator's motion time.

4 Hybrid Control System Design

Given a model of the system and its requirements specification, it is always a challenge to design a "correct" control system so that the overall system satisfies the specification. There is no automatic method in general. However, there are design principles that can be applied case by case.

In section 2, we have modeled the complete elevator system with modules `CONTROL0` and `CONTROL1` as the black boxes. `CONTROL0` is an analog controller that generates force to drive the elevator body, and `CONTROL1` is a discrete controller with the current floor number f and the current button states b_s as inputs and with command `Com` and serving state s as outputs. In this section, we will design these two control modules as a case study.

4.1 Continuous control design

The simplest analog controller is a linear proportional and derivative (PD) controller. Let

$$F = \begin{cases} F_0 & \text{if } Com = 1 \\ -F_0 & \text{if } Com = -1 \\ -K_p d_s - K_v \dot{d}_s & \text{if } Com = 0 \end{cases} \tag{5}$$

where d_s is the distance to the closest floor, F_0 is a positive constant, K_p is a proportional gain and K_v is a derivative gain. The rest of design is to choose F_0, K_p, K_v so that the following two conditions are satisfied:

1. Continuous Stability: the continuous control is stable;

2. Hybrid Consistency: the interface to the discrete control is consistent.

- **Stability**
  Notice that $\dot{h} = \dot{d}_s$, so for $Com = 0$, we have

$$\ddot{d}_s + (K + K_v)\dot{d}_s + K_p d_s = 0. \tag{6}$$

  by combining Equation 1 with Equation 5.

  The stability theorem for the second order differential equation tells us that if both $K + K_v$ and K_p are positive, the continuous system is stable, because the roots of the characteristic equation have negative real parts.

- **Consistency**
  The home position is defined as $|d_s| \leq \epsilon$ for some $\epsilon > 0$. In order to have hybrid consistency, we have to make sure that if $Com = 0$, then $Com = 0$ implies $|d_s| \leq \epsilon$ for all time (this formal requirement is represented by a $\forall$-automaton in Figure 12), i.e., no overshoot above ϵ should happen after a stop command is issued.

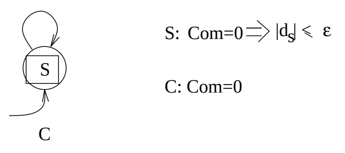


Fig. 12. The specification of the stop control

Let

$$K'_v = K + K_v \tag{7}$$

and

$$(K'_v)^2 = 4K_p. \tag{8}$$

The solution to the differential equation Equation 6 is

$$d_s = (C_0 + C_1 t)e^{-\frac{1}{2}K'_v t} \tag{9}$$

where $C_0 = -\epsilon$ and $C_1 = V_0 - \frac{1}{2}K'_v\epsilon$, where V_0 is the initial speed at distance ϵ.

The maximum distance to the floor D can be computed as follows: The maximum distance is achieved at $\dot{d}_s = 0$. Derived from Equation 9, we have $C_1 = \frac{1}{2}K'_v(C_0 + C_1 t)$, i.e., the maximum distance happens at time

$t = 2/K_v' - C_0/C_1$. Putting this t back to Equation 9, this control will overshoot at most D where

$$\begin{aligned} D &= (2C_1/K_v')e^{-1-C_0/(2C_1/K_v')} \\ &= (2V_0/K_v' - \epsilon)e^{-1-\epsilon/(2V_0/K_v'-\epsilon)}. \end{aligned}$$

If we choose

$$K_v' = V_0/\epsilon, \tag{10}$$

we have $D = \epsilon e^{-2}$. Therefore, $\max_t |d_s(t)| \leq \epsilon$.

Given F_0, the maximum speed of the elevator is then

$$V_0 = F_0/K \tag{11}$$

since the solution to Equation 1 is $\dot{h} = (F_0/K)(1 - e^{-Kt})$ given that F is F_0. If the maximum acceleration of the motor is a, we have to choose

$$F_0 \leq a \tag{12}$$

since $F = \ddot{h} + K\dot{h}$. On the other hand, if the maximum deceleration of the motor is d, V_0 must satisfy

$$K_v' V_0 - K_p \epsilon \leq d \tag{13}$$

according to Equation 6. Combining Equation 13 with Equation 10 and Equation 8, we have

$$V_0 \leq \sqrt{4\epsilon d/3}. \tag{14}$$

Combining Equation 12 and Equation 14,

$$F_0 \leq \min(K\sqrt{4\epsilon d/3}, a). \tag{15}$$

For the rest of parameters, we can have

$$K_v' = F_0/(K\epsilon) \tag{16}$$

using Equation 10 and Equation 11. Then

$$K_v = K_{v'} - K \tag{17}$$

using Equation 7 and

$$K_p = K_v^2/4 \tag{18}$$

using Equation 8.

Suppose $K = 1.0$, $a = d = 0.5$, and $\epsilon = 0.15$, we can set $F_0 = 0.33$ that satisfies Equation 15, $K_v = 1.2$ and $K_p = 1.21$.

4.2 Discrete control design

The discrete controller **CONTROL1** is a logical component. It receives the current request from the state of push buttons b_s, according to the current floor number f and the current serving state s, determines the next motion of the elevator Com and the next serving state s. We assume here three kinds of serving states: `up`, `down` and `idle`. Furthermore, we assume that the elevator is always parked (idle) at the first floor.

There are three types of request to the elevator: `UpRequest`, `DownRequest` and `StopRequest`.

- Let Ur indicate whether or not there is a request for the elevator to go up:

$$\begin{aligned} Ur &= UpRequest(f, Ub, Db, Fb) \\ &= Ub(f) \vee \bigvee_{n \geq k > f} (Ub(k) \vee Db(k) \vee Fb(k)). \end{aligned}$$

- Let Dr indicate whether or not there is a request for the elevator to go down:

$$\begin{aligned} Dr &= DownRequest(f, Ub, Db, Fb) \\ &= Db(f) \vee \bigvee_{1 \leq k < f} (Ub(k) \vee Db(k) \vee Fb(k)). \end{aligned}$$

- Let Sr indicate whether or not there is a request for the elevator to stop and the request is consistent to the current serving state s:

$$\begin{aligned} Sr &= StopRequest(f, Ub, Db, Fb, s) \\ &= \begin{cases} Db(f) \vee Fb(f) \text{ if } s = down \\ Ub(f) \vee Fb(f) \text{ otherwise.} \end{cases} \end{aligned}$$

In order to satisfy the requirements specification that any request is served in some bounded time, we have to make sure that there is no dead-lock or live-lock for all possible situations. For example, a control that always serves the nearest floor may get stuck and never respond to a request further away. Here is a simple strategy we use: the elevator will move persistently in one direction until there is no request in that direction. More specifically, if there is a request for the elevator to go up and either the last serving state is up or there is no request to go down, the current serving state will be up; if there is no request to go up and the elevator is not at the first floor, or the last serving state is down and there is a request to go down, then the the current serving state will be down; otherwise the current serving state will be idle, that is, the elevator will be parked at the first floor if there are no more requests. Formally,

$$\begin{aligned} s' &= ServingState(f, s, Ur, Dr) \\ &= \begin{cases} up & \text{if } Ur \wedge (s \neq down \vee \neg Dr) \\ down & \text{if } (\neg Ur \wedge f > 1) \vee (Dr \wedge s = down) \\ idle & \text{otherwise} \end{cases} \end{aligned}$$

Then, the current command can be determined as follows:

$$\begin{aligned} Com &= Command(Sr, s) \\ &= \begin{cases} 0 & \text{if } Sr \text{ or } s = idle \\ 1 & \text{if } \neg Sr \text{ and } s = up \\ -1 & \text{otherwise.} \end{cases} \end{aligned}$$

Putting all the functions together, the constraint net model of the discrete controller is shown in Figure 13.

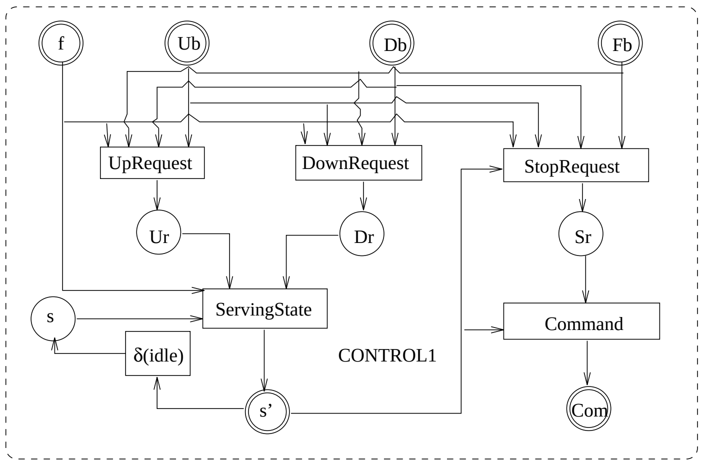


Fig. 13. The discrete control module

5 Behavior Verification Using Model Checking

Given a constraint net model of a system and a timed ∀-automaton specification of a behavior, the behavior of the system satisfies the requirements specification if and only if the (behavior) traces of the system are accepting for the timed ∀-automaton.

We develop a model checking method that uses the induction principle and generalizes both Liapunov stability analysis for dynamic systems and monotonicity of well-foundedness in discrete-event systems.

A representation between constraint nets and timed ∀-automata is a state-based transition system, such as a Kripke structure. The verification rules are applied to the Kripke structure.

5.1 Generalized Kripke structure

A useful and important type of behavior is state-based and time-invariant. Intuitively, a state-based and time-invariant behavior is a behavior whose traces after any time are totally dependent on the current snapshot. State-based and time-invariant behaviors can be defined using generalized Kripke structures [35].

A *generalized Kripke structure* $\mathcal{K}$ as a triple $\langle \mathcal{S}, \rightarrow, \Theta \rangle$ where $\mathcal{S}$ is a set of states, $\rightarrow \subset \mathcal{S} \times \mathcal{R}^+ \times \mathcal{S}$ is a state transition relation, and $\Theta \subseteq \mathcal{S}$ is a set of initial states. We denote $\langle s_1, t, s_2 \rangle \in \rightarrow$ as $s_1 \xrightarrow{t} s_2$. The state transition relation $\rightarrow$ satisfies the following conditions:

- *initiality*: $s \xrightarrow{0} s$;
- *transitivity*: if $s_1 \xrightarrow{t_1} s_2$ and $s_2 \xrightarrow{t_2} s_3$, then $s_1 \xrightarrow{t_1+t_2} s_3$;
- *infinity*: $\forall s \in \mathcal{S}, \exists t > 0, s' \in \mathcal{S}, s \xrightarrow{t} s'$.

For example, the behavior of $\dot{x} = -x$ can be represented by a generalized Kripke structure $\langle \mathcal{R}, \rightarrow, \Theta \rangle$ with $s_1 \xrightarrow{t} s_2$ iff $s_2 = s_1 e^{-t}$.

Let φ and ψ be assertions on states and time durations. For a generalized Kripke structure $\mathcal{K} = \langle \mathcal{S}, \rightarrow, \Theta \rangle$, let $\{\varphi\}\mathcal{K}\{\psi\}$ denote the validity of the following two consecution conditions:

- *Inductivity* $\{\varphi\}\mathcal{K}^-\{\psi\}$: $\exists \delta > 0, \forall 0 < t \leq \delta, \forall s, (\varphi(s) \wedge (s \xrightarrow{t} s') \Rightarrow \psi(s', t))$.
- *Continuity* $\{\varphi\}\mathcal{K}^+\{\psi\}$: $\varphi(s) \Rightarrow \exists \delta > 0, \forall 0 < t < \delta, \forall s', ((s \xrightarrow{t} s') \Rightarrow \psi(s', t))$.

These two conditions are derived from the continuous induction principle. If time is discrete, these two conditions reduce to one, i.e., $\varphi(s) \wedge (s \xrightarrow{\delta} s') \Rightarrow \psi(s', \delta)$ where δ is the time duration between s and s'.

5.2 Verification rules

The formal method for behavior verification consists of a set of model-checking rules , which is a generalization of the model-checking rules developed for concurrent programs [24].

There are three types of rules: invariance rules (I), stability or eventuality rules (L) and timeliness rules (T). Let $\mathcal{A}$ be a $\forall$-automaton $\langle Q, R, S, e, c \rangle$ and $\mathcal{K}$ be a generalized Kripke structure $\langle \mathcal{S}, \rightarrow, \Theta \rangle$. The invariance rules check to see if a set of assertions $\{\alpha\}_{q \in Q}$ is a set of invariants for $\mathcal{A}$ and $\mathcal{K}$, i.e., for any trace v of $\mathcal{K}$ and any run r of $\mathcal{A}$ over v, $\forall t \in \mathcal{T}, v(t) \models \alpha_{r(t)}$. Given $B = Q - (R \cup S)$, the stability or eventuality rules check if the B-states in any run of $\mathcal{A}$ over any trace of $\mathcal{K}$ will be terminated eventually. Given $\mathcal{TA}$ as a timed $\forall$-automaton $\langle \mathcal{A}, T, \tau \rangle$, the timeliness rules check if the T-states and the B-states in any run of $\mathcal{A}$ over any trace of $\mathcal{K}$ are bounded by the time function τ. The set of model-checking rules can be represented in first-order logic, some of which are in the form of $\{\varphi\}\mathcal{K}\{\psi\}$.

Here are the model-checking rules for a behavior represented by $\mathcal{K} = \langle \mathcal{S}, \rightarrow, \Theta \rangle$ and a specification represented by $\mathcal{TA} = \langle \mathcal{A}, T, \tau \rangle$ where $\mathcal{A} = \langle Q, R, S, e, c \rangle$:

Invariance Rules (I): A set of assertions $\{\alpha_q\}_{q \in Q}$ is called a set of *invariants* for $\mathcal{K}$ and $\mathcal{A}$ iff

(I1) *Initiality*: $\forall q \in Q, \Theta \wedge e(q) \Rightarrow \alpha_q$.
(I2) *Consecution*: $\forall q, q' \in Q, \{\alpha_q\}\mathcal{K}\{c(q,q') \Rightarrow \alpha_{q'}\}$.

The Invariance Rules are the same as those in [24] except that the condition for consecution is generalized.

Stability or Eventuality Rules (L): Given that $\{\alpha_q\}_{q\in Q}$ is a set of invariants for $\mathcal{K}$ and $\mathcal{A}$, a set of partial functions $\{\rho_q\}_{q\in Q} : \mathcal{S} \rightarrow \mathcal{R}^+$ is called a set of *Liapunov functions* for $\mathcal{K}$ and $\mathcal{A}$ iff the following conditions are satisfied:

(L1) *Definedness*: $\forall q \in Q,\ \ \alpha_q \Rightarrow \exists w, \rho_q = w$.
(L2) *Non-increase*: $\forall q \in S, q' \in Q$, $\{\alpha_q \wedge \rho_q = w\}\mathcal{K}^-\{c(q,q') \Rightarrow \rho_{q'} \leq w\}$ and $\forall q \in Q, q' \in S$, $\{\alpha_q \wedge \rho_q = w\}\mathcal{K}^+\{c(q,q') \Rightarrow \rho_{q'} \leq w\}$.
(L3) *Decrease*: $\exists \epsilon > 0, \forall q \in B, q' \in Q$, $\{\alpha_q \wedge \rho_q = w\}\mathcal{K}^-\{c(q,q') \Rightarrow \frac{\rho_{q'}-w}{t} \leq -\epsilon\}$ and $\forall q \in Q, q' \in B$, $\{\alpha_q \wedge \rho_q = w\}\mathcal{K}^+\{c(q,q') \Rightarrow \frac{\rho_{q'}-w}{t} \leq -\epsilon\}$.

If time is discrete, $\mathcal{K}^+$ rules will not be used. The Stability or Eventuality Rules generalize both stability analysis of discrete or continuous dynamic systems [20] and well-foundedness for finite termination in concurrent systems [24].

Timeliness Rules (T): Corresponding to two types of time bound, we define two timing functions. Let $\{\alpha_q\}_{q\in Q}$ be invariants for $\mathcal{K}$ and $\mathcal{A}$. A set of partial functions $\{\gamma_q\}_{q\in T}$ is called a set of *local timing functions* for $\mathcal{K}$ and $\mathcal{TA}$ iff $\gamma_q : \mathcal{S} \rightarrow \mathcal{R}^+$ satisfies the following conditions:

(T1) *Boundedness*: $\forall q \in T,\ \alpha_q \Rightarrow \gamma_q \leq \tau(q)$ and $\forall q \in T, q' \in Q$, $\{\alpha_q \wedge \gamma_q = w\}\mathcal{K}^-\{c(q,q') \Rightarrow w \geq t\}$.
(T2) *Decrease*: $\forall q \in T, \{\alpha_q \wedge \gamma_q = w\}\mathcal{K}\{c(q,q) \Rightarrow \frac{\gamma_q - w}{t} \leq -1\}$.

A set of partial functions $\{\eta_q\}_{q\in Q}$ is called a set of *global timing functions* for $\mathcal{K}$ and $\mathcal{TA}$ iff $\eta_q : \mathcal{S} \rightarrow \mathcal{R}^+$ satisfies the following conditions:

(T3) *Definedness*: $\forall q \in Q, \alpha_q \Rightarrow \exists w, \eta_q = w$.
(T4) *Boundedness*: $\forall q \in B, \alpha_q \Rightarrow \eta_q \leq \tau(bad)$.
(T5) *Non-increase*: $\forall q \in S, q' \in Q$, $\{\alpha_q \wedge \eta_q = w\}\mathcal{K}^-\{c(q,q') \Rightarrow \eta_{q'} \leq w\}$ and $\forall q \in Q, q' \in S$, $\{\alpha_q \wedge \eta_q = w\}\mathcal{K}^+\{c(q,q') \Rightarrow \eta_{q'} \leq w\}$.
(T6) *Decrease*: $\forall q \in B, q' \in Q$, $\{\alpha_q \wedge \eta_q = w\}\mathcal{K}^-\{c(q,q') \Rightarrow \frac{\eta_{q'}-w}{t} \leq -1\}$ and $\forall q \in Q, q' \in B$, $\{\alpha_q \wedge \eta_q = w\}\mathcal{K}^+\{c(q,q') \Rightarrow \frac{\eta_{q'}-w}{t} \leq -1\}$.

If time is discrete, $\mathcal{K}^+$ rules will not be used. The Timeliness Rules are modifications of the Eventuality Rules; they enforce real-time boundedness, in addition to termination.

A set of model-checking rules is *sound* if verification by the rules guarantees the correctness of the behavior against the specification; it is *complete* if the correctness of the behavior against the specification guarantees verification by the rules. The soundness and completeness of the rules are discussed in [35].

Also notice that the set of verification rules are formal guidance for the proof of a system; it is by no means automatic. However, if the system is discrete and there are computer-aided proof systems available, semi-automatic proofs can be applied. If the system is discrete and finite, automatic algorithms can be derived from the rules [34].

5.3 Is the elevator system well-designed?

This question can be answered only if we can answer the following three questions: (1) Does the model reflect the real system at the appropriate level of abstraction? (2) Is the set of requirements complete? and (3) Does the model satisfy the requirements? The method that we propose here will answer the third question.

Given a 3-floor elevator system, we can check whether or not a request to go up at the second floor will be served within some bounded time units of elevator's motion time, i.e., if the constraint net model in Figure 9 satisfies the timed $\forall$-automaton specification in Figure 11 (b).

Our method involves the following steps:

First, find a Kripke structure of the constraint nets corresponding to the level of specification. A continuous system can have a discrete Kripke structure if the specification is on high level. In this case, if there is a request to go up in the second floor, a Kripke structure of the behavior of the system can be obtained (Figure 14). Each state is of form (f, s, Com) indicating the current floor, the serving state, and the command of the elevator, with $(1, up, 0)$ as the initial state. The dashed transitions indicate the transitions inhibited by the control given the current request state, and self loops indicate the stationary transitions.

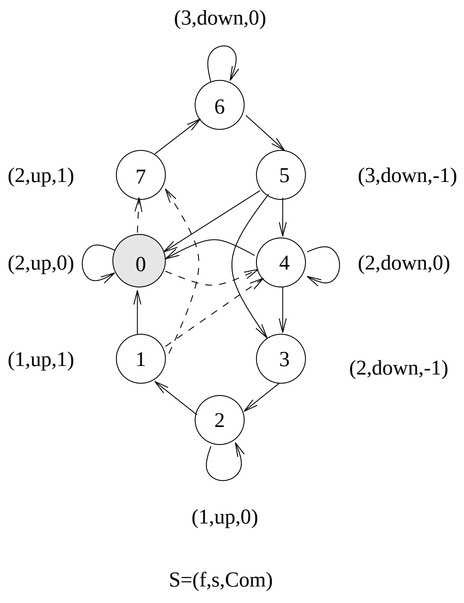


Fig. 14. Generalized Kripke structure of the behavior

Since the Kripke Structure we use is finite for this case, automatic proof can be applied. The algorithm is directly obtained from the rules, so we will illustrate how the rules are applied here.

1. Find a set of invariants for the timed $\forall$-automaton in Figure 11 (b) that satisfies the Invariance Rules. In this case, let q_1, q_2, q_3 be states with B, S and R assertions respectively. If $B =_{def} RU2 \wedge (Com \neq 0)$, $S =_{def} Com = 0$, and $R =_{def} RU2S$, we can see that B, S and R are invariants for q_1, q_2 and q_3 respectively.
2. Find a set of Liapunov functions. Let ρ be a function whose value for each state is the longest possible path from that state to the desired shaded state, without self loops (see the number in each state in Figure 14). It is easy to see that rho (for all q_1, q_2 and q_3), together with the invariants, satisfy the Stability or Eventuality Rules. That is, the request will be served within bounded time of motion.
3. Find a set of global timing functions. Let the maximum traversal time of the elevator from one floor to another be T, we define a global timing function as follows:

$$\begin{aligned}
\eta(2, up, 0) &= 0;\\
\eta(1, up, 1) &= T;\\
\eta(1, up, 0) &= T;\\
\eta(2, down, -1) &= 2 * T;\\
\eta(2, down, 0) &= 2 * T;\\
\eta(3, down, -1) &= 3 * T;\\
\eta(3, down, 0) &= 3 * T;\\
\eta(2, up, 1) &= 4 * T;
\end{aligned}$$

It is easy to see that η (for all q_1, q_2 and q_3), is a global timing function iff $4 * T \leq 30$.

we can calculate the maximum traversal time T of the elevator from one floor to another as follows. Since $\dot{h} = (F/K)(1 - e^{-Kt})$ and $h(0) = 0$, we have $h = (F/K)(t + (1/K)e^{-Kt}) - F/(K^2)$. Suppose the distance between floors is H. The time to traverse one level from stationary state will be $H \geq (F/K)(t) - F/(K^2)$, i.e., $t \leq HK/F + 1/K$. If $H = 2.0$, $K = 1.0$ and $F = 0.33$, we have $t \leq 7.061$. Therefore $T = 7.061$.

6 Conclusions

We have presented a formal approach to the modeling and analysis of hybrid control systems. The approach consists of four interleaving or concurrent phases: hybrid dynamic system modeling, requirements specification, hybrid control design and behavior verification. We have used Constraint Nets for modeling hybrid dynamic systems, timed $\forall$-automata for specifying requirements, coupled continuous control with event-driven logic, and the rule-based model checking for verifying behaviors of the overall systems.

Our approach is demonstrated by an elevator case study. The hybrid control system of the elevator couples a quite complex discrete control logic with an analog control law. The inconsistent behavior between the discrete and continuous components, as well as the incorrect behavior at each level, may cause the malfunction of the overall system. We have simulated the elevator system modeled by Constraint Nets in both Simulink and SystemBuild.

As we reviewed in this paper, much work has been done in modeling and analysis of hybrid systems [10]. Which method to use is largely dependent on the problems at hand.

Like most dataflow/net formalisms, Constraint Nets are modular and hierarchical so as to model complex hybrid systems. Developed from abstract algebra, CN provides a uniform representation for hybrid systems with a formal syntax and semantics for simulation. However, since there are no closed-form solutions for most problems, the analysis is, in general, hard.

Timed $\forall$-automata are simple for requirements specification, However, they are not powerful enough to represent all the possible behaviors.

Our verification method provides a set of formal model checking rules, which can be used to guide a formal proof procedure. However, the invariants, Liapunov and timing functions are not automatically created and the verification of the rules is not automatic, in general.

Nevertheless, the approach we developed here can be used to solve many problems in hybrid systems modeling, designing, specification and verification. With this approach, we have also developed controllers for robot soccer players [39] and hydraulically controlled robot arms [34]. The same approach can be applied to the modeling and control of most complex electromechanical systems.

Acknowledgements

Thanks to Peter Caines, Hector Levesque, Ray Reiter and Maarten van Emden for stimulating us to work on the elevator system. This work was supported in part by the Natural Sciences and Engineering Research Council, the Institute for Robotics and Intelligent Systems and the Canadian Institute for Advanced Research.

References

1. R. Alur, C. Courcoubetis, T. A. Henzinger, and P. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems*, number 736 in Lecture Notes on Computer Science, pages 209 – 229. Springer-Verlag, 1993.
2. R. Alur and D. Dill. Automata for modeling real-time systems. In M. S. Paterson, editor, *ICALP90: Automata, Languages and Programming*, number 443 in Lecture Notes on Computer Science, pages 322 – 335. Springer-Verlag, 1990.
3. R. Alur and D. Dill. The theory of timed automata. In J.W. deBakker, C. Huizing, W.P. dePoever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, number 600 in Lecture Notes on Computer Science, pages 45 – 73. Springer-Verlag, 1991.
4. R. Alur and T. A. Henzinger. A really temporal logic. In *30th Annual Symposium on Foundations of Computer Science*, pages 164 – 169, 1989.
5. H. Barringer. Up and down the temporal way. Technical report, Computer Science, University of Manchester, England, September 1985.
6. A. Benveniste and P. LeGuernic. Hybrid dynamical systems theory and the SIGNAL language. *IEEE Transactions on Automatic Control*, 35(5):535 – 546, May 1990.
7. B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using Time Petri Nets. *IEEE Transactions on Software Engineering*, 17(3):259 – 273, March 1991.
8. R. Brockett. Hybrid models for motion control systems. In H. Trentelman and J. C. Willems, editors, *Perspectives in Control*, pages 29 – 54. 1993.
9. D.N. Dyck and P.E. Caines. The logical control of an elevator. *IEEE Trans. Automatic Control*, (3):480 – 486, March 1995.
10. R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors. *Hybrid Systems*. Number 736 in Lecture Notes on Computer Science. Springer-Verlag, 1993.
11. V. Gupta, R. Jagadeesan, V. Saraswat, and D. Bobrow. Programming in hybrid constraint languages. In P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, *Hybrid Systems II*, number 999 in Lecture Notes on Computer Science. Springer-Verlag, 1995.
12. R. Hale. Using temporal logic for prototyping: The design of a lift controller. In H.S.M. Zedan, editor, *Real-Time Systems, Theory and Applications*. Elsvier Science Publishers B.V. (North-Holland), 1990.
13. Integrated Systems Inc. SystemBuild User's Guide.
14. The MathWorks Inc. Simulink User's Guide.
15. M.A. Jackson. *System Development*. Prentice-Hall, Englewood Cliffs, 1983.
16. M. S. Jaffe, N. G. Leveson, M. P. E. Heimdahl, and B. E. Melhart. Software requirements analysis for real-time process-control systems. *IEEE Transactions on Software Engineering*, 17(3):241 – 257, March 1991.
17. B. H. Krogh. Condition/event signal interfaces for block diagram modeling and analysis of hybrid systems. In *8th IEEE International Symposium on Intelligent Control*, Chicago, IL, August 25–27 1993.
18. L. Lamport. Hybrid systems in TLA+. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems*, number 736 in Lecture Notes on Computer Science, pages 77 – 102. Springer-Verlag, 1993.
19. N. G. Leveson, M. P. E. Heimdahl, H. Hildreth, and J. D. Reese. Requirements specification for process-control systems. *IEEE Transactions on Software Engineering*, 20(9):684 – 707, September 1994.

20. D. G. Luenberger. *Introduction to Dynamic Systems: Theory, Models and Applications.* John Wiley & Sons, 1979.
21. N. Lynch. Modeling and verification of automated transit systems, using timed automata, invariants and simulations. In *Hybrid Systems III.* in Verification and Control of Hybrid Systems, October, 1995, to appear in LNCS.
22. N. Lynch. Simulation techniques for proving properties of real-time systems. In J.W. deBakker, W.P. dePoever, and G. Rozenberg, editors, *A Decade of Concurrency*, number 803 in Lecture Notes on Computer Science, pages 376 – 424. Springer-Verlag, 1993.
23. O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In J.W. deBakker, C. Huizing, W.P. dePoever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, number 600 in Lecture Notes on Computer Science, pages 448 – 484. Springer-Verlag, 1991.
24. Z. Manna and A. Pnueli. Specification and verification of concurrent programs by ∀-automata. In *Proc. 14th Ann. ACM Symp. on Principles of Programming Languages*, pages 1–12, 1987.
25. A. Nerode and W. Kohn. Models for hybrid systems: Automata, topologies, controllability, observability. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems*, number 736 in Lecture Notes on Computer Science, pages 317 – 356. Springer-Verlag, 1993.
26. X. Nicollin and J. Sifakis. From ATP to timed graphs and hybrid systems. In J.W. deBakker, C. Huizing, W.P. dePoever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, number 600 in Lecture Notes on Computer Science, pages 549 – 572. Springer-Verlag, 1991.
27. B. Sanden. An entity-life modeling approach to the design of concurrent software. *Communication of the ACM*, 32(3):230 – 243, March 1989.
28. V. Saraswat. Concurrent constraint programming languages. Technical report, Computer Science Department, Carnegie–Mellon University, 1989. Ph. D. thesis.
29. M. Shaw. Comparing architectural design styles. *IEEE Software*, pages 27 – 41, November 1995.
30. I. E. Sutherland. Micropipeline. *Communication of ACM*, 32(6):720 – 738, June 1989.
31. W. Thomas. Automata on infinite objects. In Jan Van Leeuwen, editor, *Handbook of Theoretical Computer Science.* MIT Press, 1990.
32. P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56:72 – 99, 1983.
33. X. Yu, J. Wang, C. Zhou, and P. Pandya. Formal design of hybrid systems. In H. Langmaack, W.P. deRoever, and J. Vytopil, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, number 863 in Lecture Notes on Computer Science. Springer-Verlag, 1994.
34. Y. Zhang. A foundation for the design and analysis of robotic systems and behaviors. Technical Report 94-26, Department of Computer Science, University of British Columbia, 1994. Ph.D. thesis.
35. Y. Zhang and A. K. Mackworth. Specification and verification of hybrid dynamic systems using timed ∀-automata. In *Hybrid Systems III.* in Verification and Control of Hybrid Systems, October, 1995, to appear in LNCS.
36. Y. Zhang and A. K. Mackworth. Specification and verification of constraint-based dynamic systems. In A. Borning, editor, *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science 874, pages 229 – 242. Springer Verlag, 1994.

37. Y. Zhang and A. K. Mackworth. Constraint Nets: A semantic model for hybrid dynamic systems. *Theoretical Computer Science*, 138(1):211 – 239, 1995. Special Issue on Hybrid Systems.
38. Y. Zhang and A. K. Mackworth. Constraint programming in constraint nets. In V. Saraswat and P. Van Hentenryck, editors, *Principles and Practice of Constraint Programming*, pages 49 – 68. MIT Press, 1995.
39. Y. Zhang and A. K. Mackworth. Synthesis of hybrid constraint-based controllers. In P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, *Hybrid Systems II*, Lecture Notes in Computer Science 999, pages 552 – 567. Springer Verlag, 1995.