Specification and Verification of Constraint-Based Dynamic Systems

Ying Zhang
Department of Computer Science
University of British Columbia
Vancouver, B.C.
Canada V6T 1Z4
zhang@cs.ubc.ca

Alan K. Mackworth *
Department of Computer Science
University of British Columbia
Vancouver, B.C.
Canada V6T 1Z4
mack@cs.ubc.ca

Abstract

Constraint satisfaction can be seen as a dynamic process that approaches the solution set of the constraints asymptotically [8]. Constraint programming is seen as creating a dynamic system with the desired property. We have developed a semantic model for dynamic systems, Constraint Nets, which serves as a useful abstract target machine for constraint programming languages, providing both semantics and pragmatics. Generalizing, here we view a constraint-based dynamic system as a dynamic system which approaches the solution set of the constraints infinitely often. Most robotic systems are constraint-based dynamic systems with tasks specified as constraints. In this paper, we further explore the specification and verification of constraint-based dynamic systems. We first develop generalized $\forall$-automata for the specification and verification of general (hybrid) dynamic systems, then explicate the relationship between constraint-based dynamic systems and their desired behavior specifications.

1 Motivation and Introduction

We have previously proposed viewing constraints as relations and constraint satisfaction as a dynamic process of approaching the solution set of the constraints asymptotically [8]. Under this view, constraint programming is the creation of a dynamic system with the desired property. We have developed a semantic model for dynamic systems, Constraint Nets, which serves as a useful abstract target machine for constraint programming languages, providing both semantics and pragmatics. Properties of various discrete and continuous constraint methods for constraint programming were also examined [8].

Generalizing, here we consider a constraint-based dynamic system as a dynamic system which approaches the solution set of the constraints infinitely often. One of the motivations for this view is to design and analyze a robotic system composed of a controller that is coupled to a plant and an environment. The desired behavior of the controller may be specified as a set of constraints, which, in general, vary with time. Thus, the controller should be synthesized so as to solve the constraints on-line. Consider a tracking system where the target may move from time to time. A well-designed tracking control system has to ensure that the target can be tracked down infinitely often.

Here we start with general concepts of dynamic systems using abstract notions of time, domains and traces. With this abstraction, hybrid as well as discrete and continuous dynamic systems can be studied in a unitary framework. The behavior of a dynamic system is then defined as the set of possible traces produced by the system.

In order to specify desired behaviors of a dynamic system, we develop a formal specification language, a generalized version of $\forall$-automata [3]. In order to verify that a dynamic system satisfies its behavior specification, we develop a formal model checking method with generalized Liapunov functions.

*Shell Canada Fellow, Canadian Institute for Advanced Research

A constraint-based dynamic system is a special type of dynamic system. We explore the properties of constraint-based dynamic systems and constraint-based behavior specifications, then relate system verification to control synthesis.

The rest of the paper is organized as follows. Section 2 briefly presents concepts of general dynamic systems and constraint net modeling. Section 3 develops generalized $\forall$-automata for specifying and verifying desired behaviors of dynamic systems. Section 4 characterizes constraint-based dynamic systems and their behavior specifications. Section 5 concludes the paper and points out related work.

2 General Dynamic Systems

In this section, we first introduce some basic concepts in general dynamic systems: time, domains and traces, then present a formal model for general dynamic systems.

2.1 Concepts in dynamic systems

In order to model dynamic systems in a unitary framework, we present abstract notions of time structures, domains and traces. Both time structures and domains are defined on metric spaces.

Let $\mathcal{R}^+$ be the set of nonnegative real numbers. A *metric space* is a pair $\langle X, d\rangle$ where X is a set and $d : X \times X \rightarrow \mathcal{R}^+$ is a *metric* defined on X, satisfying the following axioms for all $x, y, z \in X$:

1. $d(x, y) = d(y, x)$.
2. $d(x, y) + d(y, z) \geq d(x, z)$.
3. $d(x, y) = 0$ iff $x = y$.

In a metric space $\langle X, d\rangle$, $d(x, y)$ is called "the distance between x and y". We will use X to denote the metric space $\langle X, d\rangle$ if no ambiguity arises.

A *time structure* is a metric space $\langle \mathcal{T}, d\rangle$ where $\mathcal{T}$ is a totally ordered set with a least element $\mathbf{0}$, d is a metric satisfying $\forall t_0 \leq t_1 \leq t_2 : d(t_0, t_2) = d(t_0, t_1) + d(t_1, t_2)$. We will use $\mathcal{T}$ to denote the time structure $\langle \mathcal{T}, d\rangle$ if no ambiguity arises. A discrete or continuous time structure can be defined according to the topology of its metric space. For example, the set of natural numbers can define a discrete time structure, a left closed interval of real numbers can define a continuous time structure.

A *domain* is a metric space $\langle A, d\rangle$. Let $v : T \rightarrow A$ be a function from a total order T to a domain A. A point $a^* \in A$ is a *limit* of v, iff $\forall \epsilon \exists t_0 \forall t \geq t_0 : d(v(t), a^*) < \epsilon$. Any limit is unique if it exists. We will use $\lim v$ to denote the limit of v if it exists and $\perp_A$ if it does not. Clearly, if T has a greatest element t_0, $\lim v = v(t_0)$.

A trace $v : \mathcal{T} \rightarrow A$ is a function from a time structure $\mathcal{T}$ to a domain A. Let $T \subseteq \mathcal{T}$ be a downward closed subset of $\mathcal{T}$, i.e. $t \in T$ implies $\forall t' \leq t : t' \in T$. We use $\lim v_{|T}$ to denote the limit of v on the total order T. For simplicity in representation, we introduce the following notions: given a time structure $\langle \mathcal{T}, d\rangle$ and a real number $\tau \in \mathcal{R}^+$,

- $pre(t) = \{t' \in \mathcal{T} \,|\, t' < t\}$, and $v(pre(t)) = \lim v_{|pre(t)}$;
- $t - \tau = \{t' \in \mathcal{T} \,|\, t' < t, d(t, t') \geq \tau\}$, and $v(t - \tau) = \lim v_{|t-\tau}$;
- $t + \tau = \{t' \in \mathcal{T} \,|\, t' > t, d(t, t') \leq \tau\}$.

Clearly, $pre(t) = t - 0$. If $\mathcal{T}$ is discrete, $v(pre(t))$ is the value of the previous time point and $pre(t) = t - \tau$ whenever τ is small enough.

A time structure $\mathcal{T}$ is *infinite* iff $\forall m > 0, \exists t_0 \in \mathcal{T}, \forall t \geq t_0 : d(\mathbf{0}, t) \geq m$. We will restrict ourselves to infinite time structures. This is not a real restriction, since any time structure $\mathcal{T}$ can be extended to an infinite one $\mathcal{T}' \supset \mathcal{T}$ by letting $v(t) = \lim v_{|\mathcal{T}}$ for all $t \notin \mathcal{T}$.

2.2 Constraint Nets: a model for dynamic systems

We have developed a semantic model, Constraint Nets, for general (hybrid) dynamic systems [10]. We have used the Constraint Net model as an abstract target machine for constraint programming languages [8], while constraint programming is considered as designing a dynamic system that approaches the solution set of the given constraints asymptotically.

Intuitively, a constraint net consists of a finite set of locations, a finite set of transductions, each with a finite set of input ports and an output port, and a finite set of connections between locations and ports of transductions. A location can be regarded as a wire, a channel, a variable, or a memory location, whose values may change over time. A transduction is a mapping from input traces to output traces, with the causal restriction, viz. the output value at any time is determined by the input values up to that time. For example, a temporal integration with an initial value is a typical transduction on a continuous time structure and any state automaton with an initial state defines a transduction on a discrete time structure.

A location l is the *output location* of a transduction F, iff l connects to the output port of F; l is an *input location* of F, iff l connects to an input port of F. Let CN be a constraint net. A location l is an *output location* of CN if l is an output location of some transduction in CN otherwise it is an *input location* of CN. The set of input locations of CN is denoted by $I(CN)$, the set of output locations of CN is denoted by $O(CN)$; CN is *closed* if $I(CN) = \emptyset$ otherwise it is *open*.

Semantically, a transduction F denotes an equation $l_0 = F(l_1, \ldots, l_n)$ where l_0 is the output location of F and $\langle l_1, \ldots, l_n \rangle$ is the tuple of input locations of F. A constraint net CN denotes a set of equations, each corresponds to a transduction in CN. The semantics of CN is a 'solution' of the set of equations [10], which is a set of pairs of input and output traces satisfying the equations. Let $Lc = I(CN) \cup O(CN)$ and $\{A_l\}_{l \in Lc}$ be a set of domains in CN. A *state* s of CN is a mapping from the set of locations to their corresponding domains: i.e. $s \in \times_{Lc} A_l$. Therefore, the semantics of CN is also a set of state traces with domain $\times_{Lc} A_l$.

We have modeled two types of constraint solvers, state transition systems and state integration systems, in constraint nets. The former models discrete dynamic processes and the latter models continuous dynamic processes [8]. Hybrid dynamic systems, with both discrete and continuous components, can also be modeled in constraint nets [10, 9]. The *behavior* of a dynamic system is defined as a set of possible input/output traces produced by the system, in our case, the semantics of the constraint net which models the system.

We illustrate the constraint net modeling with two simple examples. The first is a 'standard' example of *Cat and Mouse* modified from [1]. Suppose a cat and a mouse start running from initial positions X_c and X_m respectively, $X_c > X_m > 0$, with constant velocities $V_c < V_m < 0$. Both of them will stop running when the cat catches the mouse, or the mouse runs into the hole in the wall at 0. The behavior of this system is modeled by the following equations CM_1:

$$\begin{aligned} x_c &= \int (X_c)(V_c \cdot c), \\ x_m &= \int (X_m)(V_m \cdot c), \\ c &= (x_c > x_m) \wedge (x_m > 0) \end{aligned}$$

where $\int(X)$ is a temporal integration with initial state X. At any time, c is 1 if the running condition $(x_c > x_m) \wedge (x_m > 0)$ is satisfied and 0 otherwise. Let $\mathcal{R}$ be the set of real numbers and $\mathcal{B} = \{0, 1\}$. This is a closed system. The state of this system is $\langle x_c, x_m, c \rangle \in \mathcal{R} \times \mathcal{R} \times \mathcal{B}$, with its initial state $\langle X_c, X_m, 1 \rangle$. If the cat catches the mouse before the mouse runs into the hole in the wall at 0, i.e. $0 \leq x_c \leq x_m$, the cat wins; if the mouse runs into the hole before the cat, i.e. $x_m \leq 0 \leq x_c$, the mouse wins.

Consider another *Cat and Mouse* problem, where the controller of the cat is synthesized from its constraint specification, i.e. $x_c = x_m$. Suppose the plant of the cat obeys the dynamics $u = \dot{x}_c$ where u is the control input, i.e. the velocity of the cat is controlled. One possible design for the cat controller uses the gradient descent method [8] on the energy function $(x_m - x_c)^2$ to synthesize the feedback control law $u = k \cdot (x_m - x_c), k > 0$ where the distance between the cat and the mouse $x_m - x_c$ can be sensed by the cat. The cat can be modeled as an open constraint net with two equations CM_2:

$$x_c = \int (X_c)u, \quad u = k \cdot (x_m - x_c).$$

Will the cat catch the mouse?

3 Generalized ∀-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 proposed design.

We advocate a formal approach to specifying desired behaviors and to verifying the relationship between a dynamic system and its behavior specification. A trace $v : \mathcal{T} \rightarrow A$ is a generalization of a sequence. In fact, when $\mathcal{T}$ is the set of natural numbers, v is an infinite sequence. A set of sequences defines a conventional formal language. If we take the abstract behavior of a system as a language, a specification can be represented as an automaton, and verification checks the inclusion relation between the language of the system and the language accepted by the automaton.

There is always a trade-off between the power of representation, i.e., the class of languages the type of automaton can accept, and the power of analysis, i.e. the computability of checking the acceptance of traces. We would like the type of automaton to be powerful enough to state certain temporal and real-time properties, yet simple enough to have formal, semi-automatic or automatic verifications. We generalize ∀-automata [3] and Liapunov functions for our purposes.

∀-automata are non-deterministic finite state automata over infinite sequences. These automata were proposed as a formalism for the specification and verification of temporal properties of concurrent programs. It has been shown that ∀-automata have the same expressive power as Buchi automata [6] and the extended temporal logic (ETL) [7], which are strictly more powerful than the linear propositional temporal logic [6, 7]. More importantly, there is a formal verification method [3].

In this section, we generalize ∀-automata to specify languages composed of traces on continuous as well as discrete time structures, and modify the formal verification method [3] by generalizing Liapunov functions [8] and the method of continuous induction [2].

3.1 Behavior Specification

Let an *assertion* be a logical formula defined on states of a dynamic system, i.e. any assertion α on a given state s, denoted $\alpha(s)$, will be evaluated to either $true, s \models \alpha$, or $false, s \not\models \alpha$.

A ∀-*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 ∀-automaton is called *complete* iff the following requirements are met:

- $\bigvee_{q \in Q} e(q)$ is valid.
- For every $q \in Q$, $\bigvee_{q' \in Q} c(q, q')$ is valid.

We will restrict ourselves to complete automata. This is not a real restriction, since any automaton can be transformed to a complete automaton by introducing an additional error state $q_E \in B$, with the corresponding entry condition and transition conditions [3].

Let $\mathcal{T}$ be a time structure and $v : \mathcal{T} \rightarrow A$ be a trace. A *run* of $\mathcal{A}$ over v is a trace $r : \mathcal{T} \rightarrow Q$ satisfying

1. *Initiality*: $v(\mathbf{0}) \models e(r(\mathbf{0}))$;
2. *Consecution*:
   - inductivity: $\forall t > \mathbf{0}, \exists q \in Q$ and $\delta > 0, \forall 0 < \tau \leq \delta : r(t - \tau) = q$ and $v(t) \models c(r(t - \tau), r(t))$ and
   - continuity: $\forall t, \exists q \in Q$ and $\delta > 0, \forall t' \in t + \delta : r(t') = q$ and $v(t') \models c(r(t), r(t'))$.

It is easy to check that when $\mathcal{T}$ is discrete, the two conditions in *Consecution* are reduced to one, i.e. $\forall t > \mathbf{0}, v(t) \models c(r(pre(t)), r(t))$; and if, in addition, $\mathcal{A}$ is complete, every trace has a run. However, if $\mathcal{T}$ is not discrete, even if $\mathcal{A}$ is complete, not every trace has a run. For example, a trace with infinite transitions

among Q within a finite interval has no run. A trace v is *specifiable* by $\mathcal{A}$ iff there is a run of $\mathcal{A}$ over v. The behavior of a system is *specifiable* by $\mathcal{A}$, iff every trace of the behavior is specifiable.

If r is a run, let $Inf(r)$ be the set of automaton-states appearing infinitely many times in r, i.e. $Inf(r) = \{q | \forall t \exists t_0 \geq t, r(t_0) = q\}$. Clearly, if $\mathcal{T}$ has a greatest element t_0, $Inf(r) = r(t_0)$. 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. A $\forall$-automaton $\mathcal{A}$ *accepts* a dynamic system S, written $S \models \mathcal{A}$, iff for every trace v of the behavior of S, $v \models \mathcal{A}$.

One of the advantages of using automata as a specification language is the graphical representation. It is useful and illuminating to represent $\forall$-automata by diagrams. The basic conventions for such representations are the following:

- The automaton-states are depicted by nodes in a directed graph.
- Each initial state is marked by a small arrow, called the *entry arc*, pointing to it.
- Arcs, drawn as arrows, connect some of the states.
- Each recurrent state is depicted by a diamond shape 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. 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, $e(q) = false$.
- Let q, q' be two nodes in the diagram. If q' is labeled by ϕ, and arcs from q to q' are labeled by $\varphi_i, i = 1..n$, the transition condition $c(q, q')$ is given by: $c(q, q') = (\varphi_1 \vee \ldots \vee \varphi_n) \wedge \psi$. If there is no arc from q to q', $c(q, q') = false$.

A diagram representing an incomplete automaton is interpreted as a complete automaton by introducing an error state and associated entry and transition conditions.

This type of automaton is powerful enough to specify various qualitative behaviors. Some typical desired behaviors are shown in Fig. 1. Figure 1(a) accepts a trace which satisfies $\neg G$ only finitely many times, Figure 1(b) accepts a trace which never satisfies B, and Figure 1(c) accepts a trace which will satisfy S in the finite future whenever it satisfies R. For the *Cat and Mouse* examples, we can have the formal

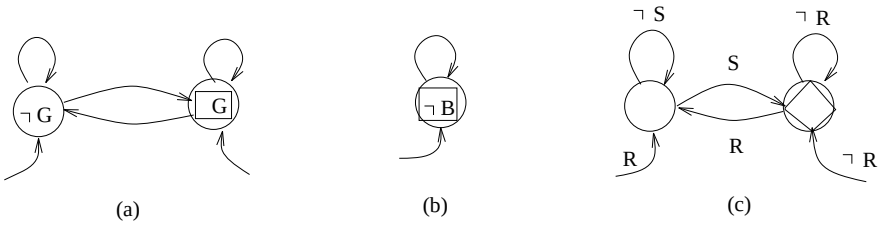

Figure 1: $\forall$-automata: (a) goal achievement or reachability (b) safety (c) bounded response

behavior specifications shown in Fig. 2.

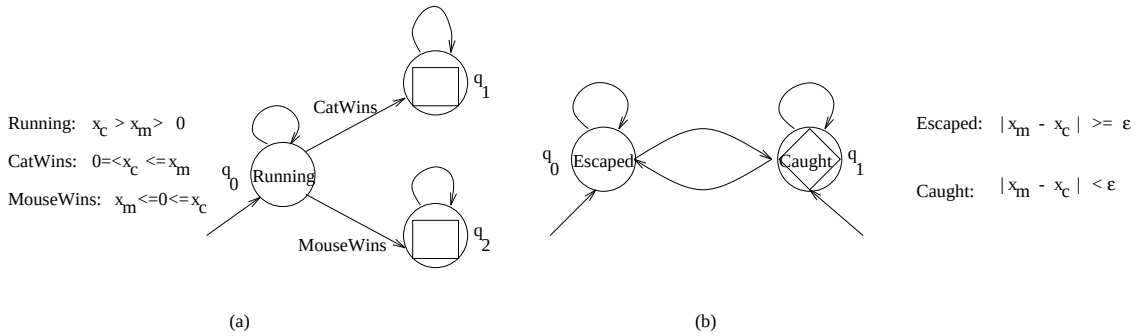

Figure 2: (a) Either the cat wins or the mouse wins (b) The cat catches the mouse persistently

3.2 System Verification

Given a constraint net model of a discrete- or continuous-time dynamic system, and a $\forall$-automaton specification of a desired behavior, a formal method is developed here for verifying that the constraint net exhibits its desired behavior.

Let CN be a constraint net model of a dynamic system, whose behavior is a set of traces. Let $\{\varphi\}CN\{\psi\}$ denote the validity of the consecutive condition: for every trace v of the behavior of CN,

- $\forall t > \mathbf{0}, \exists \delta > 0, \forall 0 < \tau \leq \delta : v(t-\tau) \models \varphi$ implies $v(t) \models \psi$ and
- $\forall t, \exists \delta > 0, \forall t' \in t + \delta : v(t) \models \varphi$ implies $v(t') \models \psi$.

Clearly, if $\mathcal{T}$ is discrete, these two conditions are reduced to one: $\forall t > \mathbf{0}, v(pre(t)) \models \varphi$ implies $v(t) \models \psi$.

Let CN be a constraint net with the set of locations Lc and the set of states $\times_{Lc} A_l$, Θ be an assertion indicating the initial state of CN, and $\mathcal{A} = \langle Q, R, S, e, c\rangle$ be a $\forall$-automaton. A set of assertions $\{\alpha_q\}_{q\in Q}$ is called a set of *invariants* for CN and $\mathcal{A}$ iff

- *Initiality*: $\forall q \in Q.\Theta \wedge e(q) \rightarrow \alpha_q$.
- *Consecution*: $\forall q, q' \in Q.\{\alpha_q\}CN\{c(q,q') \rightarrow \alpha_{q'}\}$.

Given that $\{\alpha_q\}_{q\in Q}$ is a set of invariants for CN and $\mathcal{A}$, a set of partial functions $\{\rho_q\}_{q\in Q} : \times_{Lc} A_l \rightarrow \mathcal{R}^+$ is called a set of *Liapunov* functions for CN and $\mathcal{A}$ iff the following conditions are satisfied:

- *Definedness*: $\forall q \in Q : \ \alpha_q \rightarrow \exists w.\rho_q = w$.
- *Non-increase*: $\forall q \in Q, q' \in S$:

$$\{\alpha_q \wedge \rho_q = w\}CN\{c(q,q') \rightarrow \rho_{q'} \leq w\}.$$

- *Decrease*: Let t_c denote the current time, $\exists \epsilon > 0, \forall q \in Q, q' \in B$:

$$\{\alpha_q \wedge \rho_q = w \wedge t_c = t\}CN\{c(q,q') \rightarrow \frac{\rho_{q'} - w}{d(t_c, t)} \leq -\epsilon\}$$

Let the time structure be infinite with either discrete or continuous topology. We conclude that if the behavior of a constraint net CN is specifiable by a $\forall$-automaton $\mathcal{A}$ and the following requirements are satisfied the validity of $\mathcal{A}$ over CN is proved:

(I) Associate with each automaton-state $q \in Q$ an assertion α_q, such that $\{\alpha_q\}_{q\in Q}$ is a set of invariants for CN and $\mathcal{A}$.

(L) Associate with each automaton-state $q \in Q$ a partial function $\rho_q : \times_{Lc} A_l \rightarrow \mathcal{R}^+$, such that $\{\rho_q\}_{q\in Q}$ is a set of Liapunov functions for CN and $\mathcal{A}$.

As in [3], the verification rules (I) and (L) are sound and complete, i.e. $\mathcal{A}$ accepts CN iff there exist a set of invariants and Liapunov functions.

Theorem 1 *Let the time structure be infinite with either discrete or continuous topology. If the behavior of a constraint net CN is specifiable by a $\forall$-automaton $\mathcal{A}$, verification rules (I) and (L) are sound and complete.*

Proof: (Sketch) Apply the method of continuous induction [2]. The detailed proof is shown in Appendix A of the extended version of this paper. □

We illustrate this verification method by the *Cat and Mouse* examples. Consider the first *Cat and Mouse* example adopted from [1]. We show that the constraint net model CM_1 in section 2 satisfies the behavior specification in Fig. 2(a).

First of all, the $\forall$-automaton in Fig. 2(a) is not complete. To make it complete, add an 'error' state $q_E \in B$, with $e(q_E) = false$, $c(q_E, q_E) = true$ and $c(q_0, q_E) = x_c < 0$. It is easy to see that CM_1 is specifiable by the complete $\forall$-automaton.

Secondly, associate with q_0, q_1, q_2, q_E assertions $Running$, $CatWins$, $MouseWins$ and $false$ respectively. Note that

$$\{x_c > x_m > 0\}CM_1\{x_c < 0 \rightarrow false\}$$

since x_c is continuous. (Imagine that if the cat jumps, it may not catch the mouse but hit the wall instead. Fortunately, this is not the case here.) Therefore, the set of assertions is a set of invariants.

Thirdly, associate with q_0, q_1, q_2, q_E the same function: $\rho : \mathcal{R} \times \mathcal{R} \times \mathcal{B} \rightarrow \mathcal{R}^+$, such that $\rho(x_c, x_m, 0) = 0$ and $\rho(x_c, x_m, 1) = -(\frac{x_m}{V_m} + \frac{x_c}{V_c})$. Clearly, ρ is decreasing at q_0 with rate 2 and q_E can never be reached. Therefore, it is a Liapunov function.

If we remove the square □ from node q_2 in Fig. 2(a), i.e. $q_2 \in B$, the modified behavior specification states that "the cat always wins". Clearly, not every trace of the behavior of CM_1 satisfies this specification. However, if the initial state satisfies $\frac{X_c}{V_c} > \frac{X_m}{V_m}$, in addition to $X_c > X_m > 0$, we can prove that "the cat always wins".

To see this, let $\Delta = \frac{X_c}{V_c} - \frac{X_m}{V_m}$ and let Inv denote $\frac{x_c}{V_c} - \frac{x_m}{V_m} = \Delta$. Associate with q_0, q_1, q_2, q_E assertions $Running \wedge Inv$, $CatWins$, $false$ and $false$ respectively. Note that

$$\{Running \wedge Inv\}CM_1\{Running \rightarrow Running \wedge Inv\}$$

since the derivative of $\frac{x_c}{V_c} - \frac{x_m}{V_m}$ is 0 given that $Running$ is satisfied; and

$$\{Running \wedge Inv\}CM_1\{MouseWin \rightarrow false\}$$

since x_m is continuous. Therefore, the set of assertions is a set of invariants.

Associate with q_0, q_1, q_2, q_E the same function: $\rho : \mathcal{R} \times \mathcal{R} \times \mathcal{B} \rightarrow \mathcal{R}^+$, such that $\rho(x_c, x_m, 0) = 0$ and $\rho(x_c, x_m, 1) = -(\frac{x_m}{V_m} + \frac{x_c}{V_c})$. Again, it is a Liapunov function.

Consider the second *Cat and Mouse* example, in which the motion of the mouse is unknown, but the cat tries to catch the mouse anyhow. Clearly, not every trace of the behavior of the constraint net CM_2 satisfies the behavior specification in Fig. 2(b). For example, if $\dot{x}_c = \dot{x}_m$ all the time, the distance between the cat and the mouse will be constant and the cat can never catch the mouse. However, suppose the mouse is short-sighted, i.e. it can only see the cat if their distance $|x_m - x_c| < \delta < \epsilon$, and when it does not see the cat, it will stop running within time τ.

The short-sighted property of the mouse is equivalent to adding the following assumption to CM_2:

$$\{|x_m - x_c| \geq \delta \wedge \dot{x}_m = 0\}CM_2\{|x_m - x_c| \geq \delta \rightarrow \dot{x}_m = 0\}$$

i.e. the mouse will not run if it does not see the cat. The maximum running time property of the mouse is equivalent to adding the following assumption to CM_2: let l_t be the time left for the mouse to run when it does not see the cat,

$$\{|x_m - x_c| < \delta\}CM_2\{|x_m - x_c| \geq \delta \wedge \dot{x}_m \neq 0 \rightarrow l_t \leq \tau\}$$

and

$$\{|x_m - x_c| \geq \delta \wedge \dot{x}_m \neq 0 \wedge l_t = l \wedge t_c = t\}CM_2\{|x_m - x_c| \geq \delta \wedge \dot{x}_m \neq 0 \rightarrow l_t \leq l - d(t_c, t)\}.$$

We show that no matter how fast the mouse may run, the cat tracks down the mouse infinitely often (including the case in which the mouse is caught permanently).

In order to prove this claim, we may decompose the automaton-state q_0 in Fig. 2(b) into two automaton-states q_{00} and q_{01} as shown in Fig. 3. This automaton is not complete. To make it complete, add an 'error' state $q_E \in B$ with $e(q_E) = false$, $c(q_E, q_E) = true$ and $c(q_{00}, q_E) = Escape$.

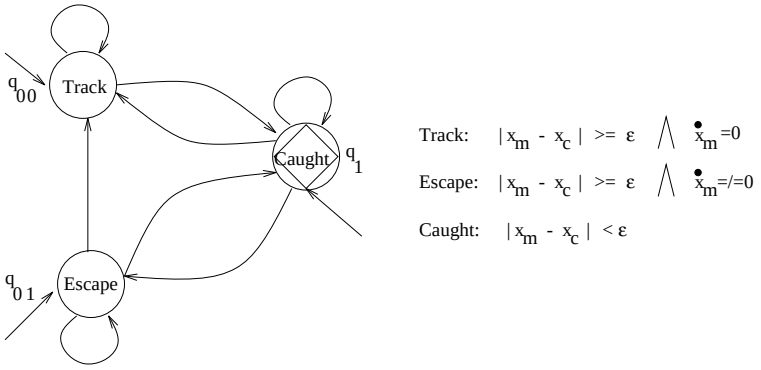

Figure 3: A refinement of the cat-mouse specification

Associate with automaton-states q_{00}, q_{01}, q_1, q_E assertions $Track, Escape, Caught$ and $false$ respectively. Note that

$$\{Track\}CM_2\{Escape \rightarrow false\}$$

because of the short-sighted property of the mouse. Therefore, the set of assertions is a set of invariants.

Let $d_m \in \mathcal{R}^+$ be the maximum distance between the cat and the mouse. Associate with automaton-states q_{00}, q_{01}, q_1, q_E functions $\rho_{q_{00}}$, $\rho_{q_{01}}$, ρ_{q_1} and ρ_{q_E} respectively, where

$$\rho_{q_{00}} = (x_m - x_c)^2, \;\; \rho_{q_{01}} = d_m^2 + l_t, \;\; \rho_{q_1} = \rho_{q_E} = d_m^2 + \tau.$$

The feedback control law of the cat guarantees that $\rho_{q_{00}}$ decreases at q_{00} at a rate not less than $2k\epsilon^2$. The maximum running time property of the mouse guarantees that $\rho_{q_{01}}$ decrease at q_{01} at a rate not less than 1. Therefore, the set of functions is a set of Liapunov functions.

4 Constraint-Based Dynamic Systems

In this section, we first explore the relationship between a constraint solver and its desired behavior specification, then define constraint-based dynamic systems as a generalization of constraint solvers.

4.1 Dynamic process and constraint solver

Constraint satisfaction can be seen as a dynamic process that approaches the solution set of the constraints asymptotically, and a constraint solver, modeled by a constraint net, exhibits this required behavior [8]. Here we briefly introduce some related concepts.

Let $\langle A, d\rangle$ be a domain. Given a point $a \in A$ and a subset $A^* \subset A$, the *distance* between a and A^* is defined as $d(a, A^*) = \inf_{a^* \in A^*}\{d(a, a^*)\}$. For any $\epsilon > 0$, the *ϵ-neighborhood* of A^* is defined as $N^\epsilon(A^*) = \{a | d(a, A^*) < \epsilon\}$; it is *strict* if it is a strict superset of A^*. For a function $v : T \rightarrow A$ from a total order T, v *approaches* A^* iff $\forall\epsilon\exists t_0\forall t \geq t_0 : d(v(t), A^*) < \epsilon$.

Let S be a domain indicating a state space and $\mathcal{T}$ be a time structure which can be either discrete or continuous. A *dynamic process* p is a function $p : S \rightarrow S^{\mathcal{T}}$ with $p(s)(\mathbf{0}) = s, \forall s \in S$. For any subset $S^* \subset S$, let $\phi_p(S^*) = \{p(s)(t) | s \in S^*, t \in \mathcal{T}\}$. S^* is an *equilibrium* of p iff $\phi_p(S^*) = S^*$. S^* is a *stable equilibrium* of p iff S^* is an equilibrium and $\forall\epsilon\exists\delta : \phi_p(N^\delta(S^*)) \subseteq N^\epsilon(S^*)$. S^* is an *attractor* of p iff there exists a strict ϵ-neighborhood $N^\epsilon(S^*)$ such that $\forall s \in N^\epsilon(S^*)$, $p(s)$ approaches S^*; S^* is an *attractor in the large* iff $\forall s \in S$,

$p(s)$ approaches S^*. If S^* is an attractor (in the large) and S^* is a stable equilibrium, S^* is an *asymptotically* stable equilibrium (in the large).

Let $C = \{C_i\}_{i \in I}$ be a set of constraints, whose solution $sol(C) = \{s | \forall i \in I : C_i(s)\}$ is a subset of a state space S. A *constraint solver* for C is a constraint net CS whose semantics is a dynamic process $p : S \rightarrow S^{\mathcal{T}}$ with $sol(C)$ as an asymptotically stable equilibrium. CS *solves* C *globally* iff $sol(C)$ is an asymptotically stable equilibrium in the large.

We have discussed two types of constraint solvers: state transition systems and state integration systems. Various discrete and continuous constraint methods have been presented, and also analyzed using Liapunov functions [8].

4.2 Constraint-based computation and control

Given a set of constraints C, let C^ϵ denote the assertion which is true on the ϵ-neighborhood of its solution set, $N^\epsilon(sol(C)) \subset S$, and let $\mathcal{A}(C^\epsilon; \Box)$ denote the $\forall$-automaton in Fig. 4(a). Clearly, CS solves C iff there exists an initial condition $\Theta \supset sol(C)$ such that $\forall \epsilon : CS(\Theta) \models \mathcal{A}(C^\epsilon; \Box)$; CS solves C globally when $\Theta = S$. We call $\mathcal{A}(C^\epsilon; \Box)$ an *open specification* of the set of constraints C. Note that it is important to have

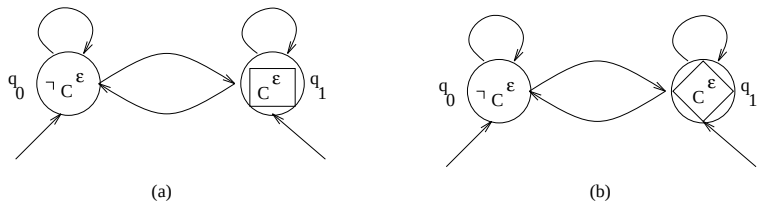

Figure 4: Specification for (a) Constraint solver (b) Constraint-based dynamic system

open specifications, otherwise, if we replace C^ϵ with $sol(C)$, a constraint solver for C may never satisfy the specification, since it may take *infinite* time to approach $sol(C)$.

However, requiring the integration of a controller with its environment to be a constraint solver is still too stringent for a control problem, with disturbance and uncertainty in its environment. If we consider the solution set of a set of constraints as the 'goal' for the controller to achieve, one relaxed requirement for the controller is to make the system stable at the goal. In other words, if the system diverges from the goal by some disturbance, the controller should always be able to regulate the system back to its goal. We call a system CB *constraint-based* w.r.t. a set of constraints C, iff $\forall \epsilon : CB \models \mathcal{A}(C^\epsilon; \Diamond)$ where $\mathcal{A}(C^\epsilon; \Diamond)$ denotes the $\forall$-automaton in Fig. 4(b). In other words, a dynamic system is constraint-based iff it approaches the solution set of the constraints infinitely often.

We may relax this condition further and define constraint-based systems with errors. We call a system CB *constraint-based* w.r.t. a set of constraints C with error δ, iff $\forall \epsilon > \delta : CB \models \mathcal{A}(C^\epsilon; \Diamond)$; δ is called the *steady-state error* of the system. Normally, steady-state errors are caused by uncertainty and disturbance of the environment. For example, the second cat-mouse system CM_2 is a constraint-based system with steady-state error δ, which is the radius of the mouse sensing range.

If $\mathcal{A}(C^\epsilon; \Box)$ is considered as an open specification of a constraint-based *computation* for a closed system, $\mathcal{A}(C^\epsilon; \Diamond)$ can been seen as an *open specification* of a constraint-based *control* for an open or embedded system.

We have developed a systematic approach to control synthesis from requirement specifications [8]. In particular, requirement specifications impose constraints over a system's global behavior and controllers can be synthesized as embedded constraint solvers which solve constraints over time. By exploring a relation between constraint satisfaction and dynamic systems via constraint methods, discrete/continuous constraint solvers or constraint-based controllers are derived.

We have developed here a behavior specification language and a formal verification method for dynamic systems. With this approach, control synthesis and system verification are coupled via requirement specifications and Liapunov functions. If we consider a Liapunov function for a set of constraints as a measurement of the degree of satisfaction, this function can be used for both control synthesis and system verification.

5 Conclusion and Related Work

We have presented a formal specification language, called generalized ∀-automata, for desired behaviors of dynamic systems. We have also presented a formal verification method, using generalized Liapunov functions, for checking that a dynamic system exhibits its desired behavior. A constraint-based dynamic system can be modeled by a constraint net, whose desired behavior can be specified by a ∀-automaton. The Liapunov functions for a given constraint specification can be used for both control synthesis and system verification.

Some related work has been done recently along these lines. Nerode and Kohn have proposed the notion of open specification for control systems [4]. Saraswat et al. have developed a family of timed concurrent constraint languages for modeling and specification of discrete dynamic systems [5]. Problems on specification and verification of hybrid dynamic systems have become a new challenge to traditional control system design and traditional programming methodologies [1]. Our work is unique in that we distinguish the executable (modeling) and logical (requirement) specifications, and develop the model checking technique based on properties of dynamic systems.

Acknowledgement

We wish to thank Nick Pippenger and Runping Qi for valuable discussions. This research was supported by the Natural Sciences and Engineering Research Council and the Institute for Robotics and Intelligent Systems.

References

[1] 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.

[2] G. F. Khilmi. *Qualitative Methods in the Many Body Problem*. Science Publishers Inc. New York, 1961.

[3] 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.

[4] 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. Springer-Verlag, 1993.

[5] V. Saraswat, R. Jagadeesan, and V. Gupta. Programming in timed concurrent constraint languages. In B. Mayoh, E. Tyugu, and J. Penjam, editors, *Constraint Programming*, NATO Advanced Science Institute Series, Series F: Computer And System Sciences. 1994.

[6] W. Thomas. Automata on infinite objects. In Jan Van Leeuwen, editor, *Handbook of Theoretical Computer Science*. MIT Press, 1990.

[7] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56:72 – 99, 1983.

[8] Y. Zhang and A. K. Mackworth. Constraint programming in constraint nets. In *First Workshop on Principles and Practice of Constraint Programming*, pages 303–312, 1993.

[9] Y. Zhang and A. K. Mackworth. Design and analysis of embedded real-time systems: An elevator case study. Technical Report 93-4, Department of Computer Science, University of British Columbia, February 1993.

[10] Y. Zhang and A. K. Mackworth. Constraint Nets: A semantic model for hybrid dynamic systems, 1994. Working Paper.

A Proof of Theorem 1

In order to prove this theorem, we shall introduce the method of continuous induction [2]. A property Γ is *inductive* on a time structure $\mathcal{T}$ iff Γ is satisfied at all $t < t_0 \in \mathcal{T}$ implies that Γ is satisfied at t_0, for all $t_0 \in \mathcal{T}$. Γ is *continuous* iff Γ is satisfied at $t_0 \in \mathcal{T}$ implies that $\exists \delta, \forall t \in t_0 + \delta$, Γ is satisfied at t. Clearly, when $\mathcal{T}$ is discrete, any property is continuous. The theorem of continuous induction [2] says:

Theorem 2 *If the property Γ is inductive and continuous on a time structure $\mathcal{T}$ and Γ is satisfied at $\mathbf{0}$, Γ is satisfied at all $t \in \mathcal{T}$.*

Proof: Clearly, the theorem reduces to conventional induction when $\mathcal{T}$ is discrete. Here we concentrate on the continuous case. We call a time point $t \in \mathcal{T}$ *regular* iff Γ is satisfied at all t', $\mathbf{0} \leq t' \leq t$. Let T denote the set of all regular time points. T is not empty since Γ is satisfied at $\mathbf{0}$. We prove the theorem by contradiction, i.e. assume that Γ is not satisfied at all $t \in \mathcal{T}$. Therefore, T is bounded above; let $t_0 = \bigvee T \in \mathcal{T}$ be the least upper bound of T. Since t_0 is the least upper bound, it follows that Γ is satisfied at all t, $\mathbf{0} \leq t < t_0$. Since Γ is inductive, it is satisfied at time t_0. Since Γ is also continuous, we can find a $\delta > 0$ such that Γ is satisfied at all $t \in t_0 + \delta$. Therefore t is a regular time point $\forall t \in t_0 + \delta$. However, this contradicts the fact that t_0 is the least upper bound of the set T. □

Using the method of continuous induction, we obtain the following two lemmas.

Lemma 1 *Suppose CN is specifiable by $\mathcal{A}$ and $\{\alpha_q\}_{q \in Q}$ is a set of invariants for CN and $\mathcal{A}$. For any trace $v \in [\![CN]\!]$ and any run r of $\mathcal{A}$ over v, $v(t) \models \alpha_{r(t)}, \forall t \in \mathcal{T}$.*

Proof: We prove that the property $v(t) \models \alpha_{r(t)}$ is satisfied at $\mathbf{0}$ and is both inductive and continuous.

- Initiality: Since $v(\mathbf{0}) \models \Theta$ and $v(\mathbf{0}) \models e(r(\mathbf{0}))$, $v(\mathbf{0}) \models \Theta \wedge e(r(\mathbf{0}))$. According to the *Initiality* condition of invariants, $v(\mathbf{0}) \models \alpha_{r(\mathbf{0})}$.
- Inductivity: Suppose $v(t) \models \alpha_{r(t)}$ is saisfied at $\mathbf{0} \leq t < t_0$. Since r is a run over v, $\exists q \in Q$ and $\delta_1 > 0, \forall 0 < \tau \leq \delta_1$, $r(t_0 - \tau) = q$ and $v(t_0) \models c(q, r(t_0))$. According to the *Consecution* condition of invariants, $\exists \delta_2 > 0, \forall 0 < \tau \leq \delta_2$, $v(t_0 - \tau) \models \alpha_q$ implies $v(t_0) \models c(q, r(t_0)) \rightarrow \alpha_{r(t_0)}$. Therefore, $\forall 0 < \tau \leq \min(\delta_1, \delta_2)$, $r(t - \tau) = q$, $v(t_0 - \tau) \models \alpha_q$ (assumption), $v(t_0) \models c(q, r(t_0)) \rightarrow \alpha_{r(t_0)}$ and $v(t_0) \models c(q, r(t_0))$. Thus, $v(t_0) \models \alpha_{r(t_0)}$.
- Continuity: Suppose $v(t_0) \models \alpha_{r(t_0)}$. Since r is a run over v, $\exists q \in Q$ and $\delta_1 > 0, \forall t \in t_0 + \delta_1$, $r(t) = q$ and $v(t) \models c(r(t_0), q)$. According to the *Consecution* condition of invariants, $\exists \delta_2 > 0, \forall t \in t_0 + \delta_2$, $v(t_0) \models \alpha_{r(t_0)}$ implies $v(t) \models c(r(t_0), q) \rightarrow \alpha_q$. Therefore, $\forall t \in t_0 + \min(\delta_1, \delta_2)$, $r(t) = q$, $v(t_0) \models \alpha_{r(t_0)}$ (assumption), $v(t) \models c(r(t_0), q) \rightarrow \alpha_q$ and $v(t) \models c(r(t_0), q)$. Thus, $\forall t \in t_0 + \min(\delta_1, \delta_2)$, $v(t) \models \alpha_{r(t)}$.

□

Given any interval of time I, let $\mu(I) = \int_I dt$ be the measurement of the interval. Given a property Γ, let $\mu(I_\Gamma) = \int_I \Gamma(t) dt$ be the measurement of time points at which Γ is satisfied.

Lemma 2 *Suppose CN is specifiable by $\mathcal{A}$ and $\{\rho_q\}_{q \in Q}$ is a set of Liapunov functions for CN and $\mathcal{A}$. For any trace $v \in [\![CN]\!]$ and any run r of $\mathcal{A}$ over v, the following property holds: For any interval I with only bad and stable automaton-states, i.e. for all $t \in I, r(t) \notin R$, the measurement on the bad states is finite, i.e. $\mu(I_{r \in B}) < \infty$.*

Proof: For any run r over v and any interval I with only bad and stable states, ρ on I is nonincreasing, i.e. for any $t_1 < t_2 \in I$, $\rho_{r(t_1)}(v(t_1)) \geq \rho_{r(t_2)}(v(t_2))$, and the decreasing speed at the bad states is no less than ϵ. Let m be the upper bound of $\{\rho_{r(t)}(v(t)) | t \in I\}$. Since $\rho_q \geq 0$, $\mu(I_{r \in B}) \leq m/\epsilon < \infty$. □

Proof of Theorem 1: Soundness is derived from Lemma 1 and Lemma 2: If there exist a set of invariants and a set of Liapunov functions, for any run r, if any automaton-state in R appears infinitely many times in r, r is accepting; otherwise there is a time point t_0, the interval $I = \{t \in \mathcal{T} | t \geq t_0\}$ has only bad and stable automaton-states. Since $\mu(I_{r \in B})$ is finite, all the automaton-states appearing infinitely many times belong to S.

On the other hand, if $\mathcal{A}$ accepts CN, there exists a set of invariants and a set of Liapunov functions. The set of invariants can be constructed as follows. Let $CN(s,q,s,q')$ denote the consecutive condition. Formally, $CN(s,q,s',q')$ iff

$$\exists v,r,t,t' : (t' > t) \bigwedge (v(t) = s \wedge r(t) = q \wedge v(t') = s' \wedge r(t') = q') \bigwedge$$

$$[\exists t_0 : t \leq t_0 \leq t' \wedge (\forall \tau, t < \tau < t_0 : r(\tau) = q) \wedge (\forall \tau, t_0 < \tau < t' : r(\tau) = q')].$$

Let the set of invariants $\{\alpha_q\}_{q \in Q}$ be defined as the fixpoint of the following equation:

$$\alpha_{q'}(s') = (\exists q, s : \alpha_q(s) \wedge CN(s,q,s',q') \wedge c(q,q')(s')) \bigvee (\Theta(s') \wedge e(q')(s')).$$

It is easy to check that $\{\alpha_q\}_{q \in Q}$ is a set of invariants and $s \models \alpha_q$ iff the pair $\langle s,t \rangle$ is reachable, i.e. $\exists v,r,t : v(t) = s \wedge r(t) = q$. The set of Liapunov functions can be constructed as follows, given the above constructed invariants:

- $\forall q \notin R$ and $s \models \alpha_q$, the Liapunov function is defined as follows. For any v,r,t, let

  $$I = \{t' | t' > t, \forall \tau, t < \tau \leq t' : r(\tau) \notin R\}$$

  be an interval with only bad and stable states and $L = \mu(I_{r \in B})$ be the measurement of bad states. Let $\rho_q(s)$ be the longest measurement of bad states, i.e.

  $$\rho_q(s) = \sup\{L | v(t) = s \wedge r(t) = q\}.$$

- $\forall q \in R$ and $s \models \alpha_q$, $\rho_q(s) = sup\{\rho_q(s) | q \notin R, s \models \alpha_q\} + 1$.

It is easy to check that $\{\rho_q\}_{q \in Q}$ is a set of Liapunov functions. $\Box$