Will the Robot Do the Right Thing?

by
Ying Zhang and Alan K. Mackworth*

Technical Report 92-31
November 1992

Department of Computer Science
The University of British Columbia
Vancouver, B. C. V6T 1Z2
Canada

email: zhang@cs.ubc.ca, mack@cs.ubc.ca

*Shell Canada Fellow, Canadian Institute for Advanced Research

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 1Z2
E-mail: zhang@cs.ubc.ca, mack@cs.ubc.ca
phone: (604)822-3731

Abstract

Constraint Nets have been developed as an algebraic on-line computational model of robotic systems. A robotic system consists of a robot and its environment. A robot consists of a plant and a controller. A constraint net is used to model the dynamics of each component and the complete system. The overall behavior of the system emerges from the coupling of each of its components. 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 by establishing a formal approach to the specification and verification of robotic behaviors. In particular, we develop a real-time temporal logic for the specification of behaviors and a new verification method, based on timed $\forall$-automata, for showing that the constraint net model of a robotic system satisfies the specification of a desired global behavior of the system. Since the constraint net model of the controller can also serve as the on-line controller of the real plant, this is a practical way of building well-behaved robots. Running examples of a coordinator for a two-handed robot performing an assembly task and a reactive maze traveler illustrate the approach.

*Shell Canada Fellow, Canadian Institute for Advanced Research

1 Motivation and Introduction

Robots are generally composed of multiple sensors, actuators and electromechanical components. Robots are 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 robot cannot be determined by any one of its components. Rather, it is emergent from the coupling of the dynamics of its various parts and its interaction with the environment. Therefore, in order to predict the robotic behaviors, we have to model the robot, the environment and their interactions. The current trend for developing intelligent robots is to combine AI techniques with the traditional control theory [15]. However, most of this work is *ad hoc*, and 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. One major problem is a fundamental mismatch of computational models. AI is based on off-line computational models and control is based on on-line computational models. (Off-line models take a computational problem as a function from inputs to outputs; on-line models take a computational problem as a process from input traces to output traces. Differential/difference equations and Mealy/Moore Machines [11, 12] are typical on-line models.)

We have advocated instead a formal approach to modeling a *robotic system* consisting of the plant, the controller and the environment. Influenced by the Operator Net model [3] and the Temporal Automaton Model [7], we have developed an algebraic model, called Constraint Nets (CN) [20], of general dynamic systems. The motivation for this unified modeling framework is to provide concise and formal semantics for integrated systems with multiple components and mixed types of dynamic structures. With CN, environments and plants as well as control structures can be represented in a single on-line framework under multiple levels of abstraction [17].

We believe that the intelligence of an agent should be judged by the quality of the agent's interaction with the environment [4]. 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 of intelligent behavior, we shall use the concept of *desired behavior*. In this paper we develop a formal specification language for desired behavior and explore a formal method for the verification of the system under design.

While modeling focuses on the underlying structure of a system, and the organization and coordination of the components or subsystems, specification imposes constraints on the system's global behavior. Since robotic behaviors are the relationships between robots and their environments *over time*, we choose temporal logics as the specification language. Various forms of temporal logics [5] have been proposed in both the systems [9, 6, 13, 2] and AI [1, 16, 10, 14] communities. We will develop a real-time linear temporal logic which is capable of representing a large class of temporal properties such as safety, liveness (recurrence or persistence), goal achievement (equilibrium) and real-time response. Furthermore, a system modeled by a constraint net can be verified against its desired global behavior specification by a general verification method. Manna and Pnueli [8] have developed a general method for verifying a concurrent program against a $\forall$-automaton specification. $\forall$-automata serve as a more perspicuous representation than linear temporal logics. We extend $\forall$-automata to timed $\forall$-automata by introducing timed states, so that any formula in the real-time temporal logic can be represented by a timed $\forall$-automaton. The general method is modified to verify a constraint net against a timed $\forall$-automaton specification.

The rest of the paper is organized as follows. Section 2 depicts the structure of robotic systems, illustrated by two running examples: a hand coordinator and a maze traveler. In addition, we introduce some general concepts of dynamic systems. Section 3 gives the structure of Constraint Nets and demonstrates constraint net modeling via the examples.

Section 4 presents the temporal logic specification language and its relationship with the Constraint Net model. Section 5 describes timed $\forall$-automata and the verification method. Section 6 concludes the paper and points out some directions for further research.

2 Structure and Dynamics of Robotic Systems

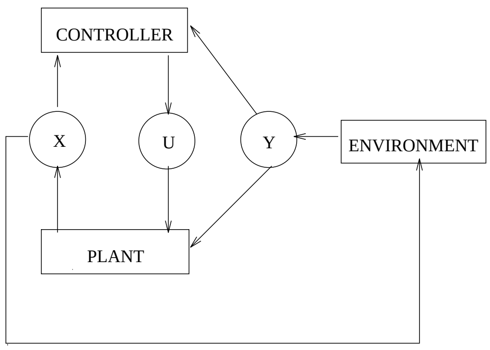

Figure 1: A robotic system

A robotic system is a constrained dynamic system consisting of the plant, the controller and the environment (Fig. 1). Basically, the roles of these three subsystems can be characterized as follows:

- *Plant*: the plant is a set of entities which must be controlled to achieve certain behaviors. For example, one or more robot arms, one or more vehicles, an airplane or a nuclear power plant can be considered as the *plant* of a robotic system.

- *Controller*: the controller is a set of computational systems which sense the states of the plant and the environment, and send commands or data to the plant being controlled. For example, an analog circuit, a digital computer, a combination of analog and digital devices as well as various sensing systems can be considered as parts of the *controller* of a robotic system.

- *Environment*: the 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, other agents, rough terrain, or the stiffness of a contacted surface 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 are working 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 are atomic. 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 pick up a jar only if the right hand has done the capping. The robotic system, shown in Fig. 2, consists of the hand coordinator (part of the controller), the left and right hands (part of the controller and the plant), and the assembly line with jars and caps (the environment). The whole system should work as

Figure 2: The hand coordinator system

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*:* Suppose a maze is composed of separated $\top$-shaped obstacles of bounded size placed in one of four directions on an unbounded plane.

A simple robot (Fig. 3 (a)) is required to traverse the maze from west to east (Fig. 3 (b)).

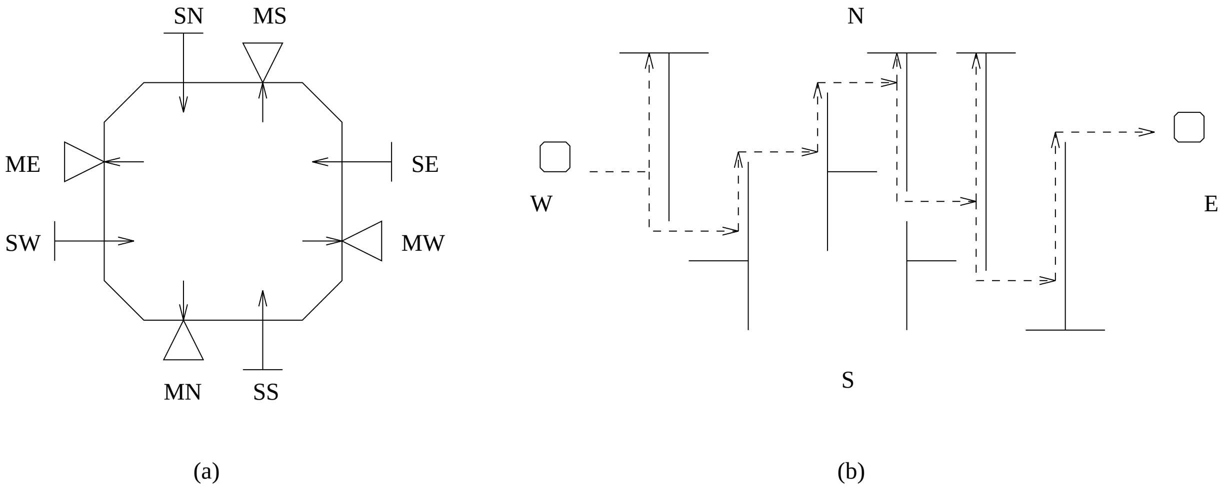

Figure 3: (a) A simple robot (b) Traveling through a maze

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.

To conclude this section, we introduce some general concepts of dynamic systems which will be used later.

- *Variable trace*: A variable trace is a function of time to a domain of values. For simplicity, in this paper, we shall consider time as the set of natural numbers with arithmetic ordering. A general time structure has been defined in [20].
- *Transduction*: A transduction is a mapping from input traces to output traces, which satisfies the causal relationship between its inputs and outputs, viz. the

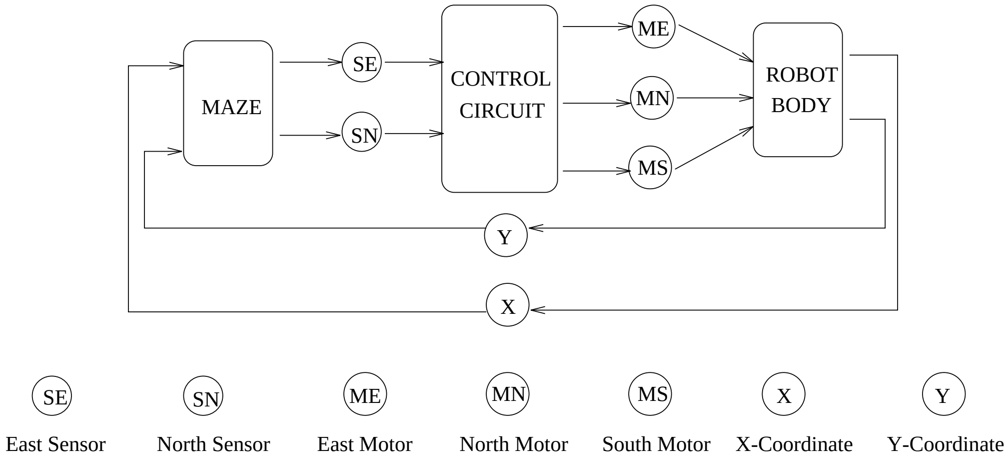

Figure 4: The maze traveler robotic system

output values at any time are determined by the input values prior to or at that time. Transductions can be considered as transformational processes. For example, a temporal integration is a transduction. We will see that any deterministic automaton defines a transduction. Clearly, transductions are closed under composition. *Transliterations* and *unit delays* are two elementary types of transductions for dynamic systems.

- *Transliteration*: A transliteration $f_{\mathcal{T}}$ is a pointwise extension of function f, that is, the output value at any time is the function of the input value at that time. Let v be the input variable trace then we have $f_{\mathcal{T}}(v) = \lambda t.f(v(t))$.

- *Unit delay*: A unit delay $\delta(init)$ is a transduction such that the initial output value is $init$ and the rest of the output values are the input values at the previous time.

- *Dynamics of Robotic Systems*: A robotic system can be modeled as a set of transductions. Formally, let X be the state traces of the plant, U be the control traces and Y be the state traces of the environment, the plant is a transduction $P : U \times Y \to X$; the controller is a transduction $C : X \times Y \to U$; and the environment is a transduction $E : X \to Y$. The overall behavior of the system is not determined by any one of the transductions, but emerges from the coupling of the interactions among all the transductions. Formally, the trajectory of the system is the solution of the following equations:

$$x = P(u, y), \quad u = C(x, y), \quad y = E(x)$$

3 Modeling with Constraint Nets

In this section, we introduce the Constraint Net (CN) 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 [20].

3.1 Constraint nets

A *constraint net* is a triple $CN \equiv \langle Lc, Td, Cn \rangle$, where Lc is a set of *locations*, each of which is associated with a *sort*; Td is a set of *transductions*, each of which is associated with a tuple of *input ports* and an *output port*, of given sorts; Cn is a set of directed *connections* between locations and ports of transductions of the same sort. Topologically, a constraint net is a bipartite directed graph where locations are represented by circles, transductions are represented by boxes and connections are represented by arcs, each from a port of a transduction to a location or vice versa, with the restriction that (1) there is at most one connection pointing to each location, (2) each port of a transduction connects to a unique location and (3) no location is isolated. A location is an *input* iff there is no connection pointing to it otherwise it is an *output*. For a constraint net CN, the set of input locations is denoted by $I(CN)$, the set of output locations is denoted by $O(CN)$. A constraint net is *closed* iff there are no input locations otherwise it is *open*.

Semantically, a constraint net is a set of equations, 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 the least fixpoint of the set of equations.

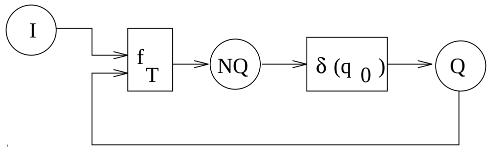

Figure 5: A state transducer

A *state transducer* is a quadruple $\langle I, Q, q_0, f\rangle$ where I is the set of inputs, Q is the set of states, $q_0 \in Q$ is the initial state and $f : I \times Q \to Q$ is the *state transition function.* A state transducer can be represented by an open constraint net with a transliteration f_T and a unit delay $\delta(q_0)$ (Fig. 5). A state transducer defines a transduction from input traces to state traces.

Example 3.1 A negated Muller-C element is a basic element in asynchronous circuits [18]. A negated Muller-C element is a state transducer defined as $\langle \mathcal{B} \times \mathcal{B}, \mathcal{B}, 0, nc\rangle$ where $\mathcal{B} = \{0, 1\}$,

$$nc(\langle i_1, i_2\rangle, q) = \begin{cases} i_1 & \text{if } i_1 \neq i_2 \\ q & \text{otherwise} \end{cases}$$

Example 3.2 A flip-flop, the memory unit in sequential circuits, is a state transducer defined as $\langle \mathcal{B} \times \mathcal{B}, \mathcal{B}, 0, ff\rangle$ where

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

A *transducer* (Mealy Machine) is a tuple $\langle I, Q, q_0, f, O, f^o\rangle$ where $\langle I, Q, q_0, f\rangle$ is a state transducer, O is the set of outputs and $f^o : I \times Q \to O$ is the *output function.* A transducer can be represented by an open constraint net composed of a state transducer and a transliteration. A transducer defines a transduction from input traces to state traces and a transduction from input traces to output traces.

Generalizing, we can model a discrete dynamic system in a constraint net composed only of transliterations and unit delays. Any output location of a unit delay is called a *state* location. A discrete dynamic system modeled by a constraint net with transliterations and unit delays can be written as a set of equations where each left-hand side is an individual output location and each right-hand side is a function of locations. Let l' denote the next value at location l. There are two types of equations:

- $l'_s = l_{ns}$ if l_s is a state location of a unit delay from location l_{ns};
- $l_o = f(l_1, \ldots, l_n)$ if l_o is an output location of a transliteration $f_{\mathcal{T}}$ from locations $l_1, \ldots, l_n$.

Let S be the set of state locations, $O = O(CN) \setminus S$ be the set of other output locations, $I = I(CN)$ be the set of input locations and A be the domain of values. Generally, a constraint net on a discrete time structure can be written as two sets of equations: $s' = f_s(i, s)$; $o = f_o(i, s)$ where $s : S \rightarrow A$ is a state tuple, $i : I \rightarrow A$ is an input tuple, f_s is a tuple of transition functions, $o : O \rightarrow A$ is an output tuple, and f_o is a tuple of output functions. Therefore, such a constraint net is globally a transducer $\langle \mathcal{I}, \mathcal{S}, s_0, f_s, \mathcal{O}, f_o \rangle$ where $\mathcal{I} = \{i | i : I \rightarrow A\}$ is the set of inputs, $\mathcal{S} = \{s | s : S \rightarrow A\}$ is the set of states, s_0 is the initial state, f_s is the state transition function, $\mathcal{O} = \{o | o : O \rightarrow A\}$ is the set of outputs and f_o is the output function.

3.2 Modules

A *module* is a pair $\langle CN, O \rangle$ where CN is a constraint net and $O \subseteq O(CN)$ is a subset of the output locations of CN; O and $I(CN)$ define the *interface* of the module. For example, a state transducer (Fig. 5) can be encapsulated as a module with I and Q as the interface. Complex modules can be hierarchically constructed from simple ones using three operations: *composition*, *coalescence* and *hiding* [20].

Example 3.3 The Hand Coordinator: The hand coordinator can be designed using negated Muller-C elements [18]. Let NC be the module representing the element; the hand coordinator is a composition of two modules as in Fig. 6.

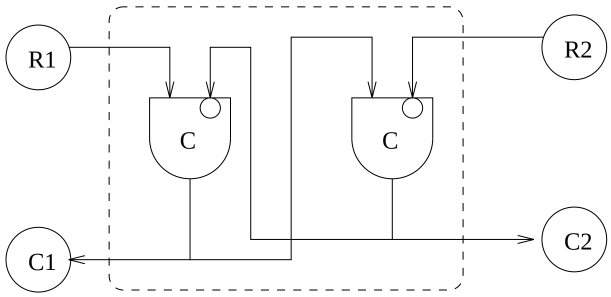

Figure 6: The hand coordinator (where $\circ$ denotes the second input port of NC)

The set of equations of this net is: $C1' = nc(\langle R1, C2\rangle, C1), C2' = nc(\langle C1, R2\rangle, C2)$ where $R1$ and $R2$ are input locations, $C1$ and $C2$ are state locations. The initial state is: $C1 = 0, C2 = 0$. We will see that this circuit works as a synchronizer in event logic; an event is signaled by a transition from 0 to 1 or 1 to 0.

Example 3.4 The Maze Traveler: The controller for the maze traveler can be built using AND, OR and NOT gates with a flip-flop memory unit (Fig. 7).

The robot is the integration of the controller and the robot body where SE and SN are input locations, FF, X and Y are state locations, and ME, MN and MS are the other output locations. The body can be modeled as a state transducer $\langle \mathcal{B} \times \mathcal{B} \times \mathcal{B}, \mathcal{Z} \times \mathcal{Z}, \langle x_0, y_0\rangle, f_{xy}\rangle$ where $\mathcal{Z}$ is the set of integers and $f_{xy}(\langle ME, MN, MS\rangle, \langle X, Y\rangle) = \langle X + ME, Y + MN - MS\rangle$, i.e. the next $\langle X, Y\rangle$ position is displaced by a grid point depending on the motion commands. The output functions of the robot are $ME = \neg SE, MN = \neg(FF \vee SN) \wedge SE$ and $MS = FF \wedge SE$. The state transition functions of the robot are $FF' = ff(\langle SN, \neg SE\rangle, FF)$ and $\langle X', Y'\rangle = f_{xy}(\langle ME, MN, MS\rangle, \langle X, Y\rangle)$, or, with the right-hand side composed of only state and input locations, $\langle X', Y'\rangle = f_{xy}(\langle \neg SE, \neg(FF \vee SN) \wedge SE, FF \wedge SE\rangle, \langle X, Y\rangle)$.

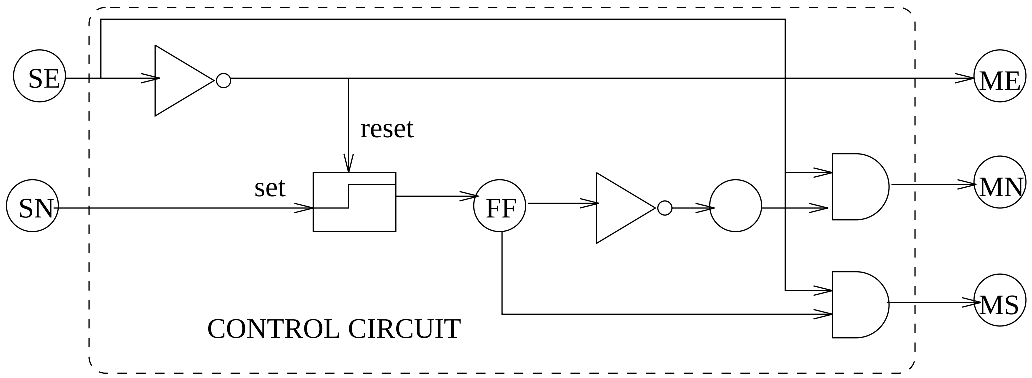

Figure 7: The controller for the maze traveler

In the next two sections, we will develop a logical specification language for representing the desired behaviors of a system and a verification method for ensuring that the model of the system satisfies the specification.

4 Specification in Real-Time Temporal Logic

While modeling focuses on the underlying structure of a system, and the organization and coordination of the components or subsystems, the overall behavior of the modeled system is not explicitly expressed. For most nonlinear systems, there is no closed form solution so that the global behavior can only be analyzed via simulation. A logical specification imposes constraints on the system's global behaviors. In this section, we develop a first order temporal logic as a specification language and establish the relationship between a constraint net and a logical specification. For simplicity, we still consider discrete time structures only. However, the logic we develop here can be extended to dense time structures without substantial modification.

4.1 Syntax and semantics

A first order temporal logic $L(A)$ on a multi-sorted algebra A is defined as follows. Let X_l be a set of local variables (locations) whose values change over time and X_g be a set of global variables (parameters) whose values are constant over the time. A term in L is a term in A on the set of variables $X_l \cup X_l' \cup X_g$. A *state formula* Fs is a first-order formula:

$$Fs \equiv false \mid T_1 = T_2 \mid p(T_1, \ldots, T_n) \mid Fs_1 \rightarrow Fs_2 \mid \exists x Fs$$

where T_i is a term, p is a predicate and $x \in X_g$ is a global variable. A formula F of the logic is recursively defined as:

$$F \equiv Fs \mid F_1 \rightarrow F_2 \mid F_1 \mathcal{U} F_2 \mid F_1 \mathcal{U}^n F_2$$

where n is a natural number. If F is a formula in L, $FV(F)$ denotes the set of free variables in F.

A frame $\mathcal{F}_r$ of the logic $L(A)$ is a pair $\langle A, I \rangle$ where I is an interpretation which maps each predicate p to a set of tuples in A. A model $\mathcal{M}$ of a formula F is a pair $\langle \mathcal{F}_r, \sigma \rangle$ where σ is a mapping from $FV(F)$ to variable traces. Let v_i be a valuation of the free variables at a given time i, i.e. $v_i(x) = \sigma(x)(i), v_i(x') = \sigma(x)(i+1)$, and v_i^* be the valuation extended to terms. A model $\mathcal{M}$ satisfies a formula F at time i, written $\mathcal{M} \models_i F$, as follows:

- $\mathcal{M} \not\models_i false$;
- $\mathcal{M} \models_i T_1 = T_2$ iff $v_i^*(T_1) = v_i^*(T_2)$;
- $\mathcal{M} \models_i p(T_1, \ldots, T_n)$ iff $(v_i^*(T_1), \ldots, v_i^*(T_n)) \in I(p)$;
- $\mathcal{M} \models_i F_1 \rightarrow F_2$ iff $\mathcal{M} \models_i F_1$ implies $\mathcal{M} \models_i F_2$;
- $\mathcal{M} \models_i \exists x F$, $x \in X_g$ iff $\exists v \in A$, $\mathcal{M} \models_i F[v/x]$, where $F[v/x]$ stands for substitution;
- $\mathcal{M} \models_i F_1 \mathcal{U} F_2$ iff $\exists i', i' > i$ and $\mathcal{M} \models_{i'} F_2$ and $\forall i'', i < i'' < i', \mathcal{M} \models_{i''} F_1$ ($\mathcal{U}$ is the basic temporal operator meaning "until");
- $\mathcal{M} \models_i F_1 \mathcal{U}^n F_2$ iff $\exists i', i < i' \leq i + n$ and $\mathcal{M} \models_{i'} F_2$ and $\forall i'', i < i'' < i', \mathcal{M} \models_{i''} F_1$ ($\mathcal{U}^n$ is the basic real-time temporal operator).

The syntax of the logic can be extended by following operators:

- $\neg F \equiv F \rightarrow false$: $\mathcal{M} \models_i \neg F$ iff $\mathcal{M} \not\models_i F$; $true \equiv \neg false$: $\mathcal{M} \models_i true$;
- $\diamond F \equiv true\mathcal{U}F$: $\mathcal{M} \models_i \diamond F$ iff $\exists i', i' > i$ and $\mathcal{M} \models_{i'} F$ ("eventually");
- $\Box F \equiv \neg(\diamond(\neg F))$: $\mathcal{M} \models_i \Box F$ iff $\forall i', i' > i$ implies $\mathcal{M} \models_{i'} F$ ("always afterwards");
- $\bigcirc F \equiv false\mathcal{U}F$: $\mathcal{M} \models_i \bigcirc F$ iff $\mathcal{M} \models_{i+1} F$ ("next");
- $\diamond^n F \equiv true\mathcal{U}^n F$, F will be true some time within n in the future;
- $\Box^n F \equiv \neg(\diamond^n \neg F)$, F will always be true within n in the future.

The logical specification is a powerful language for specifying various qualitative behaviors of a real-time embedded system. Some typical desired behaviors are:

- *goal achievement*: if P is the state formula for a desired goal, $\diamond\Box P$;
- *safety*: if P is the state formula for a dangerous situation, $\Box(\neg P)$;
- *real-time response*: if E is an event and R is the response, $\Box(E \rightarrow \diamond^n R)$.

Example 4.1 The Hand Coordinator*:* Let $E(X)$ denote that there is an event, a transition from 1 to 0 or vice versa, at X. If an event at $C1$ ($C2$) is the signal for holding a jar (fitting a cap), the desired behavior for the hand coordinator is $\mathcal{P} \wedge \mathcal{I}_1 \wedge \mathcal{I}_2$ where $\mathcal{P} \equiv \diamond E(C2) \rightarrow \neg E(C2)\mathcal{U}E(C1)$, $\mathcal{I}_1 \equiv \Box\neg(E(C1) \wedge (\neg E(C2)\mathcal{U}E(C1)))$ and $\mathcal{I}_2 \equiv \Box\neg(E(C2) \wedge (\neg E(C1)\mathcal{U}E(C2)))$. $\mathcal{P}$ states that jar holding by the left hand must *precede* cap fitting by the right hand. $\mathcal{I}_1$ and $\mathcal{I}_2$ state that the acts of holding and fitting must *interleave*.

Example 4.2 The Maze Traveler: We give a precise definition of the desired behavior for the maze traveler. It can be stated as a *liveness* property: $\Box\diamond ME$ meaning that the robot will persistently move east.

4.2 Satisfaction of a specification by a constraint net

A formula $F \in L(A)$ is a *specification* of $CN \equiv \langle Lc, Td, Cn\rangle$ iff $FV(F) \subseteq Lc$. A model of F w.r.t. CN is $\langle \mathcal{F}_r, \sigma\rangle$ such that σ is consistent with the semantics of CN, i.e. $\forall o \in O(CN) \cap FV(F)$, $\sigma(o) = F_o(\sigma(i))$ where F_o is the transduction corresponding to the output location o. CN *satisfies* F, written $CN \models F$, iff for all models $\mathcal{M}$ of F w.r.t. CN, $\mathcal{M} \models F$, where $\models$ abbreviates $\models_0$. One of the simple properties of the satisfaction relation is that if $CN \models F_1$ and $CN \models F_2$ then $CN \models F_1 \wedge F_2$. So a complex formula can be decomposed into simple ones which can be verified separately.

5 Verification by Model Checking

$\forall$-automata developed in [8] are more expressive than linear discrete temporal logics. Furthermore, there is a general verification method for a $\forall$-automaton specification of a concurrent program. In this section, we first introduce $\forall$-automata and the verification method modified from [8] with concurrent programs replaced by constraint nets, and then extend $\forall$-automata, and the verification method, to timed $\forall$-automata by adding timed states.

5.1 $\forall$-automata

A $\forall$-automaton [8] $\mathcal{A}(A)$ 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*. Let $L_s(A) \subset L(A)$ be the set of state formulas, $e : Q \to L_s(A)$ is an *entry condition* function such that $\bigvee_{q\in Q} e(q) = true$, and $c : Q \times Q \to L_s(A)$ is a *transition condition* function such that $\forall q \in Q, \bigvee_{q'\in Q} c(q, q') = true$. A *run* of $\mathcal{A}$ over a model $\mathcal{M}$ on a discrete time structure is a sequence of states $q_0, q_1, q_2, \ldots$ such that (1) $\mathcal{M} \models_0 e(q_0)$; and (2) for all time points $i > 0$, $\mathcal{M} \models_i c(q_{i-1}, q_i)$. Let $Inf(r) \subseteq Q$ denote the set of automaton states that appear infinitely many times in r. A run r is defined to be *accepting* iff: (a) $Inf(r) \cap R \neq \emptyset$; or (b) $Inf(r) \subseteq S$. A $\forall$-automaton $\mathcal{A}$ *accepts* a model $\mathcal{M}$, written

$\mathcal{M} \models \mathcal{A}$, iff *all* possible runs of $\mathcal{A}$ over $\mathcal{M}$ are accepting. $\mathcal{A}$ accepts a constraint net CN, written $CN \models \mathcal{A}$, iff $\mathcal{A}$ accepts all models of CN. It has been shown [8] that the specification power of $\forall$-automata is identical to that of ETL, an extended linear discrete temporal logic [19]. Therefore, for a discrete time structure every formula without real-time operators in $L(A)$ can be expressed as a specification in a $\forall$-automaton $\mathcal{A}(A)$.

A graphical representation of a $\forall$-automaton is a directed graph with nodes as automaton states and arcs as transitions. Each state in R is marked with $\Diamond$ and each state in S is marked with $\Box$. If $e(q) \neq false$, there are arrows to node q. If $c(q, q') \neq false$, there are arcs from q to q'. Each node, arrow or arc is labeled by a state formula F, such that $e(q) = (\bigvee_{a \in Arrows} Fa) \wedge Fq$ and $c(q, q') = (\bigvee_{a \in Arcs} Fa) \wedge Fq'$. By default, nodes, arrows or arcs without labels are labeled with the formula $true$.

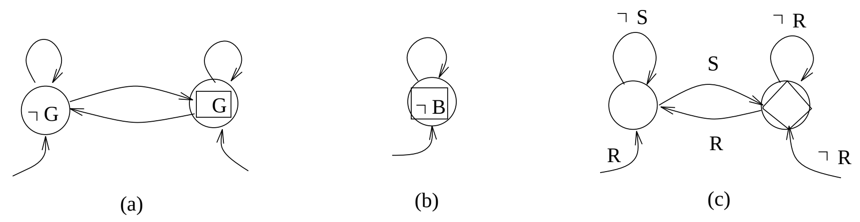

Figure 8: $\forall$-automata: (a) $\Diamond\Box P$ (b) $\Box\neg P$ (c) $\Box(E \rightarrow \Diamond R)$

Some examples of $\forall$-automata and the corresponding formulas are shown in Fig. 8. Fig. 8(a) accepts a computation which satisfies $\neg P$ only finitely many times, Fig. 8(b) accepts a computation which never satisfies P and Fig. 8(c) accepts a computation which will satisfy R in the finite future whenever it satisfies E.

5.2 Model checking

Let CN be a constraint net which is semantically represented as a transducer $\langle \mathcal{I}, \mathcal{S}, s_0, f_s, \mathcal{O}, f_o \rangle$. We write $\{\varphi\}CN\{\psi\}$ to denote the verification condition:

$$\varphi[f_o(i,s)/o] \bigwedge s' = f_s(i,s) \rightarrow \psi[i'/i, s'/s, f_o(i',s')/o]$$

where φ and ψ are state formulas.

Model checking of a constraint net CN against its specification $\mathcal{A}(A)$ involves two phases: Phase 1: Associate with each automaton state $q \in Q$ an assertion $\alpha_q \in L_s(A)$, called the *invariant* at q, such that the following requirements are satisfied:

- *Initiality*: $[s_0 \wedge e(q)] \rightarrow \alpha_q, \forall q \in Q$.
- *Consecution*: $\{\alpha_q\}CN\{c(q,q') \rightarrow \alpha_{q'}\}$, $\forall q, q' \in Q$.

Phase 2: Associate with each automaton state $q \in Q$ a ranking function $\rho_q : \mathcal{I} \times \mathcal{S} \rightarrow \mathcal{W}$, where $\mathcal{W}$ is a *well-founded* set, i.e., $\forall w_0 \in \mathcal{W}$, any decreasing sequence $w_0 > w_1 > \ldots$ is finite, such that the following requirements are satisfied:

- *Definedness*: $\alpha_q \rightarrow \exists w. \rho_q = w, \forall q \in Q$.
- *Non-increase*: $\{\alpha_q \wedge \rho_q = w\}CN\{c(q,q') \rightarrow \rho_{q'} \leq w\}, \forall q' \in S$.
- *Decrease*: $\{\alpha_q \wedge \rho_q = w\}CN\{c(q,q') \rightarrow \rho_{q'} < w\}, \forall q' \in Q \backslash (R \cup S)$.

It has been proven that these verification rules are sound and complete [8].

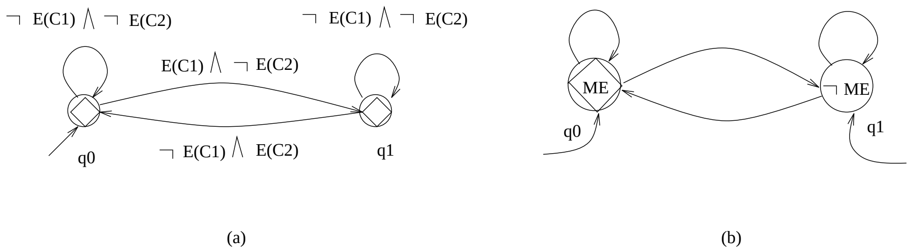

Figure 9: The specification of (a) the hand coordinator; (b) the maze traveler

Example 5.1 The Hand Coordinator: The desired behavior of the coordinator can be expressed in one $\forall$-automaton (Fig. 9(a)). The model checking is done in two phases. Phase 1: Associate with q_0, q_1, q_2 the invariants $C1 = C2$, $C1 \neq C2$ and $false$ respectively. The following verification conditions are satisfied:

- *Initiality:*

  $q_0: \quad (C1 = 0) \wedge (C2 = 0) \wedge true \rightarrow C1 = C2$

  $q_1: \quad (C1 = 0) \wedge (C2 = 0) \wedge false \rightarrow C1 \neq C2$

- *Consecution:* Let HC be $[C1' = nc(\langle R1, C2\rangle, C1)] \bigwedge [C2' = nc(\langle C1, R2\rangle, C2)]$,

  $(q_0, q_0): \quad C1 = C2 \wedge HC \rightarrow (\neg E(C1') \wedge \neg E(C2') \rightarrow C1' = C2')$

  $(q_0, q_1): \quad C1 = C2 \wedge HC \rightarrow (E(C1') \wedge \neg E(C2') \rightarrow C1' \neq C2')$

  $(q_0, q_2): \quad C1 = C2 \wedge HC \rightarrow (E(C2') \rightarrow false)$

  $\ldots$

  where $E(X') \equiv X' \neq X$.

Phase 2: since $q_0, q_1 \in R$ and the invariant of q_2 is $false$, the verification conditions are trivially satisfied.

Example 5.2 The Maze Traveler*:* The desired behavior of the maze traveler can be represented by the $\forall$-automaton in Fig. 9 (b). The first phase is trivially satisfied since $c(q, q') = \alpha_{q'}, e(q) = \alpha_q$ for any q, q' in this example. For the second phase, suppose the maximum length of an obstacle is L. Associate with each automaton state a ranking function $\rho : \mathcal{B} \times \mathcal{B} \times \mathcal{Z} \rightarrow \mathcal{B} \times \mathcal{D}$ where $\mathcal{D}$ is the interval $[0, L+1]$ of natural numbers. The ranking function is defined as:

$$\rho(SE, FF, Y) = \begin{cases} \langle 1, 1+L\rangle & \text{if } \neg SE \\ \langle 1, DN - Y\rangle & \text{if } SE \wedge \neg FF \\ \langle 0, Y - DS\rangle & \text{if } SE \wedge FF \end{cases}$$

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 $\mathcal{B} \times \mathcal{D}$ is defined as: $\langle 0, -\rangle < \langle 1, -\rangle$ and $\langle X, Y1\rangle \leq \langle X, Y2\rangle$ iff $Y1 \leq Y2$. $\mathcal{B} \times \mathcal{D}$ is a well-founded set since L is finite. With this ranking function and the well-founded set, any transition that ends up at $q_1 \in Q \backslash (R \cup S)$ would lead to a decrease in ranking. To see this, let MT be

$$[FF' = ff(\langle SN, \neg SE\rangle, FF)] \bigwedge [\langle X', Y'\rangle = f_{xy}(\langle \neg SE, \neg(FF \vee SN) \wedge SE, FF \wedge SE\rangle, \langle X, Y\rangle)].$$

The following two requirements are satisfied:

$$(q_0, q_1) : SE = 0 \wedge \rho(SE, FF, Y) = w \wedge MT \rightarrow (SE' = 1 \rightarrow \rho(SE', FF', Y') < w)$$

and

$$(q_1, q_1) : SE = 1 \wedge \rho(SE, FF, Y) = w \wedge MT \rightarrow (SE' = 1 \rightarrow \rho(SE', FF', Y') < w).$$

5.3 Timed $\forall$-automata

To guarantee real-time response, we extend $\forall$-automata to timed $\forall$-automata by adding a new class of automaton state, *timed states*: $T \subseteq Q$. Associated with any $q \in T$, there is a natural number n such that a run is accepting if, in addition, for any run segment starting with $q \in T$, there is $q' \neq q$, such that the length of the segments $q, q, \ldots q, q'$ is bounded by $n + 1$. Graphically, a T state is denoted by the natural number n. Any formula in $L(A)$ can be represented by a timed $\forall$-automaton $\mathcal{TA}(A)$ for any discrete time structure. For example, $F_1 \mathcal{U}^n F_2$ is represented as a timed $\forall$-automaton in Fig. 10.

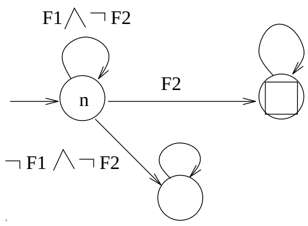

Figure 10: The timed $\forall$-automaton for $F_1 \mathcal{U}^n F_2$

Example 5.3 The real-time response $\Box(E \rightarrow \Diamond^n R)$ can be depicted by the timed $\forall$-automaton in Fig. 11 (a) and the real-time behavior of the maze traveler $\Box(\neg ME \rightarrow \Diamond^{2L} ME)$ is shown in Fig. 11 (b).

Model checking for timed automata is augmented with a phase checking the boundedness. Phase 3: Associate with each $q \in T$ a timing function, $\gamma_q : \mathcal{I} \times \mathcal{S} \rightarrow \mathcal{N}$ where $\mathcal{N}$ is the set of natural numbers, such that the following requirements are satisfied:

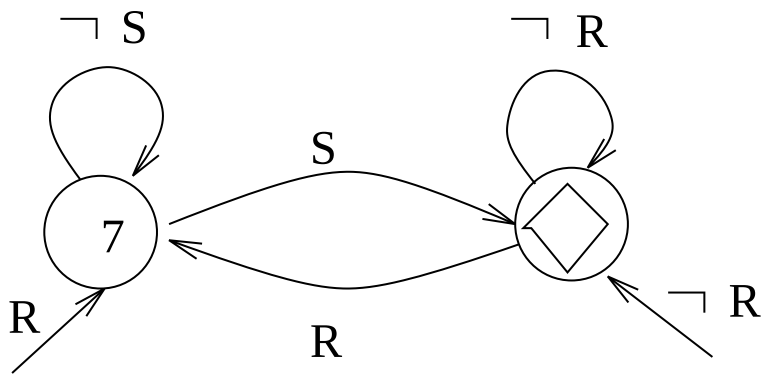

Figure 11: (a) Real-time response (b) Real-time behavior of the maze traveler

- *Boundedness*: $\alpha_q \rightarrow 1 \leq \gamma_q \leq n_q, \forall q \in T$;
- *Decrease*: $\{\alpha_q \wedge \gamma_q = w\} CN \{c(q,q) \rightarrow \gamma_q < w\}, \forall q \in T$.

6 Conclusion and Further Work

Will the robot do the right thing? One can guarantee the answer yes by modeling the complete robotic system at an appropriate level of abstraction and proving that the model satisfies the desired behavioral specification. We are developing an integrated environment known as Alert (A laboratory for embedded real-time systems) for modeling robotic systems with constraint nets. We intend to present tools for the specification and verification of robotic systems based on the results in this paper, in the future.

Acknowledgements

This research was supported by the Natural Sciences and Engineering Research Council and the Institute for Robotics and Intelligent Systems.

References

[1] 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.

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

[3] 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.

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

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

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

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

[8] Z. Manna and A. Pnueli. Specification and verification of concurrent programs by $\forall$-automata. In *Proc. 14th Ann. ACM Symp. on Principles of Programming Languages*, pages 1–12, 1987.

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

[10] 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.

[11] G. H. Mealy. A method for synthesizing sequential circuits. *Bell Sys. Tech. Journal*, 34:1045 – 1079, 1955.

[12] E. F. Moore. Gedanken-experiments on sequential machines. In C.E. Shannon and J. McCarthy, editors, *Automata Studies*. Princeton University Press, 1956.

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

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

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

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

[17] Y. Shoham. Agent-oriented programming. Technical Report STAN-CS-1335-90, Computer Science Department, Stanford University, 1990.

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

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

[20] Y. Zhang and A. K. Mackworth. Constraint Nets: A semantic model of real-time embedded systems. Technical Report 92-10, Department of Computer Science, University of British Columbia, 1992.