A question about hierarchical systems*

Pravin Varaiya
Department of Electrical Engineering & Computer Sciences
University of California, Berkeley
varaiya@eecs.Berkeley.EDU

Abstract

The control of large systems is always organized in a distributed hierarchy. We discuss two approaches to understanding such a hierarchy. The approach propagated by work in verification adopts a 'one-world' semantics in which syntactical constructs at higher levels are 'compiled' into a single interpretation at the lowest level. Many hierarchical systems, however, are designed and analyzed using a 'multi-world' semantics, with a separate interpretation at each level. One-world semantics offers a sound way of stating and proving claims about the system, but multi-world semantics better conforms to practice. The paper poses the question: how to join the theoretical advantages of one-world semantics to the practical convenience of multi-world semantics.

1 Introduction

There are good reasons for organizing the control of large systems in a distributed hierarchy. Among these are: deeper understanding facilitated by the hierarchical structure, reduction in complexity of communication and computation, modularity and adaptability to change, robustness, and scalability. The fact of the matter is that the control of every large system is organized in a distributed hierarchy, so the question is not whether it is a good idea to control large systems in this way. The interesting questions are different: 'How to describe such systems in ways that make meaningful distinctions among different hierarchical, distributed control organizations?'; 'What approaches help to assess system performance?'; 'What tools and techniques aid in the design of good control organizations?'

The paper discusses a different question: How do we understand hierarchical systems? To do so, we begin with a a basic theoretical framework: a formal syntax for describing such systems, and associated formal and informal semantic structures for interpreting the syntax.

*This paper is dedicated to Sanjoy Mitter. It reflects some of his preoccupations—in the manner that I have understood them. Research supported by Office of Naval Research.

We explore two approaches. The first uses the 'one-world' semantics used in control theory and AI. The second is a 'multi-world' semantics to interpret a hierarchy.

In section 2 we present a a framework for a study of systems comprising three structures—syntax, semantics, and environment. In section 3 we discuss how the syntax needs to be augmented to describe a hierarchy, and in section 4 we discuss the one-world semantics for hierarchical descriptions. Section 5 presents a multi-world semantics, and poses our question. Section 6 suggests some responses.

2 Three structures of a systems framework

A theoretical framework for a study of systems requires three structures:

- Syntax—rules for *expressing* system descriptions and specifications;
- Semantics—mathematical model for *interpreting* syntactic expressions;
- Environment—real or simulated or imagined environment for *implementing* descriptions.

The table lists four example frameworks.

Name	Syntax	Semantics	Environment
LISP	LISP expressions and programs	Register machine	Computer
SHIFT	SHIFT programs	Hybrid automata networks	Simulation
Continuous systems	Systems of ode	Function spaces of trajectories, inputs and outputs	Vehicle, airplane
Discrete systems	Temporal logic expressions	Transition systems	Path planning

Example frameworks

We comment on each framework to better suggest what is meant by syntax, semantics, and environment. A study of LISP involves LISP syntax which determines the set of (well-formed or syntactically correct) LISP programs. The semantics or meaning of a LISP program may be supplied by describing how a the program would be executed in an ideal register machine [1]. This is 'ideal' in that the memory is perfect and the machine executions are flawless. More importantly it is ideal in the sense that the register machine is a mathematical object. The environment for a LISP

program might be a real computer that executes the program. In that computer there may be imperfections that cause its execution to be different from that of the semantics determined by the ideal machine. The possible differences between the real and the ideal executions cannot be captured *a priori*, because the real machine may at any time carry out unanticipated executions (think about the viruses that may present).

SHIFT is a language for simulating hybrid systems [3, 4]. As with LISP, the SHIFT syntax determines syntactically correct programs. The execution of a SHIFT program is interpreted as a computational run in a hybrid automaton [9]. The implementation of a SHIFT program—a simulation—is carried out in a PC or workstation that can serve as the environment.

A class of continuous systems may be described by a system of odes (ordinary differential equations). The syntax of odes is usually not formally stated, but there is an informal and unambiguous syntax and conventions among mathematically literate people to understand an ode in a unique way. The meaning of an ode is given by its solutions, i.e., the continuous state trajectories, and the input and output functions. Finally, an environment for an ode might be a physical object (such as a vehicle or airplane) of which the ode is a description. Observe that no matter how elaborate the ode might be, the real vehicle or airplane may behave differently from the solutions of the ode. The difference may arise because of modeling simplifications (eg. distributed parameter nature of the aircraft frame is ignored, or a kinematic model is used in place of a richer dynamical model) or because certain possibilities (eg. some faults and disturbances in the real vehicles or plane) are not accounted for in the odes.

Temporal logic expressions are used to describe a discrete system as well as to specify its behavioral requirements. Temporal logic expressions are interpreted (modeled) as behaviors of finite state machines or more general transition systems.[1] On the other hand the discrete system may be a description of, for example, a planning system that generates a plan for a mission as a sequence of maneuvers that an autonomous vehicle must follow. This system can be specified (syntactically) as a finite state machine (see, eg. [6]) and the implemented or simulated planning system serves as its environment.

Three relationships tie together the structures of a framework as depicted in Figure 1:

- Truth-claim relationship—assertions about syntactical descriptions are *proved* in semantics;
- Behavior-relevance—relevance of semantic behavior is *experienced* in environment;
- Granularity-fidelity relationship—details about environment *correspond* to primitive entities in the syntax.

We briefly illustrate these relationships in the context of a control system for an autonomous vehicle. Suppose the control law and the vehicle dynamics are described by an ode, and the meaning of the ode is carried by its state trajectories. A claim like ‘The control law is stable’, could then be proved

[1] There is a large literature on the use of temporal logic for specification and verification. An early reference is [2].

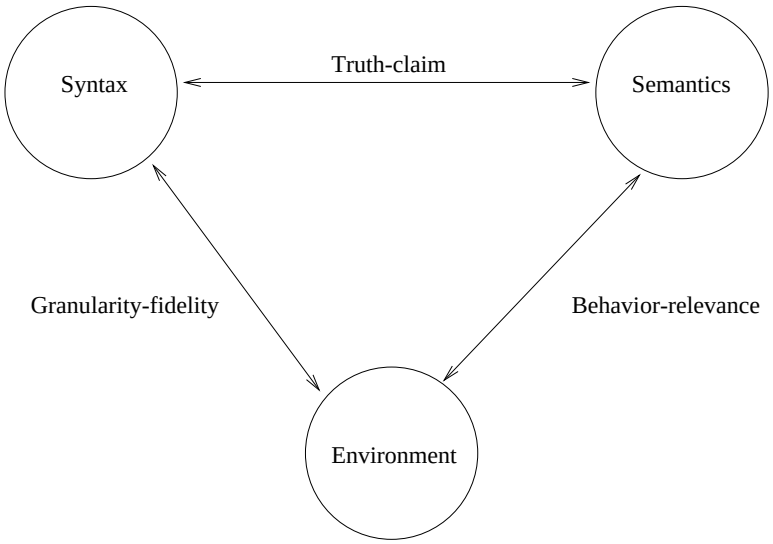

Figure 1: Relationships between structures

by showing that the state trajectories behave in such and such way (eg. the vehicle speed achieves its steady state value exponentially fast).

Second, the details or granularity-level of the primitive entities (used to build more complex expressions) must reflect the fidelity with which we wish to study the real system—its environment. The more details of the real system we wish to study, the richer the set of entities we need in the syntactic description. For example, if we wish to study the influence of the vehicle's sensors and actuators, then the system description obviously must incorporate elements that can be associated with those sensors and actuators.

3 Syntax for hierarchical systems

In order to describe a distributed, hierarchical system, the syntax needs to be enhanced to reflect those aspects of distributed hierarchy that we intuitively find to be essential. In terms of Figure 1 this need arises from the granularity-fidelity relationship. The first problem we will face is to circumscribe 'hierarchical, distributed systems' in a usefully precise way. The meaning of 'distributed system' is at least initially fairly clear: it refers to a system comprising several components or subsystems that are distinguished from each other by function or merely by identity. Thus we may have components that function as sensors, actuators, controllers, vehicles, etc. This is functional differentiation. Or we may merely have a collection of functionally identical microbots (agents) distinguished by name or location (identity). Of course, a distinguished functional component (eg. software agent or mental organ like speech processor) need not be implemented in a distinct physical component. So we should not associate 'functional component' of a distributed system with a distinct physical entity.

In summary, and as a preliminary step, we conclude that in order to describe distributed systems, the syntax must permit us to define different components, and name these entities. Such a naming facility provides the simplest and most essential means for abstraction. Additional syntactical facilities will be needed in order to reflect relationships between named entities. Object-oriented programming languages, for example, offer such facilities. However, a package for numerical integration of odes may not offer such a facility, and so by itself it would not meet this requirement.

The meaning of 'hierarchical system' is less clear. The term is used in several different ways: there are hierarchies of goals, authority, concepts, tasks, system composition, and classification. Thus we may say that safety or survival is a 'higher order' goal than efficiency, meaning thereby that in system operation efficiency should be considered only when safety is not threatened. We may also say that the leader of a group of agents has greater authority than a group's member, if the action of the former can override that of the latter. We may say that a concept like 'a path from A to B' is at a higher level than an action like 'move left or right one meter' since 'path' can be composed as a 'sequence of actions'. From this kind of concept hierarchy one obtains a task hierarchy: 'find a path from A to B' is a higher level task than 'move left three meters'.

We also sometimes refer to a higher level system description using say a block diagram if the diagram can be decomposed into a more detailed one. In object-oriented classification schemes, as in most taxonomies, we refer to more abstract classes being specialized into more concrete ones.

At the core of all these notions of hierarchical system is a syntax in which we can place a partial order on different entities that the syntax allows us to construct. The partial order may be one of containment, specialization, or abstraction. The syntactical facility that allows us to impose the partial order(s) should help us formulate our intuitive notions of hierarchy. We can draw upon the wide practice of structured programming for suggestions as to how to create such a facility.

For this reason we conclude that there is no obstacle to enhancing our syntax with the facility to express partial orders that describe hierarchy. The obstacles begin to appear when we come to the semantics. One approach to the meaning of hierarchy is to adopt what we call 'one-world' semantics.

As we will see, the advantage of this approach is that it provides a unique interpretation to system description and evaluation, it promotes good design practice, and in principle, it offers a direct path from system requirements to specification to design to implementation. However, we argue that this advantage is gained at the heavy cost of a rigidity of framework that makes it unsuitable for most complex and heterogeneous system. An alternative, 'multi-world' semantics will be suggested here.

4 One-world semantics for hierarchical systems

It may be simplest to consider an example of an autonomous vehicle.[2] We assume that a syntax with a facility for hierarchical descriptions is available to describe a vehicle as a three-level hierarchy:

- Level 1—odes describing vehicle, sensor, and actuator dynamics. Control law expression describes how sensor data are processed, and how actuator signals are evaluated;
- Level 2—vehicle movement described as concatenations of maneuver symbols: 'left turn', 'right turn', 'cruise', 'emergency stop', ... ;
- Level 3—vehicle route from A to B described as sequence of highway links traversed by vehicle.

At this point, 'ode', 'maneuver', 'route' and 'link traversal' are unrelated syntactic expressions, with no formal meaning. (Of course, the mental concepts we associate with these terms have meaning.) Let us suppose further that the syntax permits us to write expressions like:

route from A to B	$\rightarrow$	link 1 traversal followed by link 2 traversal ...
traversal of link i	$\rightarrow$	right turn followed by cruise maneuver ...
right turn	$\rightarrow$	feedback law for right turn
cruise	$\rightarrow$	feedback law for cruise control
other maneuvers	$\rightarrow$	...
feedback law for right turn	$\rightarrow$	ode for this feedback law and vehicle dynamics
other feedback laws	$\rightarrow$	...

Intuitively, the intention behind these expressions is this. We want to say that in order to take the route from A to B one should first travel over link 1, then link 2, and so on; in order to travel over link 1 one should first make a right turn, followed by a straight cruise, and so on; in order for the vehicle to make a right turn, it should be controlled by a particular feedback law, namely, the right-turn feedback law, and so on; and finally, the vehicle controlled by the right-turn feedback law is described by the right-turn ode, and so on. This intended meaning is imperative or procedural: it tells us how to execute a route from A to B by means of a sequence of link traversals; how to execute a link traversal by means of a sequence of maneuvers; how to execute a maneuver by a feedback law; and so on.

Thus with this intended meaning, a higher-level expression such as 'route from A to B' is compiled or translated or reduced into a complex lower-level expression involving 'link traversal'. Continuing

[2] For a formal treatment of distributed, hierarchical systems that uses one-world semantics see [7].

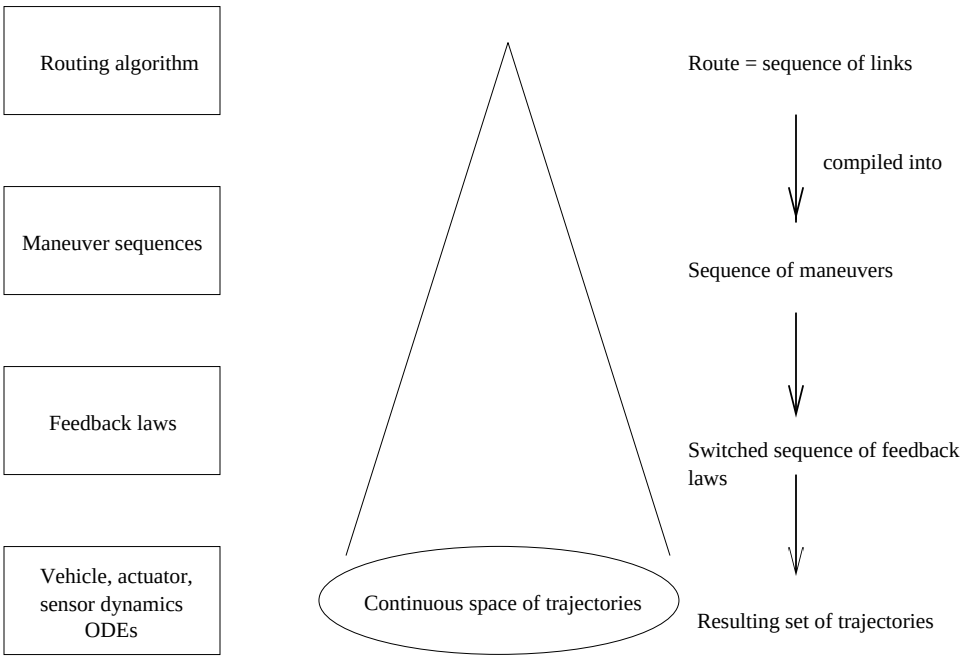

Figure 2: One-world semantics

in this way, we see that eventually any expression involving higher-level entities can be mechanically translated into the lowest-level entities which, in this example, are systems of switched odes. This compilation is depicted in Figure 2.

This compilation achieves several things in terms of the triad of relationships in Figure 1. First, it flattens out the syntactic hierarchy: higher-level expressions are translated into the lowest or ode-level expressions. Second, it seems to accord with our intuitive feeling expressed in the granularity-fidelity relationship that it is only the odes describing vehicle movement that correspond to the 'real' environment of vehicles-on-highways, whereas higher-level entities such as routes and maneuvers represent 'virtual' entities that don't refer to anything in the environment except as a short-hand expression that can be expanded into an ode expression.

More important than these two advantages is that the syntactic flattening or compilation can be used to argue that it is sufficient to provide a semantic interpretation only for the lowest level expressions. Why is this? Suppose we interpret odes in term of their solution trajectories. We can then interpret any high-level expression in a two-step compilation-interpretation process: we first compile the expression into a system of odes, and we then interpret the odes as their continuous trajectories. Let us call this methodological step the 'semantics flattening' move. Semantic flattening requires us to construct a semantics only for the lowest-level expressions. It thereby affords us a great simplification. What is the impact of this simplification on the two relationships of Figure 1 that involve semantics?

First consider the truth-claim relationship. Semantic flattening implies that any high-level truth-claim is proved in the lowest-level semantic domain. To return to our example, the truth-claim 'this route-finding algorithm is correct' is proved by compiling the algorithm first into a sequence of link traversals, etc., and eventually into a system of switched odes. One then interprets the original truth-claim as meaning that the solutions of this switched odes behave in such and such way, eg. all solutions of the odes starting at a point A end up in point B. If we have available a facility for automatically checking whether solutions of odes behave in the prescribed manner, then we have an automatic procedure for checking any truth-claim. This is a significant advantage. We will shortly discuss the disadvantages of semantic flattening.

Now consider the behavior-relevance relationship. The semantic flattening move implies that only those aspects of the behavior are relevant to our experience in the environment that can be associated with the lowest level. Thus, for example, if we wish to say that in our experience route 1 for going from A to B is shorter than route 2 for going from A to B, then we should somehow be able to say the same thing in terms of the solutions of the odes into which routes 1 and 2 are compiled.[3] This seems to be correct: the solutions of the route 1 odes should take a shorter time to go from A to B. We will see, however, that semantic flattening induces some undesirable features into the behavior-relevance relationship.

We now discuss the disadvantages of one-world semantics. Let us re-examine the implication that in order to prove the claim 'this route-finding algorithm is correct' we must show that the solutions of the odes into which the claim is compiled satisfy a certain property, namely the solutions go from A to B. We argue that this is unreasonable. Suppose that in our one-world semantic our vehicle can develop a fault that causes it not to move so that there is a solution to the ode that does not go from A to B. The claim 'this route-finding algorithm is correct' would then be false. On the other hand, if no fault were permitted, the claim would be true. Thus the validity of the claim will depend upon features of the world that are not expressible in the higher-level syntax. We can of course avoid this by reformulating the claim as 'this route-finding algorithm is correct, provided that there is no fault in the vehicle'.[4] But then we will have to introduce such reformulations of higher-level syntax and truth-claims every time we change a feature at any of the lower levels. This avoidance mechanism is deficient for reasons that are well-known in AI and associated with the problem of 'closed world' and 'non-monotonic reasoning'. So we won't pursue those deficiencies any further.

A bigger disadvantage is that in practice we don't design systems or validate truth-claims this way. Instead we use a multi-world semantics.

[3] There is a philosophical point here but this is not the place to elaborate upon it. Suffice it to say that it concerns whether the compilation of a higher-level expression into a lower-level expression is a meaning-preserving translation.

[4] Note that this reformulation may require extension of higher-level syntax to express 'no fault in vehicle'.

5 Multi-world semantics for hierarchical systems

In practice we design decision-making procedures at each level of the hierarchy and make truth-claims about those procedures by interpreting them in their own separate world. This is a kind of 'horizontal' semantics depicted in Figure 3, in contrast with the vertically collapsed one-world semantics of Figure 2.

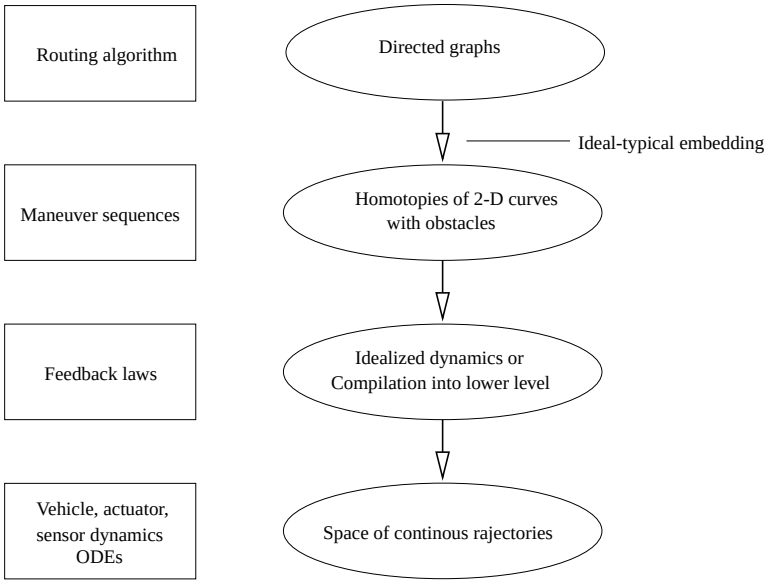

Figure 3: Multi-world semantics

As indicated, in multi-world semantics we interpret a claim at any level in a semantic domain tailored to that world. Thus, for example,

- Proof of 'route-finding algorithm is correct' is conducted in the semantics of directed graphs;
- Proof of 'correctness of algorithm for traversing link with sequence of maneuvers' is conducted in semantics of homotopy of curves in 2-D space with obstacles;
- Proof of 'stability of feedback law' is conducted in space of continuous trajectories.

Observe that this is how hierarchical decision procedures are indeed designed and validated in practice. We design a route-finding algorithm keeping in mind a model of directed graphs and we trust that a route-finding algorithm is correct if its correctness is proved in the semantic domain of directed graphs. The practice with the other lower-level claims above is similar.

This practice also has to do with the behavior-relevance relationship. Suppose our route-finding algorithm graph-semantics is instead collapsed into a one-world of faulty vehicles and it turns out that in that world the correctness of the algorithm is false. Even so we are unlikely to say that the

algorithm is incorrect, because we are unwilling to sacrifice its correctness in the *declarative* world of graphs in the face of its incorrectness in the *imperative* one-world of faulty vehicles.

This creates a conundrum. We wish to preserve our intuition about graphs and correct graph-theoretic routing algorithms. At the same time we must acknowledge that routes followed by faulted vehicles may not reach their destination. The difficulty is that preserving our intuition, which also and importantly, accords with practice, inevitably leads to disconnected multiple semantic worlds (one at each level of expression), and we don't know how to connect relate these worlds together.

This raises the question of finding a way to keep the advantages of the two kinds of semantics.

6 Suggestions

Here are three directions that may provide a useful response to the question.

- Ideal compilation—instead of syntactic translation followed by semantic interpretation at lower level, a higher-level expression is compiled into an idealized lower-level expression and then interpreted. Such ideal-prototypical compilation seems common in human reasoning and cognition.
- Invariants may be used to show that higher-level truth-claims now become conditional lower-level truth-claims, ie. higher-level truth-claims become necessary conditions. Note this is opposite of one-world semantics in which higher-level claims are sufficient conditions (cf abstraction in verification).
- Modal decomposition—suppose one-world interpretation leads to falsification of higher-level claims that are true in multi-world semantics. This can happen, say, because lower-level faults are not accounted for in higher-level descriptions. Then, multi-world semantics must be 'split' into multiple-frameworks, each dealing with identified faults.

The directions alluded to above suggest the decision structure of Figure 4. There are several scenarios, indexed by k. Each *scenario* comprises several levels, indexed by i, arranged in a hierarchy. At any instant, the decisions conform to a particular scenario. The scenario refers to the current 'estimate' of the system environment. Given that scenario, decisions at each level are specified according to the syntactical rules, and interpreted at that level. Behavior at each level is semantically compiled into ideal-typical behavior at a lower level.

In implementation, whether real or simulated or imagined, changes in the environment may cause large deviations of actual from ideal-typical behavior. This may trigger an alarm. The alarm may lead to a change in scenario and an intervention from the higher level.

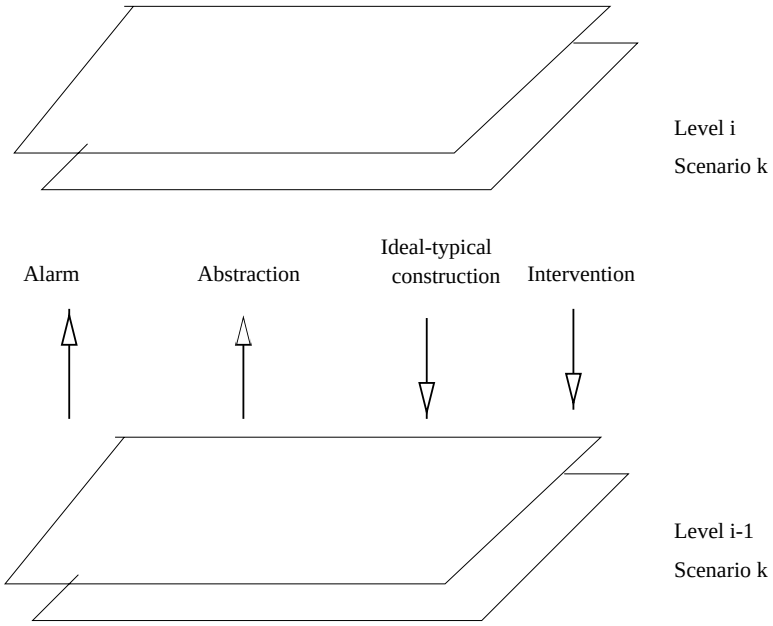

Figure 4: Hierarchical decision structure

Small changes in the environment may be handled by adaptive decisions that don't require scenario change. It is only the large changes for which 'parametric' adaption is ineffective that trigger scenario change. All this is intuitive. The challenge is to give a formal meaning to the structure. In closing, we note that this structure has proved useful in our work on automated highways [8, 5].

References

[1] H. Abelson and G.J. Sussman with J. Sussman. *Structure and interpretation of computer programs*. MIT Press, 1996.

[2] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.

[3] A. Deshpande, A. Gollu, and L. Semenzato. The SHIFT programming language and run-time system for dynamic networks of hybrid systems. *IEEE Transactions on Automatic Control*, 43: 584-587, 1998.

[4] A. Deshpande, A. Gollu, and P. Varaiya. A formalism and a programming language for dynamic networks of hybrid automata. *Hybrid Systems IV*, LNCS, Springer, 1997. To appear.

[5] D. N. Godbole, J. Lygeros, E. Singh, A. Deshpande, E. Lindsey, and S. Sastry. Design and Verification of Communication Protocols for Degraded Modes of Operation of AHS. In *Proceedings of the IEEE Conference on Decision and Control*, pages 427–432, 1995.

[6] A. Hsu, F. Eskafi, S. Sachs, and P. Varaiya. Protocol design for an automated highway system. *Discrete Event Dynamic Systems*, 2:183–206, 1993.

[7] M.P. Singh. *Multiagent Systems: A theoretical framework for Intentions, Know-How, and Communications*, volume LNAI 799. Springer-Verlag, 1994.

[8] P. Varaiya. Smart cars on smart roads: Problems of control. *IEEE Transactions on Automatic Control*, 38(2):195–207, February 1993.

[9] S. Yovine. SHIFT semantics. Unpublished Draft, March 1998.