Will The Robot Do The Right Thing?

Ying Zhang and Alan K. Mackworth*
Department of Computer Science
University of British Columbia
Vancouver, B.C.
Canada v6T 1Z4

Abstract

Robots are generally composed of multiple sensors, actuators and electromechanical parts. The overall behavior of a robot is emergent from coordination among its various parts and its interaction with its environment. Designing a 'correct' robot which does 'the right thing' in a given environment is an important and challenging problem. The question posed in the title is decomposed into two questions. First, what is the right thing? Second, how does one guarantee the robot will do it? We answer these questions in this paper by establishing a formal approach to the design and analysis of robotic systems and behaviors.

1 Motivation and Introduction

Building control systems for intelligent, reliable, robust and safe autonomous robots working in complex environments is an increasingly important challenge for research in electrical and mechanical engineering, and computer science.

Robots are generally composed of multiple sensors, actuators and electromechanical parts. Robots should be reactive as well as purposive systems, closely coupled with their environments; they must deal with inconsistent, incomplete and delayed information from various sources. Such systems are usually complex, hierarchical and physically distributed. Each component functions according to its own dynamics. The overall behavior of a system is emergent from coordination among its various parts and its interaction with its environment. We call the integration of a robot and its environment a *robotic system*, and the relation on the state of a robot and its environment over time the *robotic behavior*.

The current trend for developing intelligent robots is to combine AI techniques with traditional control theory [Schoppers, 1991]. However, most of this work is *ad hoc*; there is no well defined interface between the higher level (AI) and the lower level (control). The coordination between these levels is not fully understood, and the behavior of the whole system cannot be analyzed. One fundamental problem is the mismatch of the underlying computational models. AI is based on off-line computational models and control is based on on-line computational models. (The distinction between off-line and on-line models is analogous to the distinction between functions and processes.)

In this paper, we advocate a formal approach to modeling a robotic system. We have developed a formal model, Constraint Nets (CN), for general dynamic systems [Zhang and Mackworth, 1994a]. CN is an abstraction of general dynamic systems so that a system with discrete as well as continuous time, and asynchronous as well as synchronous event structures can be modeled in a unitary framework. Using aggregation operators, a system can be modeled hierarchically in CN; therefore, the dynamics of the environment as well as the dynamics of the robot can be modeled individually and then integrated. Based on abstract algebra and topology, CN supports multiple levels of abstraction, so that a system can be analyzed at different levels of detail. With a rigorous formalization, CN provides a programming semantics for the design of robot control systems.

We believe that the intelligence of an agent should be judged by the quality of the agent's interaction with the environment [Brooks, 1991]. The intelligence of an agent is measured by its ability to accomplish difficult tasks in complex, hazardous or uncertain environments. However, because there is, as yet, no rigorous definition for intelligent behaviors, we shall use the concept of *desired behaviors*.

In this paper, we advocate a formal approach to specifying desired behaviors and to verifying the relationship between a dynamic system and its behavior specification. Since robotic behaviors are the relationships between robots and their environments *over time*, the specification language should at least be able to represent temporal behaviors: states of a system over time. Various forms of temporal logics [Emerson, 1990] have been proposed in both the systems [Manna and Pnueli, 1992; Lamport, 1991; Ostroff, 1989; Alur and Henzinger, 1989] and AI [Allen, 1990; Shoham, 1988; McDermott, 1990; Rosenschein, 1985] communities. We have adopted an automaton-based specification language, called $\forall$-automata [Manna and Pnueli, 1987], which is capable of representing a large class of temporal properties

*Shell Canada Fellow, Canadian Institute for Advanced Research

In Proc. Artificial Intelligence '94, Banff, AB, May 1994, pp.255-262.

such as safety, liveness (recurrence, persistence, stability or controllability), goal achievement (reachability) and bounded response. Furthermore, a system modeled by a constraint net can be verified against its desired behavior specification by a general verification method.

The rest of the paper is organized as follows. Section 2 depicts the structure of robotic systems and the specification of robotic behaviors, illustrated by two running examples: a hand coordinator and a maze traveler. Section 3 gives the formal model for robotic systems, the Constraint Net model, and demonstrates constraint net modeling via the examples. Section 4 presents the formal specification language and its relationship with the Constraint Net model. Section 5 describes the formal verification method. Section 6 concludes the paper and points out some related research.

2 Robotic Systems and Behaviors

From a systemic point of view, a robotic system is a coupling of a robot to its environment, while the robot is an integration of a plant and its controller (Fig. 1). Basically, the roles of these three subsystems can be characterized as follows:

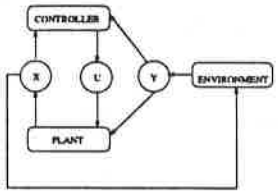

Figure 1: A robotic system

- *Plant*: a plant is a set of entities which must be controlled to achieve certain behaviors. For example, a robot arm with multiple joints, a car with throttle and steering, an airplane or a nuclear power plant can be considered as the *plant* of a robotic system.
- *Controller*: a controller is a set of sensors and actuators together with software/hardware computational systems which sense the states of the plant (X) and the environment (Y), and compute desired inputs (U) to actuate the plant. For example, an analog circuit, a program in a digital computer, various motors and sensors can be considered as parts of the *controller* of a robotic system.
- *Environment*: an environment is a set of entities beyond the control of the controller, with which the plant may interact. For example, obstacles to be avoided, objects to be reached, and rough terrain to be traversed can be considered as the *environment* of a robotic system.

We introduce two running examples to illustrate the general structure of robotic systems.

Example 2.1 The Hand Coordinator: Suppose a two-handed robot is required to fit caps on jars on an automated assembly line. The robot must pick up a jar and hold it with one hand and then fit a cap on the jar with its other hand. However, the hands work asynchronously at their own speed; for example, jars or caps may occasionally be unavailable, but we can assume that the acts of jar picking and cap fitting take some constant time. We will design a hand coordinator so that the right hand will cap only if the left hand is holding a jar; the left hand will put down the jar and pick up a new one only if the right hand has done the capping.

The robotic system, shown in Fig. 2, consists of the hand coordinator, and the left and right hands.

Figure 2: The hand coordinator system

The whole system should work as follows. Whenever there are more jars available, the left hand will request permission ($R1$) to pick up a jar, and the coordinator will grant the request ($C1$) if the previous jar has been capped. On the other hand, whenever there are more caps available, the right hand will request permission ($R2$) to cap a jar, and the coordinator will grant the request ($C2$) if the left hand is holding a jar in place.

Example 2.2 The Maze Traveler: Consider a maze composed of separated T-shaped obstacles of bounded size placed in one of four orientations on an unbounded plane. A simple robot (Fig. 3 (a)) is required to traverse the maze from west to east (Fig. 3 (b)).

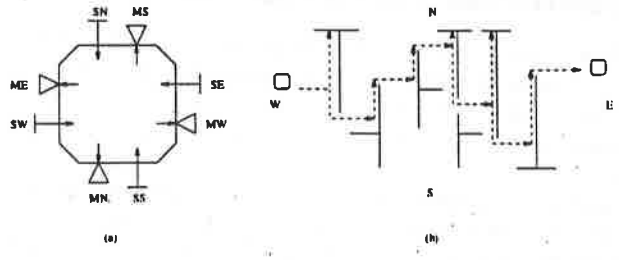

Figure 3: (a) A simple robot (b) A simple maze

In this example, the plant is the body of the robot which can move in one of four directions; the environment is the maze; and the controller connects sensing signals to motor commands (Fig. 4). For example, when the north sensor SN is on, the robot is touching a wall directly to its north; when the east motor ME is on, the robot moves east, if it is not blocked.

While the model of a robotic system states how the system is composed and how the system works formally, the specification of a robotic behavior represents what should the robot do overall. For the hand coordinator, one desired behavior is that the acts of jar picking and cap fitting should interleave. For the maze traveler, one desired behavior is that the robot should move to the east persistently. Given a formal model of a designed system and a formal specification of a desired behavior,

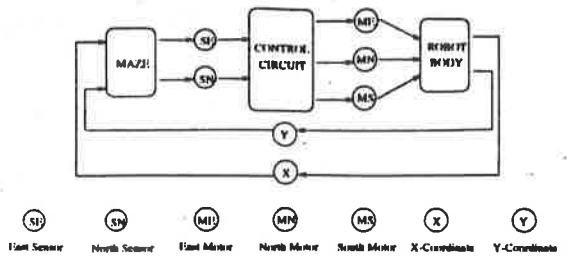

Figure 4: The maze traveler robotic system

one should be able to prove that the model does satisfy the specification, that is, that the robot will do the right thing.

Therefore, 'do the right thing' here does not necessarily mean that the robot has rationality built in [Russell and Wefald, 1991]; it simply means that the robot does what it is designed to do. Furthermore, the specification language we propose focuses only on temporal aspects; probability and stochastic analysis will be incorporated into this modeling framework in the future.

3 Model for Robotic Systems

In this section, we introduce the Constraint Net model and characterize its composite structure and modularity. The formal semantics of the model, based on the fixpoint theory of continuous algebras, has been presented in [Zhang and Mackworth, 1994a].

3.1 Dynamic systems

Since a robotic system is a dynamic system in general, we start with some basic concepts of dynamic systems. Let a *time structure* be a totally ordered set, which could be intervals of reals or ordered events, with a metric topology. Let a *domain* be a set of values, which could be numbers, symbols or strings. Both time and domains can be either continuous or discrete. In this paper, we mainly discuss discrete systems. However, all definitions here works for the general case.

- *Trace*: A trace $v : \mathcal{T} \rightarrow A$ is a function from a time structure $\mathcal{T}$ to a domain A. The set of traces is denoted by *trace space* $A^{\mathcal{T}}$. Traces can be transformed from one to another via *transductions*.
- *Transduction*: A transduction is a function from input traces to output traces which satisfies the causal relationship between its inputs and outputs, viz. the output values at any time are determined by the input values up to that time. Transductions can be considered as transformational processes. For example, a temporal integration is a typical transduction in continuous time; a state automaton defines a transduction in discrete time. Clearly, transductions are closed under functional composition. There are two basic types of transductions: *transliterations* and *delays*.
- *Transliteration*: A transliteration $f_{\mathcal{T}}$ is a pointwise extension of function f on time structure $\mathcal{T}$, that is, the output value at any time is the function of the input value at that time only. Let v be the input trace then we have $f_{\mathcal{T}}(v) = \lambda t.f(v(t))$. Intuitively, a transliteration is a transformational process without memory or internal state, for example, a combinational circuit. We use f to denote the transliteration $f_{\mathcal{T}}$ if no ambiguity arises.
- *Unit delay*: A unit delay $\delta(v_0)$ is a transduction defined mainly for discrete time structures, such that the output value at initial time 0 is v_0 and the rest of the output values are the input values at the previous time:

$$\delta(v_0)(v) = \lambda t. \begin{cases} v_0 & \text{if } t = 0 \\ v(pre(t)) & \text{otherwise} \end{cases}$$

where $pre(t)$ indicates the previous time point adjacent to t in the total order. The definition of $pre(t)$ can be generalized to arbitrary time structures [Zhang and Mackworth, 1994a]. A unit delay acts as a unit memory in discrete dynamic systems.

Any discrete dynamic system can be modeled using only transliterations and unit delays. A hybrid dynamic system, composed of both discrete and continuous components, can be modeled by event-driven transductions and transport delays, in addition to these two types of transductions [Zhang and Mackworth, 1994a].

3.2 Constraint nets

A dynamic system can be modeled by a constraint net. Influenced by some dataflow-like models [Ashcroft, 1986; Lavignon and Shoham, 1990; Benveniste and LeGuernic, 1990; Caspi *et al.*, 1987], the Constraint Net model is developed on an abstract dynamics structure, with abstract time and domains.

Intuitively, a constraint net consists of a finite set of locations, a finite set of transductions and a finite set of connections. A location can be regarded as a wire, a channel, a variable, or a memory location, whose value may change over time. Each transduction is a causal mapping from input traces to output traces. Connections relate locations with ports of transductions.

Syntactically, a *constraint net* is a triple $CN = \langle Lc, Td, Cn \rangle$, where Lc is a finite set of *locations*, each of which is associated with a domain; Td is a finite set of labels of *transductions*, each of which is associated with a set of *input ports* and an *output port*; Cn is a set of *connections* between locations and ports of transductions, with the following restrictions: (1) there is at most one output port connecting to each location, (2) each port of a transduction connects to a unique location and (3) no location is isolated.

A location l is an *output location* of a transduction F iff there is a connection between the output port of F and l; l is an *input location* of F iff there is a connection between an input port of F and l. A location is an *output* of the constraint net if it is an output location of a transduction otherwise it is an *input*. The set of input locations of a constraint net CN is denoted by $I(CN)$, the set of output locations is denoted by $O(CN)$. A constraint net is *open* if there is an input location otherwise it is *closed*.

A constraint net is represented by a bipartite graph where locations are depicted by circles, transductions are depicted by boxes and connections are depicted by arcs. We have seen examples of constraint nets in Fig. 1, Fig. 2 and Fig. 4.

Semantically, a constraint net is a set of equations, $\vec{o} = \vec{F}(\vec{i}, \vec{o})$, where each left-hand side is an individual output location and each right-hand side is an expression composed of transductions and locations. The semantics is defined as a solution of the set of equations [Zhang and Mackworth, 1994a]. If CN is a constraint net with time structure $\mathcal{T}$ and domain A_l for $l \in Lc$, the semantics is a transduction from input traces to output traces: $[\![CN]\!] : \times_{I(CN)} A_i^{\mathcal{T}} \rightarrow \times_{O(CN)} A_o^{\mathcal{T}}$ where $\times_L A_l^{\mathcal{T}}$ denotes the product of a family of trace spaces.

A constraint net can be hierarchically organized. A *module* is a triple $\langle CN, I, O \rangle$, denoted by $CN(I, O)$, where CN is a constraint net, $I \subseteq I(CN)$ and $O \subseteq O(CN)$ are subsets of the input and output locations of CN; $I \cup O$ defines the *interface* of the module. Locations $I(CN) - I$ and $O(CN) - O$ are called ***hidden input*** and ***hidden output locations*** respectively.

Graphically, a module is depicted by a box with rounded corners. If $[\![CN]\!]$ is the semantics of CN, the semantics of $CN(I, O)$ is the semantics of CN projected onto the interface, i.e. $[\![CN(I,O)]\!] = \Pi_{I \cup O}[\![CN]\!] = \{(\vec{i}, \vec{o}) | o = [\![CN]\!]_o(\vec{u}, \vec{i}), \vec{u} \in \times_U A_l^{\mathcal{T}}\}$ where $U = I(CN) - I$ is the set of hidden input locations. If $U \neq \emptyset$, i.e. $I \subset I(CN)$, $[\![CN(I,O)]\!]$ is a relation between input and output traces in general, rather than a function. Thus, while more powerful, and simpler, than most inherently nondeterministic models, nondeterminism can be modeled with hidden inputs, and probabilistic and stochastic analysis can be incorporated (if we provide random distributions on hidden inputs).

Generally speaking, modularity provides a kind of abstraction: hidden inputs capture nondeterminism and hidden outputs encapsulate internal state.

Furthermore, a complex module can be constructed from simple ones with aggregation operators [Zhang and Mackworth, 1994a], and the semantics has compositional properties.

Given a system modeled as a module $CN(I, O)$, the behavior of the system can be formally defined as a set of observable input/output traces $\{\vec{v} \in \times_{I \cup O} A_l^{\mathcal{T}} | \vec{v} \in [\![CN(I,O)]\!]\}$. We also use $[\![CN(I,O)]\!]$ to denote the behavior of the system.

So far, we have briefly presented the Constraint Net model (CN). CN is as powerful as the existent computational models, in either discrete sequential computations (Turing Machines) or continuous analog computations (smooth non-hypertranscendental functions [Shannon, 1941]) [Zhang, 1994]. CN is able to represent a hybrid system, consisting of a non-trivial mixture of discrete and continuous components. For the two running examples given in the previous section, we focus here only on their discrete models. Examples of hybrid system modeling can be found in [Zhang and Mackworth, 1993b].

In general, a robotic system is modeled as an integration of a plant module, a control module and an environment module (Fig. 1), and each of which may be further decomposed into a hierarchy of modules. The overall behavior of the system is not determined by any one of the modules, but emerges from the coupling of the interaction among all the components. Formally, the behavior of the system is the solution of the following equations:

$$\begin{aligned} X &= PLANT(U, Y), \\ U &= CONTROLLER(X, Y), \\ Y &= ENVIRONMENT(X). \end{aligned}$$

As we can see here, a robot, composed of a plant and a controller, is an open system in general, and a robotic system, with a robot coupled to its environment, is a closed system.

Now we illustrate the constraint net modeling using the two examples: the hand coordinator and the maze traveler.

Example 3.1 The Hand Coordinator: The hand coordinator can be designed using negated Muller C-elements [Sutherland, 1989], since the desired behavior of the hand coordinator is similar to a buffer synchronizer.

Consider the request and grant signals as events, transitions from 0 to 1 or 1 to 0. The Muller C-element acts as the 'and' for events: if both of its inputs have the same value, the output and its next state are copies of that value, otherwise the output and its next state are unchanged.

A Muller C-element can be modeled by a module $C(\{i_1, i_2\}, o)$ composed of a transliteration c and a unit delay:

$$o = c(i_1, i_2, q), q = \delta(0)(o)$$

where

$$c(i_1, i_2, q) = \begin{cases} i_1 & \text{if } i_1 = i_2 \\ q & \text{otherwise.} \end{cases}$$

We use the standard 'and' logic symbol with a 'C' inside it to represent Muller C-elements and 'bubbles' on input or output ports to represent inversions.

Assume that jar picking and cap fitting each take constant time. Two negated Muller C-elements and two delay elements are used to synchronize events in the controller (Fig. 5).

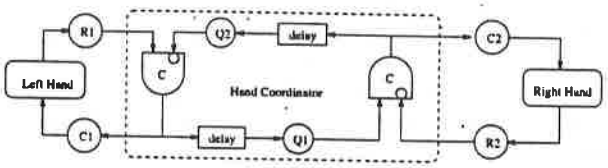

Figure 5: The hand coordinator

Example 3.2 The Maze Traveler: The controller for the maze traveler can be built using 'and' and 'not' gates with a flip-flop memory unit (Fig. 6).

The flip-flop $FF(\{i_1, i_2\}, o)$ is composed of a transliteration ff and a unit delay:

$$o = ff(i_1, i_2, q), q = \delta(0)(o)$$

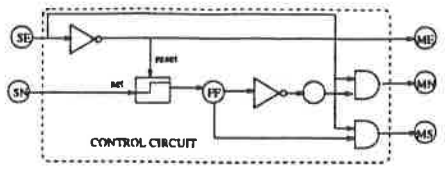

Figure 6: The controller for the maze traveler

where

$$ff(i_1, i_2, q) = \begin{cases} 1 & \text{if } i_1 = 1 \text{ (set)} \\ 0 & \text{else if } i_2 = 1 \text{ (reset)} \\ q & \text{otherwise.} \end{cases}$$

The control outputs to the robot body are: $ME = \neg SE, MN = \neg FF \wedge SE$ and $MS = FF \wedge SE$.

The robot body $BODY(\{ME, MN, MS\}, \{X, Y\})$ is composed of a transliteration f_{xy} and a unit delay:

$$\begin{aligned} \langle NX, NY \rangle &= f_{xy}(ME, MN, MS, X, Y), \\ \langle X, Y \rangle &= \delta(x_0, y_0)(NX, NY) \end{aligned}$$

where

$$f_{xy}(ME, MN, MS, X, Y) = \langle X + ME, Y + MN - MS \rangle,$$

i.e. the next $\langle X, Y \rangle$ position is displaced by a grid point depending on the motion commands.

Even though sensor and command domains are Boolean $\{0, 1\}$, notice that this robotic system is not finite, since the domain of both X and Y is the integers.

Will these robots do the right thing? We have given informal descriptions of what these two robotic systems are supposed to do in the previous section. Enthusiastic readers will check by hand that they seem to do the 'right thing'. However, even though a constraint net model gives a precise definition of what the behavior of the system is, it is infeasible to generate the behavior and check all the traces of the behavior. In the next two sections, we will present a behavior specification language for representing the desired behaviors of a system and a verification method for ensuring that the behavior of the system does satisfy its specification.

4 Specification for Robotic Behaviors

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 this design. For example, the acts of jar picking and cap fitting must interleave, the maze traveler should persistently move east. In this section, we present an automaton-based language for formally specifying robotic behaviors, and establish the relationship between a constraint net and a 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 have adopted $\forall$-automata [Manna and Pnueli, 1987] for our purpose.

$\forall$-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 $\forall$-automata have the same expressive power as Buchi automata [Thomas, 1990] and the extended temporal logic (ETL) [Wolper, 1983], which are strictly more powerful than the linear propositional temporal logic [Thomas, 1990; Wolper, 1983]. More importantly, there is a formal verification method. We have been able to generalize $\forall$-automata for accepting general traces [Zhang and Mackworth, 1994b]. In this paper, we focus only on discrete systems and infinite sequences.

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 *$\forall$-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.

For simplicity, let time be the set of natural numbers $\mathcal{N}$ and $v : \mathcal{N} \rightarrow A$ be a sequence. A *run* of $\mathcal{A}$ over v is a mapping $r : \mathcal{N} \rightarrow Q$ such that (1) $v(0) \models e(r(0))$, and (2) for all $n > 0$, $v(n) \models c(r(n-1), r(n))$.

A $\forall$-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.

These two requirements guarantee that any sequence has a run over it, and that any partial run can always be extended to an infinite sequence. 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 [Manna and Pnueli, 1987].

If r is a run, let $Inf(r)$ be the set of automaton-states appearing infinitely many times in r, that is $Inf(r) =$

$\{q|\forall n \exists m > n, r(m) = q\}$. 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 sequence v, written $v \models \mathcal{A}$, iff *all* possible runs of $\mathcal{A}$ over v are accepting. A $\forall$-automaton $\mathcal{A}$ *accepts* a discrete time system $CN(I,O)$, written $CN(I,O) \models \mathcal{A}$, iff for all $v \in [\![CN(I,O)]\!]$, $v \models \mathcal{A}$.

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. 7. Figure 7(a) accepts a sequence which satisfies $\neg G$ only finitely many times, Figure 7(b) accepts a sequence which never satisfies B, and Figure 7(c) accepts a sequence which will satisfy S in the finite future whenever it satisfies R.

Now we can formally specify the desired behaviors for the hand coordinator and the maze traveler.

Example 4.1 The Hand Coordinator: Let the acts of jar picking and cap fitting be controlled by the events at $C1$ and $C2$ respectively. Let $E(X)$ be an assertion denoting that there is an event at X. If $E(C1)$, the robot picks up a jar, and if $E(C2)$, the robot fits the cap.

Figure 7: $\forall$-automata: (a) goal achievement or reachability (b) safety (c) bounded response

One desired behavior for the hand coordinator is that $E(C1)$ and $E(C2)$ must interleave and $E(C1)$ always precedes $E(C2)$. This behavior can be represented by a $\forall$-automaton in Fig. 8 (a).

Example 4.2 The Maze Traveler: Let ME be an assertion denoting that the east motor is on, or $ME = 1$. One desired behavior for the maze traveler is the *liveness* property represented by a $\forall$-automaton in Fig. 8 (b), meaning that the robot will persistently move east.

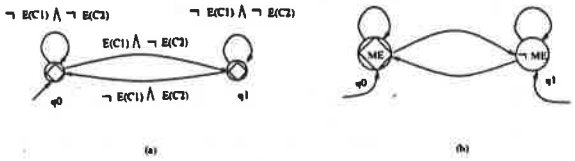

Figure 8: The specification of (a) the hand coordinator (b) the maze traveler

5 A Formal Verification Method

In this section, we present a verification method modified from [Manna and Pnueli, 1987] with concurrent programs replaced by discrete constraint nets.

Any discrete constraint net CN is composed of two types of transductions, transliterations and unit delays. Therefore CN can be represented by two sets of *domain equations*, each of the form $l_0' = l$, if l_0 is an output location of a unit delay from the input location l, or $l_0 = f(l_1, \ldots, l_n)$, if l_0 is an output location of a transliteration f from the input location tuple $\langle l_1, \ldots, l_n \rangle$.

The domain equations for the hand coordinator and the maze traveler are as follows.

Example 5.1 The Hand Coordinator: For simplicity, assume that the delays in the hand coordinator in Fig. 5 are unit delays. The set of equations for the hand coordinator is:

$$C1 = c(R1, \neg Q2, Q1), \quad Q1' = C1,$$

$$C2 = c(Q1, \neg R2, Q2), \quad Q2' = C2.$$

Example 5.2 The Maze Traveler: The set of equations for the maze traveler is:

$$FF = ff(SN, \neg SE, Q), Q' = FF,$$

$$ME = \neg SE, MN = \neg FF \wedge SE, MS = FF \wedge SE,$$

$$\langle X', Y' \rangle = f_{xy}(ME, MN, MS, X, Y)$$

A state s of CN is a mapping from locations to domains, i.e. $s \in \times_{Lc} A_l$ where $Lc = I(CN) \cup O(CN)$. A pair of states $\langle s, s' \rangle$ is said to be *consistent* with CN, denoted $CN(s, s')$, iff for every equation of the form $l_0 = f(l_1, \ldots, l_n)$, $s(l_0) = f(s(l_1), \ldots, s(l_n))$ and $s'(l_0) = f(s'(l_1), \ldots, s'(l_n))$, and for every equation of the form $l'_0 = l$, $s'(l_0) = s(l)$.

Let φ and ψ be assertions on states of a constraint net. We write $\{\varphi\}CN\{\psi\}$ to denote that the consecutive condition: $\varphi(s) \bigwedge CN(s, s') \rightarrow \psi(s')$ is valid.

Let Θ be an assertion indicating the initial state of CN, and $\mathcal{A} \equiv \langle Q, R, S, e, c \rangle$ 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 $\mathcal{A}$ and W is a *well-founded set*, i.e. any decreasing sequence of W is finite, a set of partial functions $\{\rho_q\}_{q \in Q} : \times_{Lc} A_l \rightarrow W$ is called a set of *ranking* 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*: $\forall q \in Q, q' \in B.$

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

We conclude that if the following requirements are satisfied the validity of a $\forall$-automaton $\mathcal{A}$ over a constraint net CN is proved:

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

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

As in [Manna and Pnueli, 1987], the verification rules (I) and (R) are sound and complete for a complete $\forall$-automaton $\mathcal{A}$ and a discrete constraint net CN, i.e. $\mathcal{A}$ accepts CN iff there exist a set of invariants and ranking functions.

Now we can prove that the hand coordinator in Fig. 5 does satisfy its specification in Fig. 8(a), and the maze traveler controlled by the control circuit in Fig. 6 does satisfy its specification in Fig. 8(b).

Example 5.3 The Hand Coordinator: The $\forall$-automaton in Fig. 8(a) is not complete. To make it complete, we add another automaton-state $q_E \in B$ with $e(q_E) = \mathit{false}$, $c(q_0, q_E) = E(C2)$, $c(q_1, q_E) = E(C1)$, $c(q_E, q_i) = \mathit{false}$ for $i = 0, 1$, and $c(q_E, q_E) = \mathit{true}$.

Let $E(C1)$ be $C1 \neq Q1$ and $E(C2)$ be $C2 \neq Q2$. Let the initial condition Θ be $C1 = C2$. It is easy to check that $C1 = C2$, $C1 \neq C2$ and false are invariants for q_0, q_1 and q_E respectively. Therefore the verification rule (I) is satisfied.

Since $q_0, q_1 \in R$ and the invariant of the only bad state q_E is false, the verification rule (R) is trivially satisfied.

As a result, we have proved that the acts of jar picking and cap fitting, controlled by the hand coordinator, do interleave.

Example 5.4 The Maze Traveler: Let q_0, q_1 in Fig. 8(b) be associated with ME and $\neg ME$ respectively. The verification rule (I) is trivially satisfied.

Suppose the maximum length of an obstacle is L. Associate with each automaton-state the same function $\rho : \{0,1\} \times \{0,1\} \times \{0,1\} \times \mathcal{Z} \rightarrow \{0,1\} \times \mathcal{D}$ where $\mathcal{Z}$ is the set of integers and $\mathcal{D}$ is the interval $[0, L+1]$ of natural numbers. Let ρ be defined as:

$$\rho(ME, MN, MS, Y) = \begin{cases} \langle 1, 1 + L \rangle & \text{if } ME = 1 \\ \langle 1, DN - Y \rangle & \text{if } MN = 1 \\ \langle 0, Y - DS \rangle & \text{if } MS = 1 \end{cases}$$

where DN (DS) is the Y-coordinate of the north (south) end of the current maze block. Obviously $DN - Y$ and $Y - DS \leq L$. The order on $\{0,1\} \times \mathcal{D}$ is defined as: $\langle 0, - \rangle < \langle 1, - \rangle$ and $\langle X, Y_1 \rangle \leq \langle X, Y_2 \rangle$ iff $Y_1 \leq Y_2$. $\{0,1\} \times \mathcal{D}$ is a well-founded set since L is finite. With this function and the well-founded set, any transition that ends up at $q_1 \in B$ would lead to a decrease. Therefore, ρ is a ranking function.

As a result, we have proved that the maze traveler does move east infinitely often, escaping any finite maze of that type.

We should notice that the verification method is formal but not necessarily amenable to automation if the domains of the constraint net are not finite. In fact, there is no verification algorithm, in general, since the discrete constraint net is powerful enough to simulate a Turing machine and the specification language is rich enough to state the halting problem. However, an automatic or semi-automatic theorem prover can be used for proving the validity of the formulas derived from this method.

6 Conclusion and Related Work

Will the robot do the right thing? One can guarantee the answer 'yes' by modeling the robotic system, including the environment when necessary, at an appropriate level of abstraction and proving that the model satisfies the desired behavior specification. In this paper, we have illustrated a formal approach to the modeling, specification and verification of discrete robotic systems.

We have done some further related work on this subject, (1) designing a verification algorithm for finite systems [Zhang and Mackworth, 1994c], (2) extending $\forall$-automata to timed $\forall$-automata to deal with real-time responses [Zhang and Mackworth, 1994c], and (3) extending ranking functions to Liapunov functions to deal with continuous time and domains [Zhang and Mackworth, 1994b]. We have also worked on control synthesis based on constraint satisfaction using Liapunov functions [Zhang and Mackworth, 1993a]. In fact, the problems of synthesis and verification are coupled in the design and analysis of robotic systems.

Acknowledgement

We wish to thank the anonymous referees for their constructive comments on the paper. This research was supported by the Natural Sciences and Engineering Research Council and the Institute for Robotics and Intelligent Systems.

References

[Allen, 1990] J. F. Allen. Towards a general theory of action and time. In James Allen, James Hendler, and Austin Tate, editors, *Readings in Planning*, pages 464 – 479. Morgan Kaufmann Publishers Inc., 1990.

[Alur and Henzinger, 1989] R. Alur and T. A. Henzinger. A really temporal logic. In *30th Annual Symposium on Foundations of Computer Science*, pages 164 – 169, 1989.

[Ashcroft, 1986] E. A. Ashcroft. Dataflow and eduction: Data-driven and demand-driven distributed computation. In J. W. deBakker, W.P. deRoever, and G. Rozenberg, editors, *Current Trends in Concurrency*, number 224 in Lecture Notes on Computer Science. Springer-Verlag, 1986.

[Benveniste and LeGuernic, 1990] A. Benveniste and P. LeGuernic. Hybrid dynamical systems theory and the SIGNAL language. *IEEE Transactions on Automatic Control*, 35(5), May 1990.

[Brooks, 1991] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47(1 – 3), January 1991.

[Caspi *et al.*, 1987] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice. LUSTRE: A declarative language for programming synchronous systems. In *ACM Proceeding of Principles of Programming Languages*, 1987.

[Emerson, 1990] E. Emerson. Temporal and modal logic. In Jan Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics. Elsevier, The MIT Press, 1990.

[Lamport, 1991] L. Lamport. The temporal logic of actions. Technical Report 79, Digital Systems Research Center, Palo Alto, California, December 1991.

[Lavignon and Shoham, 1990] J. Lavignon and Y. Shoham. Temporal automata. Technical Report STAN-CS-90-1325, Robotics Laboratory, Computer Science Department, Stanford University, Stanford, CA 94305, 1990.

[Manna and Pnueli, 1987] 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.

[Manna and Pnueli, 1992] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer-Verlag, 1992.

[McDermott, 1990] D. McDermott. A temporal logic for reasoning about processes and plans. In James Allen, James Hendler, and Austin Tate, editors, *Readings in Planning*, pages 436 – 463. Morgan Kaufmann Publishers Inc., 1990.

[Ostroff, 1989] J. S. Ostroff. *Temporal Logic For Real Time Systems.* John Wiley & Sons Inc., 1989.

[Rosenschein, 1985] S. J. Rosenschein. Formal theories of knowledge in AI and robotics. *New Generation Computing*, 1985.

[Russell and Wefald, 1991] S. Russell and E. Wefald. *Do the Right Thing: studies in limited rationality.* MIT Press, 1991.

[Schoppers, 1991] M. Schoppers, editor. *Communications of ACM.* ACM, August 1991. Special Section on Real-Time Knowledge-Based Control Systems.

[Shannon, 1941] C. E. Shannon. Mathematical theory of the differential analyzer. *Journal of Mathematics and Physics*, 20:337 – 354, 1941.

[Shoham, 1988] Y. Shoham. *Reasoning about Change* MIT Press, 1988.

[Sutherland, 1989] I. E. Sutherland. Micropipeline *Communication of ACM*, 32(6), June 1989.

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

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

[Zhang and Mackworth, 1993a] 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.

[Zhang and Mackworth, 1993b] 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.

[Zhang and Mackworth, 1994a] Y. Zhang and A. K. Mackworth. Constraint Nets: A semantic model for hybrid dynamic systems, 1994. Working Paper. Department of Computer Science, UBC.

[Zhang and Mackworth, 1994b] Y. Zhang and A. K. Mackworth. Specification and verification of constraint-based dynamic systems. In *Second Workshop on Principles and Practice of Constraint Programming*, May 1994.

[Zhang and Mackworth, 1994c] Y. Zhang and A. K. Mackworth. Specification and verification of discrete dynamic systems using timed ∀-automata, 1994. Working Paper. Department of Computer Science. UBC.

[Zhang, 1994] Y. Zhang. A foundation for the design and analysis of robotic systems and behaviors, 1994. PhD thesis, forthcoming. Department of Computer Science, UBC.