Formal Specification of Performance Metrics for Intelligent Systems

Ying Zhang
System and Practice Lab, Xerox Palo Alto Research Center
Palo Alto, CA 94304
Alan K. Mackworth
Department of Computer Science, University of British Columbia
Vancouver, B.C., Canada, V6T 1Z4
Email: yzhang@parc.xerox.com, mack@cs.ubc.ca

ABSTRACT

There are now so many architectures for intelligent systems: deliberative planning vs. reactive acting, behavioral subsuming vs. hierarchical structuring, machine learning vs. logic reasoning, and symbolic representation vs. procedural knowledge. The arguments from all schools are all based on how natural systems (i.e., biologically inspired, from basic forms of life to high level intelligence) work by taking the parts that support their architectures. In this paper, we take an engineering point of view, i.e., by using requirements specification and system verification as the measurement tool. Since most intelligent systems are real-time dynamic systems (all lives are), requirements specification should be able to represent timed properties. We have developed timed ∀-automata that fit to this purpose. We will present this formal specification, examples for specifying requirements and a general procedure for verification.

KEYWORDS: *formal specification, constraint-based requirements, system verification*

1. INTRODUCTION AND MOTIVATION

Over the last half a century, intelligent systems have become more and more important to human society, from everyday life to exploration adventures. However, unlike most other engineering fields, there has been little effort towards developing sound and deep foundations for quantitatively measurement and understanding such systems. The lack of measurement and understanding leads to unsatisfactory behavior or even potential danger for customers. The systems may not achieve desired performance in certain environments, or, the systems may even result in catastrophe in life-critical circumstances.

Many researchers have suggested measures of performance for intelligent systems, such as the Turing Test [12], Newell's expanded list [9,10] and Albus's definition of intelligence [4]. However, most of these measures are not based on formal quantitative metrics. There are also efforts on comparing performance on pre-defined tasks, such as a soccer competition [11]. However, these methods are domain specific therefore hard to apply to general cases. We advocate formal methods for specifying performance requirements of intelligent systems. Much research has been done on formal methods (http://archive.comlab.ox.ac.uk/formal-methods.html) over the last twenty years. In this paper, we explore one of the approaches, namely, using timed ∀-automata for specifying performance requirements.

The timed ∀-automata model was developed in [13, 17] as an extension of discrete time ∀-automata [8] to continuous time, annotations with real-time. Timed ∀-automata are simple yet able to represent many important features of dynamic systems such as safety, stability, reachability and real-time response. In the rest of this paper, we introduce the formal definition of timed ∀-automata first, then present examples of timed ∀-automata for representing performance metrics, and finally describe a general verification procedure for this type of requirements specification.

2. TIMED ∀-AUTOMATA

In general, there are two uses of automata: 1. to describe computations, such as input/output state automata, and 2. to characterize a set of sequences, such as regular grammars/languages. Examples of the first category are mostly deterministic and examples of the second category are mostly non-deterministic. However, all the original automata work is based on discrete time steps/sequences. Approaches to extending automata to continuous time have been explored in hybrid systems community over the last decades [1,2,7]. The timed ∀-automata model that we developed belongs to the second category, i.e., *non*-deterministic *finite* state automata specifying behaviors over *continuous* time. The discrete time version of ∀-automata was originally proposed as formalism for the specification and verification of temporal properties of concurrent programs [8].

2.1. *Syntax*

Syntactically, a timed ∀-automaton is defined as follows.

[**Definition 1**] A ∀-*automaton* A is a quintuple (Q, R, S, e, c) where Q is a finite set of automaton-states, R ⊆ Q is a set of recurrent states and S ⊆ Q is a set of stable states. With each q ∈ 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' ∈ 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 generalizations of accepting states. We denote by B = Q – (R ∪ S) the set of non-accepting (bad) states. Let R^+ be the set of non-negative real numbers representing time durations. A *timed ∀-automaton* is a triple (A, T, τ) where A is a ∀-automaton, T ⊆ Q is a set of timed automaton-states and τ: T ∪ {B} → R^+ ∪ {∞} is a time function.

One of the engineering advantages of using automata as a specification language is its graphical representation. It is useful and illuminating to represent timed ∀-automata by diagrams. A timed ∀-automaton can be depicted by a labeled directed graph, where automaton-states are depicted by circle nodes and transition relations are by directional arcs. In addition, each automaton-state may have an entry arc pointing to it; each recurrent state is depicted by a diamond and each stable state is depicted by a square, inscribed within a circle. Nodes and arcs are labeled by assertions as follows. A node or an arc that is left unlabeled is considered to be labeled with **true**. Furthermore, (1) if an automaton-state q is labeled by ψ and its entry arc is labeled by φ, the entry condition e(q) is given by e(q) = ψ ∧φ; if there is no entry arc, e(q) = **false**, and (2) if arcs from q to q' are labeled by $φ_i$, i = 1…n, and q' is labeled by ψ, the transition condition c(q, q') is given by c(q, q') = ($φ_1$ ∨…∨$φ_n$) ∧ψ; if there is no arc from q to q', c(q, q') = **false**. A T-state is denoted by a nonnegative real number indicating its time bound. Some examples of timed ∀-automata are shown in Figure 1.

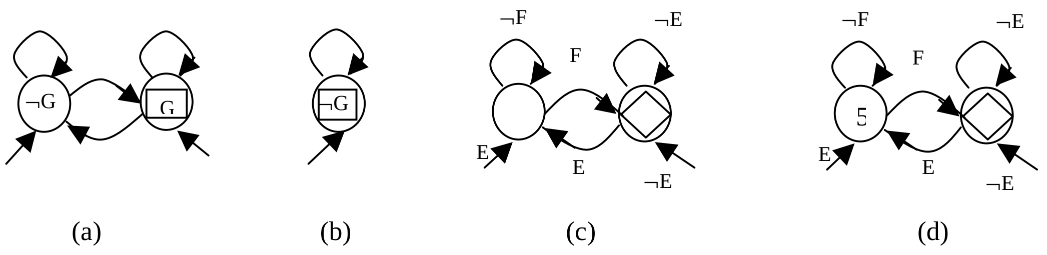

Figure 1. Examples of timed ∀-automata

2.2. *Semantics*

Semantically, each assertion denotes a constraint defined on a domain of interest. Let D be a domain of interest; D can be finite, discrete, or continuous, or a cross product of a finite number of domains. Physically, D can represent, for example, speeds, distances, torques, sentences, commands or a combination of the above. A constraint C defined on D is a subset of D, C ⊆ D. Physically, a constraint represents certain relation on a domain, such as a relation between external environment stimuli and an agent's internal knowledge representation, or, a relation between internal states and actions, or, the relation between the current and next state. An element d in domain D satisfies constraint C, if and only if d ∈ C.

The semantics of timed ∀-automaton is defined as follows. Let *T* be a time domain, which can be continuous, for example, R^+. First, let us define runs of ∀-automata. Let A = (Q, R, S, e, c) be a ∀-automaton and v: *T* → D be a function of time. A *run* of A over v is a function r: *T* →Q satisfying:

1. *Initiality*: v(0) ∈ e(r(0));
2. *Consecution*:
   a. *Inductivity*: ∀t>0, ∃q∈Q, t'<t,∀t'', t'≤t''<t, r(t'')=q and v(t) ∈ c(r(t''), r(t)) and
   b. *Continuity*: ∀t, ∃q∈Q, t'>t, ∀t'', t<t''<t', r(t'')=q and v(t'') ∈ c(r(t), r(t'')).

   When *T* is discrete, the two conditions in *Consecution* reduce to one, i.e., ∀t>0, v(t) ∈ c(r(pre(t)), r(t)) where pre(t) is the previous time point of t.

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|∀t∃t'≥t, r(t')=q}. A run is called *accepting* if and only if

1. Inf(r) ∩R≠0, i.e., some of states appearing infinitely many times in r belong to R, or
2. Inf(r) ⊆ S, i.e., all the states appearing infinitely many times in r belong to S.

For a timed ∀-automaton, in addition for a run to be accepting, it has to satisfy time constraints. Let I ⊆ *T* be a time interval and |I| be the time measurement, and let r|I be a segment of r over time interval I. A run satisfies time constraints if and only if:

1. *Local*: For any $q \in T$ any time interval I, if $r|I$ is a segment of consecutive states of q, then $|I| \leq \tau(q)$;
2. *Global*: For any time interval I, if $r|I$ is a segment of consecutive states of $B \cup S$, then $\int_I \chi_B(r(t))dt \leq \tau(B)$, where $\chi_{B:}\ Q \rightarrow \{0,1\}$ is the characterization function for the set B.

[**Definition 2**] A timed ∀-automaton TA = (A, T, τ) accepts a trace v, if and only if

1. All runs are accepting for A;
2. All runs satisfy the time constraints.

With the semantics defined, we can infer that, for the timed ∀-automata in Figure 1, (a) specifies the behavior of reachability, i.e., eventually the system should satisfy constraint G, (b) specifies the behavior of safety, i.e. constraint G is never satisfied, (c) specifies the behavior of bounded response, i.e., whenever constraint E is satisfied, constraint F will be satisfied within bounded time and (d) specifies the behavior of real-time response, i.e., whenever constraint E is satisfied, constraint F will be satisfied within 5 time units.

3. EXAMPLES OF PERFORMANCE SPECIFICATION

Timed ∀-automata are simple yet powerful for the specification of behaviors of dynamic systems, since it integrates constraint specification with timed dynamic behavior specification.

3.1. *Examples of Constraint Specification*

Constraint specification alone can specify many performance metrics. Constraints specify relations between external environment stimuli and an agent's internal knowledge representation, or between internal states and actions, or between the current and next states. Constraints can be finite, discrete or continuous, or any combination of the above. Constraints can be linear, nonlinear, equalities or inequalities. Moreover, constraints can also specify optimal conditions or optimality with extra constraints, or combinations of multiple optimal criteria and additional constraints.

Considering the following examples for specifying constraints:

1. *Inequality:* $f(x) \leq 0$ where x is a vector of variables and f is a vector of functions.
2. *Optimality*: $\min |f(x)|$ where $|x|$ is a norm for x.
3. *Negation*: $x \neq y$.
4. *Constrained Optimality*: $\min|f(x)|$ given $g(x) \leq 0$.
5. *Robustness*: Let f(x) be a set of output functions with x as inputs. The robustness can be represented by its Jacobian $J = \Delta f/\Delta x$. There are many ways to state an optimal condition for robustness. One method is to minimize $|w|$ where w is the diagonal elements of W in the singular value decomposition of $J = UWV^T$.

3.2. Examples of ∀-Automata

With automata, timed dynamic behaviors can be specified. Here is a set of examples for specifying performance using timed ∀-automata, as shown in Figure 1:

1. Let G be a constraint that the distance between the robot and its desired position is less than some constant value. Then Figure 1(a) specifies that the robot will eventually arrive its desired position.
2. Let G be a constraint that the error of a learning algorithm is less than a desired tolerance. Then Figure 1(a) specifies that the learning will eventually convergence. If let the state of ¬G in Figure 1(a) as a timed state with time bound t, it further specifies that the learning will be done within time t.
3. Let G be a constraint that the distance between the robot and obstacles is less than some constant value. Then Figure 1(b) specifies that the robot will never hit any obstacle. If G denotes that the current memory usage is out of the limit, Figure 1(b) specifies that the memory usage at any time is within its limit.
4. Let E be an external stimuli and F be a response. Then Figure 1(c) specifies that there is a response after stimuli within bounded time. Figure 1(d) specifies that such a response is within 5 time units.

Even though timed ∀-automata are powerful, still they are not able to represent all forms of performance metrics. For example, optimal performance over time $\min\int f(t)dt$ is not specifiable with timed ∀-automata. This form is mostly used for characterizing energy, efficiency or overall errors. Furthermore, specification with probability behaviors are not included either. However, it is not hard to add probability, for example, instead of "all runs" must be accepting and satisfying time constraints, we can say "x% runs" must be accepting and satisfying time constraints.

3.3 *Performance Comparisons*

Note that requirements specification defines what the system should do, rather than defining how the system is organized, i.e., its architecture. For example, behavior-based control [4,6] (which is arbitration based or a horizontal hierarchy) has a different form of architecture from function-based control [5] (which is abstraction-based or a vertical hierarchy); model-based systems have a different form of architecture from learning-based systems, event-driven systems have a different kind of architecture from time-driven systems. Different systems with different architectures can still be compared based on the behavioral interface under the formal performance specification. For example, given a set of requirements specification Rs and system A satisfies a subset As $\subseteq$ Rs and system B satisfies a subset Bs $\subseteq$ Rs. If As $\subseteq$ Bs, system A is not better than system B with respect to requirements Rs. Similarly, if system A satisfies requirement α and system B satisfies requirement β and if α implies β, system A is better than system B with respect to the requirement.

However, this specification does not define metrics on architectures. The measurement of performance should come from the customer's point of view, but the measurement of architecture should come from the developer's point of view, i.e., design time, debug time, upgrading time, modularity and the percentage of re-usable components.

4. SYSTEM VERIFICATION

For most dynamic systems, stability or convergence is the most important property that needs to be verified. For example, we can verify that equation $dx/dt = 0$ satisfies the property of $\forall$-automaton in Figure 1(a) with G as $|x|\leq\varepsilon$ for any positive number ε. The most commonly used method for the verification of such properties is the use of Liaponov functions. We developed a formal method based on model-checking, that generalizes Liaponov functions [13,17]. This method is automatic if the domain of interest is finite discrete and time is discrete [13].

The details of the model-checking method are out of the scope of this paper. The basic principle is to first find a set of invariants, each associated with an automaton-state in the timed $\forall$-automaton. Then, find a set of Liaponov functions, which are non-increasing in stable states and decreasing in bad states. Finally, find a set of local and global timing functions, where local timing functions are decreasing in timed states and global timing functions, like Liaponov functions, are non-increasing in stable states and decreasing in bad states, in addition to be bounded in values.

5. RELATED WORK AND CONCLUSION

Much work has been done in formal approaches to system specification and verification [1,2,7,8]. In general, there are two schools. One is to develop a uniform specification for both systems and their requirements; the other is to use two different specifications, one for systems and one for requirements. The advantage of the former is that the same formal approach can apply to both system synthesis and system verification. However, in most cases, if the specification language is powerful for both systems and requirements, the synthesis or verification tasks become hard. We advocate the latter approach, i.e., using timed $\forall$-automata for requirements specification and using Constraint Nets [13,18,19] for system modeling. Control synthesis [13,14] and verification [13,15,16,17,20] are also studied in this framework.

In this paper, we have shown how to use formal methods to specify the performance metrics of intelligent systems, with timed $\forall$-automata as an example. The advantage of formal methods over other methods lies in their precision and generality. Timed $\forall$-automata, with its graphical depiction and constraint specification, is a simple yet powerful formalism for specifying many properties of dynamic systems.

6. REFERENCES

[1] Alur, R., C. Courcoubetis, T.A Henzinger and P. Ho, "Hybrid automata: Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems*, LNCS 736, Springer-Verlag, 1993, pp. 209 – 229.

[2] Alur, R. and D. Dill, "Automata for modeling real-time systems," M.S. Peterson, editor, ICALP90: *Automata, Languages and Programming*, LNCS 443, Springer-Verlag, 1990, pp. 322 – 335.

[4] Albus, J.S., "Outline for a Theory of Intelligence," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 21, No. 3, pp. 473 – 509, May/June 1991.

[5] Arkin, R.C., *Behavior-Based Robotics*, The MIT Press, Cambridge, MA, 1998.

[6] Brooks, R.A., "Intelligence without reason," in IJCAI 1991, Sydney, Australia, pp. 569 – 595.

[7] Henzinger, T.A., Z. Manna and A. Pnueli, "Timed transition systems," J.W. deBakker, C. Huizing, W.P. dePoever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, LNCS 600, Springer-Verlag, 1991, pp. 226 – 251.

[8] Manna, Z. and A. Pnueli, "Specification and verification of concurrent programs by $\forall$-automata," in Proc. 14th Ann. ACM Symp. On Principles of Programming Languages, 1987, pp. 1-12.

[9] Newell, A., "The Knowledge Level," *Artificial Intelligence*, 18(1), pp. 87-127, 1982.

[10] Newell, A., and Simon, H., GPS: A Program that Simulates Human Thought," Feigenbaum and Feldman,

editors, *Computers and Thought*, McGraw-Hill, New York, 1963.

[11] Sahota, M. and A. K. Mackworth, "Can situated robots play soccer?" in Proc. Artificial Intelligence, 1994, Banff, Alberta, pp. 249 – 254.

[12] Turing, A. "Computing Machinery and Intelligence." *Mind 59*, pp. 433-460, 1950. Reprinted in Feigenbaum and Feldman, editors, *Computers and Thought*, McGraw-Hill, New York, 1963.

[13] Zhang, Y., "A Foundation for the Design and Analysis of Robotic Systems and Behaviors", PhD Thesis, University of British Columbia, Canada, 1994.

[14] Zhang, Y. and A. K. Mackworth, "Synthesis of Hybrid Constraint-Based Controllers," P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, *Hybrid Systems and Automatic Control*, LNCS 999, Springer-Verlag, 1994, pp. 552 – 567.

[15] Zhang, Y. and A. K. Mackworth "Specification and Verification of Constraint-Based Dynamic Systems," A. Borning, editor, *Principles and Practice of Constraint Programming*, LNCS 874, Springer-Verlag, 1994, pp. 229 – 242.

[16] Zhang, Y. and A. K. Mackworth, "Will The Robot Do The Right Thing," in Proc. Artificial Intelligence, 1994, Banff, Alberta, pp. 255 – 262.

[17] Zhang, Y. and A. K. Mackworth, "Specification and Verification of Hybrid Dynamic Systems Using Timed ∀-Automata," *Verification and Control of Hybrid Systems*, LNCS 1066, Springer-Verlag, 1995.

[18] Zhang, Y. and A. K. Mackworth, "Constraint Programming in Constraint Nets", V. Saraswat and P. Van Hentenryck, editors, *Principles and Practice of Constraint Programming*, MIT Press, Cambridge, MA, 1995, pp. 49 – 68.

[19] Zhang, Y. and A. K. Mackworth, "Constraint Nets: A Semantic Model for Hybrid Dynamic Systems," *Journal of Theoretical Computer Science*, Vol. 138, No. 1, pp. 211 – 239, 1995.

[20] Zhang, Y. and Alan K. Mackworth, "Modeling and Analysis of Hybrid Systems: An Elevator Case Study," H.Levesque and F.Pirri, editors, *Logic Foundations for Cognitive Agents*, Springer, Berlin, 1999, pp. 370-396.