Specification and Verification of Constraint-Based Dynamic Systems

Ying Zhang and Alan K. Mackworth*

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

Abstract. Constraint satisfaction can be seen as a dynamic process that approaches the solution set of the given constraints asymptotically [6]. Constraint programming is seen as creating a dynamic system with the required 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 given constraints persistently. 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 requirements specification.

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 [6]. Under this view, constraint programming is the creation of a dynamic system with the required 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 have also been examined [6].

Generalizing, here we consider a constraint-based dynamic system as a dynamic system which approaches the solution set of the given constraints persistently. 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.

* Shell Canada Fellow, Canadian Institute for Advanced Research

In A. Borning (ed.) "Principles and Practice of Constraint Programming" Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1994, pp. 229-242.

The required properties 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 persistently.

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

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

The rest of this 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 required properties of dynamic systems. Section 4 characterizes constraint-based dynamic systems and requirements specification. 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, 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 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 linearly ordered set with a least element 0 and d is a metric satisfying that for all $t_0 \leq t_1 \leq t_2, d(t_0, t_2) = d(t_0, t_1) + d(t_1, t_2)$. We will use $\mathcal{T}$ to denote time structure $\langle \mathcal{T}, d \rangle$ if no ambiguity arises. In this paper, we consider a time structure $\mathcal{T}$ with the following properties: (1) $\mathcal{T}$ is *infinite*, i.e., $\sup_{t \in \mathcal{T}}\{d(0,t)\} = \infty$, and (2) $\mathcal{T}$ is *complete*, i.e., if $T \subset \mathcal{T}$ has an upper bound, T has a least upper bound. $\mathcal{T}$ can be either discrete or continuous. For example, the set of natural numbers defines discrete time and the set of nonnegative real numbers defines continuous time.

Let X be a metric space representing a discrete or continuous *domain*. A *trace* $v : \mathcal{T} \rightarrow X$ is a function from time to a domain.

2.2 Constraint Nets: a model for dynamic systems

We have developed a semantic model, Constraint Nets, for general (hybrid) dynamic systems [8]. We have used the Constraint Net model as an abstract target machine for constraint programming languages [6], 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 cell, 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 continuous time and any state automaton with an initial state defines a transduction on discrete time.

A location l is the *output location* of a transduction F iff it connects to the output port of F; l is an *input location* of F iff it connects to an input port of F. Let CN be a constraint net. A location is an *output location* of CN if it is an output location of some transduction in CN; it is otherwise an *input location* of CN. CN is *open* if there are input locations; it is otherwise *closed*.

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, denoted $[\![CN]\!]$, is a "solution" of the set of equations [8], which is a transduction from input to output traces. The *behavior* of a dynamic system is defined as a set of possible input/output traces produced by the system. We will also use $[\![CN]\!]$ to denote the behavior of a dynamic system modeled by CN if no ambiguity arises.

We have modeled two types of constraint solver, state transition systems and state integration systems, in constraint nets. The former models discrete dynamic processes and the latter models continuous dynamic processes [6]. Hybrid dynamic systems, with both discrete and continuous components, can also be modeled in constraint nets [7, 8].

We illustrate the constraint net modeling with two simple examples. Without loss of generality, let time be the set of nonnegative real numbers $\mathcal{R}^+$ and domains be the set (or product) of real numbers $\mathcal{R}$.

Consider 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 set of equations CM_1:

$$x_c = \int(X_c)(V_c \cdot c), \quad x_m = \int(X_m)(V_m \cdot c), \quad c = (x_c > x_m) \wedge (x_m > 0)$$

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. This is a closed system. 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 requirements 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 [6] 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 the following set of 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 $\forall$-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 required properties and to verifying the relationship between the behavior of a dynamic system and its requirements specification. A trace $v : \mathcal{T} \to X$ 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 behavior of a system as a language and a specification as an automaton, then verification is to check the inclusion relation between the language of the system and the language accepted by the automaton.

$\forall$-automata [3] 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. In this section, we generalize $\forall$-automata to specify languages composed of traces on continuous as well as discrete time, and modify the formal verification method [3] by generalizing both Liapunov functions [6] and the method of continuous induction [2].

3.1 Requirements specification

Let an *assertion* be a logical formula defined on a domain X, i.e., an assertion α for a value $x \in X$ will be evaluated to either *true*, denoted $x \models \alpha$, or *false*, denoted $x \not\models \alpha$.

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.

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

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

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., $Inf(r) = \{q | \forall t \exists t_0 \geq t, r(t_0) = q\}$. 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; $\mathcal{A}$ *accepts* a behavior $\mathcal{B}$, written $\mathcal{B} \models \mathcal{A}$, iff $\forall v \in \mathcal{B}$, $v \models \mathcal{A}$.

One of the advantages of using automata as a specification language is its 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 $\varphi_i, i = 1 \cdots n$, 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', $c(q, q') = false$.

This type of automaton is powerful enough to specify various qualitative properties. Some typical required properties are shown in Fig. 1: (a) accepts a trace which satisfies $\neg G$ only in finite time, (b) accepts a trace which never satisfies B, and (c) accepts a trace which will satisfy S in the finite future whenever it satisfies R.

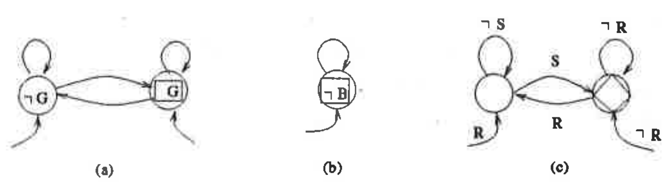

Fig. 1. $\forall$-automata: (a) reachability (b) safety (c) bounded response

For the *Cat and Mouse* examples, we can have the formal requirements specifications shown in Fig. 2.

3.2 Behavior verification

Given a constraint net model of a discrete- or continuous-time dynamic system, the behavior of the system is obtained from a "solution" of the set of equations denoted by the model. Given a behavior and a $\forall$-automata specification of requirements, a formal method is developed here for verifying that the behavior satisfies its requirements specification.

For any trace $v : \mathcal{T} \rightarrow X$, let $\{\varphi\}v\{\psi\}$ denote the validity of the following two consecutive conditions:

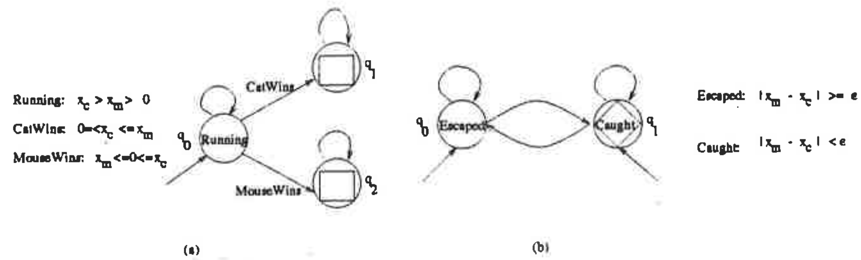

Fig. 2. (a) Either the cat or the mouse wins (b) The cat catches the mouse persistently

- $\{\varphi\}v^-\{\psi\}$: for all $t > 0$, $\exists t' < t, \forall t'', t' \leq t'' < t, v(t'') \models \varphi$ implies $v(t) \models \psi$.
- $\{\varphi\}v^+\{\psi\}$: for all t, $v(t) \models \varphi$ implies $\exists t' > t, \forall t'', t < t'' < t', v(t'') \models \psi$.

Let $\mathcal{B}$ be a behavior with time $\mathcal{T}$ and domain X, $\Theta = \{v(0) | v \in \mathcal{B}\}$ be the set of initial values of $\mathcal{B}$, 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 $\mathcal{B}$ and $\mathcal{A}$ iff

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

Without loss of generality, we assume that time is encoded in domain X by $t_c : X \rightarrow \mathcal{T}$. Given that $\{\alpha_q\}_{q \in Q}$ is a set of invariants for $\mathcal{B}$ and $\mathcal{A}$, a set of partial functions $\{\rho_q\}_{q \in Q} : X \rightarrow \mathcal{R}^+$ is called a set of *Liapunov functions* for $\mathcal{B}$ and $\mathcal{A}$ iff the following conditions are satisfied:

- *Definedness*: $\forall q \in Q, \;\; \alpha_q \rightarrow \exists w, \rho_q = w$.
- *Non-increase*: $\forall v \in \mathcal{B}$, $\forall q \in S, q' \in Q$,

$$\{\alpha_q \wedge \rho_q = w\}v^-\{c(q, q') \rightarrow \rho_{q'} \leq w\}$$

  and $\forall q \in Q, q' \in S$,

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

- *Decrease*: $\forall v \in \mathcal{B}$, $\exists \epsilon > 0$, $\forall q \in B, q' \in Q$,

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

  and $\forall q \in Q, q' \in B$,

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

We conclude that if the behavior of a system $\mathcal{B}$ is specifiable by a $\forall$-automaton $\mathcal{A}$ and the following requirements are satisfied, the validity of $\mathcal{A}$ over $\mathcal{B}$ 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 $\mathcal{B}$ and $\mathcal{A}$.
(L) Associate with each automaton-state $q \in Q$ a partial function $\rho_q : X \to \mathcal{R}^+$, such that $\{\rho_q\}_{q\in Q}$ is a set of Liapunov functions for $\mathcal{B}$ and $\mathcal{A}$.

Theorem 1 *If $\mathcal{B}$ is specifiable by $\mathcal{A}$, and both (I) and (L) are satisfied, $\mathcal{B} \models \mathcal{A}$.*

Proof: (Sketch, details in appendix) Use the method of continuous induction to show that $\forall v \in \mathcal{B}$ and a run r of $\mathcal{A}$ over v, $v(t) \models \alpha_{r(t)}$, $\forall t \in \mathcal{T}$. □

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 requirements specification in Fig. 2(a).

Associate with q_0, q_1 and q_2 assertions *Running*, *CatWins* and *MouseWins*, respectively. Therefore, the set of assertions is a set of invariants.

Associate with q_0, q_1 and q_2 the same function $\rho : \mathcal{R} \times \mathcal{R} \times \{0, 1\} \to \mathcal{R}^+$, 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. Therefore, it is a Liapunov function.

The behavior of CM_1 is specifiable by the automaton in Fig. 2(a) since x_c and x_m are continuous. Therefore, CM_1 satisfies the required property.

If we remove the square □ from node q_2 in Fig. 2(a), i.e., $q_2 \in B$, the modified requirements specification declares that "the cat always wins." Not every trace of the behavior of CM_1 satisfies this specification. However, if the initial value $\langle X_c, X_m\rangle$ 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 and q_2 assertions *Running* $\wedge$ *Inv*, *CatWins* and *false*, respectively. Note that for all $v \in [\![CM_1]\!]$,

$$\{Running \wedge Inv\}v\{Running \to Running \wedge Inv\}$$

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

$$\{Running \wedge Inv\}v\{MouseWin \to false\}$$

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

Associate with q_0, q_1 and q_2 the same function $\rho : \mathcal{R} \times \mathcal{R} \times \{0, 1\} \to \mathcal{R}^+$, 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 requirements 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 may 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: for all $v \in [\![CM_2]\!]$,

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

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, for all $v \in [\![CM_2]\!]$,

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

and

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

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

In order to prove this claim, we decompose the automaton-state q_0 in Fig. 2(b) into two automaton-states q_{00} and q_{01} as shown in Fig. 3.

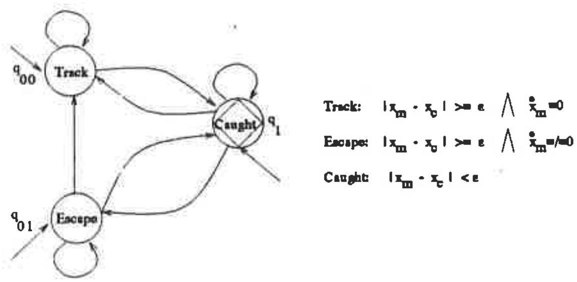

Fig. 3. A refinement of the cat-mouse specification

Associate with automaton-states q_{00}, q_{01} and q_1 assertions *Track*, *Escape* and *Caught*, respectively. Note that $\forall v \in [\![CM_2]\!]$, $\{Track\} v \{Escape \rightarrow false\}$. Therefore, $[\![CM_2]\!]$ is specifiable and the set of assertions is a set of invariants.

Let $V_m \in \mathcal{R}^+$ be the maximum speed of the mouse and $D_m = V_m\tau + \delta$. Associate with automaton-states q_{00}, q_{01} and q_1 functions $\rho_{q_{00}}$, $\rho_{q_{01}}$ and ρ_{q_1}, respectively, where

$$\rho_{q_{00}} = (x_m - x_c)^2, \;\; \rho_{q_{01}} = D_m^2 + l_t, \;\; \rho_{q_1} = 0.$$

The feedback control law of the cat guarantees that $\rho_{q_{00}}$ decreases at q_{00} at a rate no 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 no less than 1. Therefore, ρ decreases at q_0 with minimum rate $\min(2k\epsilon^2, 1)$. We can check that 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 requirements specification, then define constraint-based dynamic systems as a generalization of constraint solvers.

4.1 Constraint solver

Constraint satisfaction can be seen as a dynamic process that approaches the solution set of the given constraints asymptotically, and a constraint solver, modeled by a constraint net, satisfies this required property [6]. Here we briefly introduce some related concepts.

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

Given a dynamic process [6] $p : X \rightarrow (\mathcal{T} \rightarrow X)$ and $X^* \subset X$, let $\phi_p(X^*) = \{p(x)(t) | x \in X^*, t \in \mathcal{T}\}$. X^* is an *equilibrium* of p iff $\phi_p(X^*) = X^*$. X^* is a *stable equilibrium* of p iff X^* is an equilibrium and $\forall\epsilon\exists\delta, \phi_p(N^\delta(X^*)) \subseteq N^\epsilon(X^*)$. X^* is an *attractor* of p iff there exists a strict ϵ-neighborhood $N^\epsilon(X^*)$ such that $\forall x \in N^\epsilon(X^*)$, $p(x)$ approaches X^*; X^* is an *attractor in the large* iff $\forall x \in X$, $p(x)$ approaches X^*. If X^* is an attractor (in the large) and X^* is a stable equilibrium, X^* is an *asymptotically* stable equilibrium (in the large).

Let $C = \{C_i\}_{i \in I}$ be a set of constraints, whose solution $sol(C) = \{x | \forall i \in I, C_i(x)\}$ is a subset of domain X. A *constraint solver* for C is a constraint net CS whose semantics is a dynamic process $p : X \rightarrow (\mathcal{T} \rightarrow X)$ 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 [6].

4.2 Constraint-based requirements specification

Given a set of constraints C, let C^ϵ denote the assertion which is true on the ϵ-neighborhood of its solution set $N^\epsilon(sol(C))$, and let $\mathcal{A}(C^\epsilon; \Box)$ denote the $\forall$-automaton in Fig. 4(a). Using the asymptotic property of constraint solvers, we can verify that 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 Θ is the domain itself. We call $\mathcal{A}(C^\epsilon; \Box)$ an *open specification* of the set of constraints C. Note that it is important to have open specification, 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 $sol(C)$.

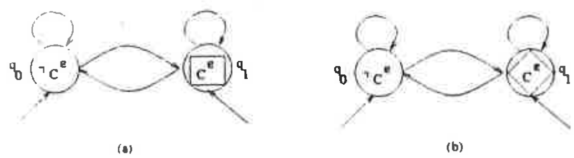

Fig. 4. Specification for (a) constraint solver (b) constraint-based dynamic system

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 there exists an initial condition $\Theta \supset sol(C)$ such that $\forall \epsilon, [\![CB(\Theta)]\!] \models \mathcal{A}(C^\epsilon; \Diamond)$ 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 given constraints persistently.

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 $\forall \epsilon > \delta, [\![CB(\Theta)]\!] \models \mathcal{A}(C^\epsilon; \Diamond)$; 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.

4.3 Constraint-based control and behavior verification

We have developed a systematic approach to control synthesis from requirements specification [6]. In particular, requirements specification imposes 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 requirements specification language and a formal verification method for dynamic systems. With this approach, control synthesis and behavior verification are coupled via requirements specification 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 behavior verification.

5 Conclusion and Related Work

We have presented a formal language, generalized ∀-automata, for specifying required properties of dynamic systems, and a formal method, based on generalized Liapunov functions, for verifying that the behavior of a dynamic system satisfies its requirements specification. A constraint-based dynamic system can be modeled by a constraint net, whose desired behavior can be specified by a ∀-automaton.

Some related work has also been done. 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 the specification and verification of hybrid dynamic systems have become a new challenge to both the traditional control systems design and the traditional programming methodology [1].

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. 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. A revised version will appear in a book with the same title in MIT Press, 1995.
7. 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.
8. Y. Zhang and A. K. Mackworth. Constraint Nets: A semantic model for hybrid dynamic systems, 1994. Accepted for TCS Special Issue on Hybrid Systems.

A Proof of Theorem 1

In order to prove this theorem, we shall introduce a method of continuous induction modified from [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 $\exists t_1 > t, \forall t, t_0 < t < t_1$, Γ is satisfied at t. We should notice that when $\mathcal{T}$ is discrete, any property is continuous. The theorem of continuous induction says:

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

Proof: We call a time point $t \in \mathcal{T}$ *regular* iff Γ is satisfied at all t', $0 \le t' \le t$. Let T denote the set of all regular time points. T is not empty since Γ is satisfied at 0. We prove the theorem by contradiction, i.e., assume that Γ is not satisfied at all $t \in \mathcal{T}$. Therefore, $T \subset \mathcal{T}$ is bounded above; let $t_0 = \bigvee T \in \mathcal{T}$ be the least upper bound of T ($\mathcal{T}$ is complete). Since t_0 is the least upper bound, it follows that Γ is satisfied at all t, $0 \le t < t_0$. Since Γ is inductive, it is satisfied at time t_0. Therefore, $t_0 \in T$.

Since $T \subset \mathcal{T}$, t_0 is not the greatest element in $\mathcal{T}$. Let $T' = \{t | t > t_0\}$. There are two cases: (1) if T' has a least element t', since Γ is inductive, $t' \in T$ is a regular time point. (2) otherwise, for any $t' \in T'$, $\{t | t_0 < t < t'\} \neq \emptyset$. Since Γ is also continuous, we can find a $t' \in T'$ such that Γ is satisfied at all $T'' = \{t | t_0 < t < t'\}$. Therefore, t is a regular time point $\forall t \in T''$. Both cases contradict 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 *Let $\{\alpha_q\}_{q \in Q}$ be invariants for $\mathcal{B}$ and $\mathcal{A}$. If r is a run of $\mathcal{A}$ over $v \in \mathcal{B}$, $\forall t \in \mathcal{T}, v(t) \models \alpha_{r(t)}$.*

Proof: We prove that the property $v(t) \models \alpha_{r(t)}$ is satisfied at 0 and is both inductive and continuous on any time structure $\mathcal{T}$.

- Initiality: Since $v(0) \models \Theta$ and $v(0) \models e(r(0))$, we have $v(0) \models \Theta \wedge e(r(0))$. According to the *Initiality* condition of invariants, we have $v(0) \models \alpha_{r(0)}$.
- Inductivity: Suppose $v(t) \models \alpha_{r(t)}$ is saisfied at $0 \le t < t_0$. Since r is a run over v, $\exists q \in Q$ and $t'_1 < t_0, \forall t, t'_1 \le t < t_0$, $r(t) = q$ and $v(t_0) \models c(q, r(t_0))$. According to the *Consecution* condition of the invariants, $\exists t'_2 < t_0, \forall t, t'_2 \le t < t_0$, $v(t) \models \alpha_q$ implies $v(t_0) \models c(q, r(t_0)) \rightarrow \alpha_{r(t_0)}$. Therefore, $\forall t, \max(t'_1, t'_2) \le t < t_0$, $r(t) = q$, $v(t) \models \alpha_q$ (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 $t'_1 > t_0, \forall t, t_0 < t < t'_1$, $r(t) = q$ and $v(t) \models c(r(t_0), q)$. According to the *Consecution* condition of the invariants, $\exists t'_2 > t_0, \forall t, t_0 < t < t'_2$, $v(t_0) \models \alpha_{r(t_0)}$ implies $v(t) \models c(r(t_0), q) \rightarrow \alpha_q$. Therefore, $\forall t, t_0 < t < \min(t'_1, t'_2)$, $r(t) = q$, $v(t_0) \models \alpha_{r(t_0)}$ (assumption), $v(t) \models c(r(t_0), q) \rightarrow \alpha_q$ and $v(t) \models c(r(t_0), q)$. Thus, $\forall t, t_0 < t < \min(t'_1, t'_2)$, $v(t) \models \alpha_{r(t)}$.

□

Given any interval I of time $\mathcal{T}$, 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 *Let $\{\alpha_q\}_{q\in Q}$ be invariants for $\mathcal{B}$ and $\mathcal{A}$ and r be a run of $\mathcal{A}$ over a trace $v \in \mathcal{B}$. If $\{\rho_q\}_{q\in Q}$ is a set of Liapunov functions for $\mathcal{B}$ and $\mathcal{A}$, then*

- $\rho_{r(t_2)}(v(t_2)) \leq \rho_{r(t_1)}(v(t_1))$ *when* $\forall t_1 \leq t \leq t_2, r(t) \in B \cup S$,
- $\frac{\rho_{r(t_2)}(v(t_2)) - \rho_{r(t_1)}(v(t_1))}{d(t_1,t_2)} \leq -\epsilon$ *when* $t_1 < t_2$ *and* $\forall t_1 \leq t \leq t_2, r(t) \in B$, *and*
- *for any interval I with only bad and stable automaton-states, $\mu(I_B)$ is finite.*

Proof: For any run r over v and for any segments $q^* : I \to Q$ of r with only bad and stable states, ρ on q^* 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 $\rho_q \geq 0$, $\mu(I_B) \leq m/\epsilon < \infty$. □

Proof of Theorem 1: For any trace v of $\mathcal{B}$, there is a run since $\mathcal{B}$ is specifiable by $\mathcal{A}$. For any run r of $\mathcal{A}$ over v, if any automaton-state in R appears infinitely many times in r, r is accepting. Otherwise there is a time point t_0, the sub-sequence r on $I = \{t \in \mathcal{T} | t \geq t_0\}$, denoted q^*, has only bad and stable automaton-states. If there exist a set of invariants and a set of Liapunov functions, $\mu(I_B)$ is finite. Since time is infinite, all the automaton-states appearing infinitely many times in r belong to S; r is accepting too. Therefore, every trace is accepting for the automaton. □