A Unified Approach for Studying Discrete and Continuous Dynamical Systems

Oded Maler
VERIMAG
2, av. de Vignate
38610 Gières, France
maler@imag.fr
www-verimag.imag.fr/PEOPLE/Oded.Maler

Abstract

The goal of this paper is to present discrete transition systems and continuous dynamical systems in a uniform manner, stressing the fundamental differences as well as the commonalities between these two fundamental models. Such a framework seems to be a pre-requisite to any theory and methodology for hybrid systems. For both types of systems we introduce three models (a closed system, a system with one type of input and a system with two types of input) such that the problems associated with them correspond respectively to the tasks of *simulation*, *verification* and control *synthesis*. We will discuss some of the computational problems associated with building control CAD tools that carry these tasks.

1 Introduction

The primary object of computer science and of discrete event systems (DES) is the *discrete transition system* (automaton). In control theory this role is played by the *continuous dynamical system*. Hybrid systems combine these two models and in order to develop a theory to support them, it is useful to step back and attempt to unify these two models, and reformulate problems, results and algorithms in a similar fashion. This is what this paper attempts to do, without presenting any new results. We believe that every scientific domain can benefit from re-examination of its underlying assumptions, its models and its (often implicit) rules of the game, especially when the ultimate goal is to build an inter-disciplinary research community.

We sketch three models (closed systems, systems with one type of input and systems with two types of inputs) and show how the corresponding activities of simulation, verification and control synthesis are treated in the continuous and discrete cases. Due to time and space limitation, this manuscript is far from being complete and many important issues are only mentioned briefly.

2 Common Features

We will use T to denote the *time domain*. For discrete transition systems, T is the set $\mathbb{N}$ of natural numbers. More often than not, it is the *order* relation on $\mathbb{N}$ which is important and not the *metric*: the "real" time that has elapsed between two consecutive events can be arbitrarily small or large.[1] For continuous systems, we need to make a distinction between a *theoretical* model, where T will be the set $\mathbb{R}_+$ of non-negative reals (as defined in calculus textbooks) and an *effective* model, used in computations, where T is something like $\{n\Delta : n \in \mathbb{N}\}$ for some rational Δ.

We will use three data domains: X will be the *state-space* of the system under consideration. At the first level of modeling (*simulation*) we will consider closed systems such that given an initial state $x_0 \in X$, the state of the system is *determined* for every $t \in T$. At the second level (*verification*), we add an input domain U, affecting the dynamics of the system.[2] Finally, at the third level of modeling (*synthesis*), we consider two input domains, U and V, having different interpretations. One stands for the controller's actions while the other models uncontrolled disturbances — a two-person game situation.[3] In this model we are interested in finding a control law which guarantees that the systems behaves properly in the presence of *any* admissible disturbance.

For transition systems, X, U and V are usually finite (or at most, countable) sets. They will be rather *amorphous* sets admitting neither order relations, nor operations such as addition or multiplication.[4] For contin-

[1] In fact, in models such as *Timed Automata* metric time is re-introduced.

[2] This can be interpreted in two ways: either U represents uncontrolled disturbances, in that case answering questions concerning *all* possible input patterns of U has, indeed, a verification flavor. We can view U as the output of a controller (computed as function of the state) and thus, in the absence of disturbances, equivalent to a pre-computed control.

[3] This can be generalized to an arbitrary number of players, which are not necessarily antagonistic. Such a model can be a useful for distributed systems.

[4] We ignore, for the moment, the problem of the representation of X by a product of smaller domains, which has *a lot*

uous systems, the theoretical domains are reasonable subsets of $\mathbb{R}^n$ (or other differentiable manifolds) while for doing effective calculations on a computer we will assume them to be "sufficiently dense" finite subsets of the rationals, without going too deep into the foundations of numerical analysis.

A *behavior* (or an input) of a system is a function of the form $\xi : T \to X$ (or a partial function defined over some interval $[0,t] \subseteq T$). These are called *sequences*[5] for discrete T, and *signals*[6] for continuous T. The set of all such behaviors is denoted by X^* (we borrow the computer science notation). In the continuous case X^* is usually restricted to have some nice properties (measurable, continuous, smooth, Lipschitz). A length-preserving[7] function $f : X^* \to Y^*$ is a causal transduction if for every t the value of $f(\xi)$ at t depends only on the values of ξ in the interval $[0,t)$. Any function $f : X \to Y$ admits a natural pointwise extension $f : X^* \to Y^*$.

3 Model I

3.1 Discrete Systems

Definition 1 (System I-D) *A transition system is $S = (X, \delta)$ where X is a finite set and $\delta : X \to X$ is the transition function.*

Given an initial state $x_0 \in X$, the behavior of the system is a sequence $\xi : \mathbb{N} \to X$ (or equivalently $\xi \in X^*$) such that $\xi[0] = x_0$ and for every i,

$$\xi[i+1] = \delta(\xi[i]).$$

A useful abuse of notation is:

$$x_{i+1} = \delta(x_i).$$

Yet another way to characterize the behavior of S is to define two operators (functions) from X^* to itself. One is the pointwise extension of δ and the other is the delay (shift) operation z,

$$z(\xi)[i] = \xi[i-1].$$

The behavior of the system is the fixed-point of $\delta \circ z$, that is, it satisfies

$$\xi = \delta(z(\xi)).$$

3.2 Continuous Systems

Definition 2 (System I-C) *A differential system is $S = (X, f)$ where X is $\mathbb{R}^n$ and $f : X \to X$ is a continuous function (vector field).*

Given $x_0 \in X$, a behavior of the system is a signal $\xi : \mathbb{R}_+ \to X$ satisfying $\xi[0] = x_0$ and for every t,

$$d\xi[t]/dt = f(\xi[t]).$$

People also say[8]

$$\dot{x} = f(x).$$

This can be phrased also as:

$$\xi[t] = x_0 + \int_0^t f(\xi[\tau])d\tau.$$

Again, the behavior can be seen as a fixed point of a signal operator, composed from the pointwise extension of f and the integrator I, defined for every signal ξ as

$$I(\xi)[t] = x_0 + \int_0^t f(\xi[\tau])d\tau.$$

The solution[9] is the fixed-point of $I \circ f$,

$$\xi = I(f(\xi)).$$

Note the difference from the discrete case: here the initial value x_0 is a basis for the ongoing *summation*, while in the discrete case no "inertia" is involved.[10]

It is worth mentioning that both discrete and continuous systems satisfy the *semi-group property*, that is, if ξ is the behavior of the system starting at x_0 and ξ' is the behavior starting at $\xi[t_1]$ then $\xi[t_1 + t_2] = \xi'[t_2]$.

3.3 Approximated Continuous System (Sketch)

There are two basic steps leading from the ideal mathematical continuous objects to their effective realizations. At the first level, one still pretends that X is indeed $\mathbb{R}^n$, but admits T to be discretized time (with a fixed step). This is the level which corresponds to *discrete-time dynamical systems*. When looking further into computational realizations, X is discretized as well, and a whole science of error bounds is needed.

3.4 Problems and Solutions

Given a description of a dynamical system, the most natural thing to ask is how it will behave starting from some initial state. In many cases, we are particularly interested in avoiding a certain set of "bad" states.

Definition 3 (Basic Reachability Problem) *The basic reachability problem for a dynamical system S is: given x_0 and a subset P of X, does there exist a time t such that the behavior of S starting at x_0 satisfies $\xi[t] \in P$.*

of significance when dealing with complexity, compositionality, distributed control etc.

[5] Also called *runs*, *executions*, *words*, *strings*, *traces*.

[6] Also known as *trajectories*, *orbits*, *solutions to the Cauchy problem*.

[7] I.e. the domain of definitions of the input and output signals coincide.

[8] In general, theoretical computer science, due to the influence of mathematical logic, is much more concerned with the distinction between syntax (symbols, formulae) and semantics (what the syntactic objects denote).

[9] When exists and is unique and all that.

[10] What this vague statement is supposed to say is that in continuous systems, the "next-state" function $\xi[t+\Delta t] = g(\xi[t], \Delta t)$ can be decomposed naturally into $g(\xi[t], \Delta t) = \xi[t] + h(\xi[t], \Delta t)$ while in the discrete case, such a decomposition usually makes no sense.

A related problem is to find *all* the states reachable from x_0.

Remark [Properties]: Reachability is only one type of what is called in verification a *property*. A property is nothing but an expression in some formalism which denotes some subset of X^*. For example, in linear time temporal logic, the formula $\Box P$ denotes the set of sequences $\{\xi : \forall t \in T, \xi[t] \in P\}$. Likewise, the formula $\Box\Diamond P$ denotes the set of sequences such that $\xi[t] \in P$ for *infinitely many* t's. Properties can, of course, be viewed as functions from X^* to $\{0,1\}$ which makes them closer to *cost* and *value* functions used in optimal control, but their all-or-nothing nature calls sometimes for a different treatment.

Remark [Deductive vs. Algorithmic]: In discrete systems verification, one distinguishes between two basic approaches to solving reachability problems. Within the *deductive* or theorem-proving approach, reachability properties are inferred formally from axioms and rules concerning the dynamics of the system. The main disadvantage of this approach from the CAD point of view is that it is not *fully-automatic*, that is, one does not feed the computer with the description of the system, pushes a button and obtains the result. Even with the help of an automatic theorem prover, an active participation of a human user who understands the dynamics of the system in question is required. The analog of this approach in continuous systems would be, for example, proving a reachability property using a user-supplied Lyapunov function. In the rest of this paper we restrict the discussion to the alternative *algorithmic* approach, in which the computer is expected to solve the problem without any human intervention.

For finite-state discrete systems every behavior is an ultimately-periodic[11] sequence of states. Hence a simple algorithm can solve the reachability problem: start with $\xi[0] = x_0$ and calculate $\xi[i+1] = \delta(\xi[i])$ until either $\xi[i] \in P$ (answer is "yes") or $\xi[i] = \xi[j]$ for some $j < i$ and we have reached a cycle without visiting P (answer is "no"). You can either memorize the visited states or just count the number of steps: if P has not been reached in $|X - P|$ steps, it will never be. The ultimately-periodic state sequence uv^ω, which can be extracted from the algorithm, gives truth assignments to every other temporal property on X^*. For example the temporal logic property $\Box\Diamond P$ (infinitely-often P) is true if some $x \in P$ appears in the period v.

Remark [Backward vs. Forward]: The abovementioned algorithm solves the reachability problem by *forward* simulation. There is a similar method using *backward* simulation from P which can be used to determine all the states from which the system goes to P (a kind of "domain of attraction"). Since going backwards may introduce non-determinism, we will discuss it in the next section. Note that unlike systems defined by differential equations, discrete transition systems are rarely reverse-deterministic.

Finiteness plays an important role in this setting: the transition function, the set P, and the set of reachable states accumulated during the simulation can all be enumerated explicitly and be stored in finite data-structures. Finiteness also guarantees the ultimate-periodicity of the trajectory.

By relaxing the finiteness condition and allowing a countable state-space such as $\mathbb{N}^n$, and an effectively-computable δ (i.e. a procedure for calculating the next-state), the reachability problem becomes *undecidable*. This notion is a bit alien to control theorists so it is worth elaboration: by saying that the reachability problem for discrete infinite-state systems is undecidable, we mean that *there is no general algorithm* that can take an effective description of *any* discrete dynamical system with unbounded integer variables (e.g. a program or a recurrence equation over $\mathbb{N}^n$), and solve the reachability problem on it. All that you can do is to simulate forward until you reach P ("yes") or make a cycle ("no"), but none of these is guaranteed to happen.[12] This notion allows theoretical computer scientists to publish *negative* results concerning the provable *inability* to produce certain algorithms.

Other problems that come with infinitude are the representation of the set P and of the accumulated trajectory. Typical representations of P would be combinations of linear inequalities or even worse types of inequalities. Checking the membership of $\xi[t]$ in P becomes another computational issue.

How much of this carries over from discrete to continuous systems? Continuous systems allow two types of infinitude, unbounded (which seems roughly like discrete infinity) and bounded (compact if you want to sound more mathematical). If hypothetically our computers had infinitesimal computation power (i.e. the capability to perform continuous integration) we could simulate trajectories forward but still suffer from the problem of infinite state-space: the trajectories are not necessarily periodic. Hence, we would have only a semi-decision procedure: if the simulation reaches P, the answer is positive, otherwise, unless we have detected a cycle, we can never know. So even in this ideal setting, the reachability problem for arbitrary continuous systems can be at best semi-decidable (for certain sub-classes of continuous systems, the problems can be solved by analytical methods).

But our machines do not have such a computational power and the simulation is performed in *discrete time* on a *discretized space*. It seems that by adding epsilons

[11] A sequence is ultimately periodic if there exist k, l such that for every n, $\xi[k + nl] = \xi[k + (n+1)l]$. In formal language theory such sequences are written as uv^ω where u and v are finite sequences, the first denoting the *prefix* and the second denoting the *period* of ξ.

[12] Not *all* infinite-state systems have undecidable reachability problems – some sub-classes, such as systems with one integer variable, admit reachability algorithms which always terminate.

in the right places, e.g. asking whether $|P, \xi[t]| \leq \varepsilon$ instead of $\xi[t] \in P$, numerical simulations can solve the reachability problem for this type of systems. A more serious discussion of numerical aspects[13] will eventually appear in an expanded version of this paper.

4 Model II

4.1 Discrete Systems

Definition 4 (System II-D) *A one-input transition system is $S = (X, U, \delta)$ where X and U are finite sets and $\delta : X \times U \to X$ is the transition function.*

The system evolution is now influenced also by an input. A behavior of S *given* some $\psi \in U^*$ is a sequence ξ such that[14] for every i,

$$\xi[i+1] = \delta(\xi[i], \psi[i]).$$

An arbitrary behavior of the systems is a sequence ξ such that for every i,

$$\exists u \in U : \xi[i+1] = \delta(\xi[i], u)$$

and this is equivalent to projecting away U from the transition function, and obtaining a non-deterministic transition function (also known as *transition relation*) $\hat{\delta} : X \to 2^X$ with

$$\hat{\delta}(x) = \{x' : \exists u \in U, \delta(x, u) = x'\}.$$

This is one of the main reasons for computer science non-determinism: when we do not know the values of some variables we consider *all* the possible transitions which these values might induce.[15] Note that this non-determinism is completely *qualitative* and it does not assign any probabilities to subsets of U^* or X^*. All it does is to specify the sets of all possible behaviors.

Yet another way to characterize behaviors of a type II system is as a sequential function from U^* to X^* or as a subset of $(U \times X)^*$ consisting of all the pairs (ψ, ξ) satisfying

$$(\psi, \xi) = (\psi, \delta(z(\xi), \psi)),$$

i.e. a fixed point of an operator on $(U \times X)^*$.

Remark [Admissible Inputs]: Here we made an implicit assumption that all elements of U^* are admissible. Sometimes only a subset of U^* can be considered as input to the system, e.g. sequences where no element of U repeats more than 3 consecutive times. Such restrictions can be captured by a model of automata with both input and output. We assume (X, U, δ) as before and connect it to another system whose *output* ranges over U and which generates only the admissible inputs. When both are composed together, U becomes internal, and the product system is non-deterministic. If all U^* is admissible, the input generator is simply the one-state trivial automaton.

The main important observation to remember is:

Open Deterministic System
=
(via input projection)
Closed Non-Deterministic System

4.2 Continuous Systems

Definition 5 (System II-C) *A differential system is (X, U, f) where X is $\mathbb{R}^n$ and $f : X \times U \to X$ is a continuous function.*

Given $\psi \in U^*$ the behavior of the system is a signal $\xi : \mathbb{R}_+ \to X$ satisfying for every t,

$$d\xi[t]/dt = f(\xi[t], \psi[t])$$

or

$$\dot{x} = f(x, u)$$

or

$$\xi[t] = x_0 + \int_0^t f(\xi[\tau], \psi(\tau)) d\tau.$$

The set of all possible behaviors of the system is obtained by projecting away the input signals, i.e. all ξ such that there exists some admissible $\psi \in U^*$ satisfying the above equation. As we did for the discrete δ, we can get rid of the input at the equation level by defining $F : X \to 2^X$ as

$$F(x) = \bigcup_{u \in U} f(x, u).$$

The set of behaviors of S can be obtained as solutions of the *differential inclusion*

$$\dot{x} \in F(x),$$

that is, the set of behaviors ξ such that for every t,

$$d\xi[t]/dt \in F(\xi[t]).$$

For completeness sake we also write the set of behaviors as a (functional) relation on $(U \times X)^*$ which is the solution fixed-point equation

$$(\psi, \xi) = (\psi, I(f(\xi, \psi))$$

where I is the integral operator. Solutions of differential inclusions which are "tubes" of trajectories are

[13] For example, what is the relation between the existence of a cycle in the ideal system and the detection of a cycle in the numerical simulation.

[14] From now on we omit explicit reference to the initial state x_0 and the fact that $\xi[0] = x_0$.

[15] Other uses of non-determinism where the computing devices "guesses" some values during the computation, are very common in computability and complexity theory, but seem irrelevant to the current discussion.

not the favorite object for most dynamicists. There are serious problems concerning the interpretation of solutions, existence and uniqueness which we will not touch here. However, for computer scientists such objects are very natural (or at least no more bizarre than single continuous trajectories).

4.3 Approximated Continuous System [Sketch]
Due to the discretization of T, U^* is restricted to *piecewise-constant* signals. Further discretization of U can make the techniques used for discrete systems, as described in the next section, applicable to continuous systems.

4.4 Problems and Solutions
As in the case of closed systems, we are interested in the possible behaviors of the system for any admissible input. There are two ways to interpret the input U from a control point of view. It can either stand for "our" choice of a control action at a given state, or a disturbance (an uncontrolled action of the external environment). The chosen interpretation and the types of questions which are asked determine whether the quantification over the possible behavior is existential or universal.

Let H be some property of trajectories, and let $L(S,\psi)$ denote the behavior of a type II system S given an input $\psi \in U^*$. Interpreting U as being under our control we can phrase the question *"Is there some ψ which steers the system such that it satisfies H?"* as:

$$\exists \psi \; L(S,\psi) \in H \tag{1}$$

and its negation as:

$$\forall \psi \; L(S,\psi) \notin H \tag{2}$$

On the other hand if U stands for the adversary, we can ask whether the system will satisfy H for *any* admissible input/disturbance:

$$\forall \psi \; L(S,\psi) \in H \tag{3}$$

and its negation as:

$$\exists \psi \; L(S,\psi) \notin H \tag{4}$$

Note that if we project away U, and let $L(S)$ denote the set of all behavior of the resulting non-deterministic system, question (1) becomes

$$L(S) \cap H \neq \emptyset \tag{5}$$

and question (3) becomes

$$L(S) \subseteq H \tag{6}$$

Question (6) is essentially what verification is all about, although it is not easy for non-natives to discover this fact under the formalistic make-up, as it is hard to get to the essence of control by browsing the CDC proceedings. For historical reasons, solving problems such as (6) is called *model-checking*.

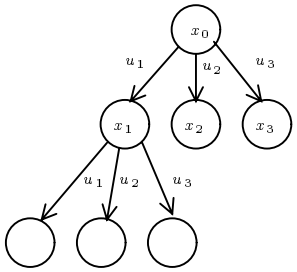

Figure 1: An initial part of the execution tree of a type II system.

Whenever the answer to questions (1) or (4) is positive we would like a constructive answer, i.e. an input $\psi \in U^*$ which induces the behavior in question. In the former case ψ can be a pre-computed open-loop control sequence (remember that the system is deterministic once ψ is determined). In the latter case it will be an example of an external input which causes the system to violate H. Whether or not we are interested in such "witnesses" will influence our decision to hide the input U and treat the system as a non-deterministic one.

Before elaborating on the type of properties we are interested in, let us reflect a little bit on the structure of the set of behaviors of a type II system. This is the natural point of view for the the discrete case, but with some dose of infinitesimal imagination one may view the continuous case similarly. The set of trajectories starting from x_0, can be organized as a *tree*, with x_0 as its root and where each node has successors corresponding to possible choices of U. Each behavior is a sequence of nodes along one branch of the tree, and the input is the sequence of labels on the edges (see figure 1).

Checking a property of the set of trajectories amounts to searching the execution tree (which in the finite-state case folds into a graph). If the property H we are interested in is a simple reachability property ("safety"), i.e. $\xi \in H$ iff $\xi[t] \in P$ for some t, then the way we search the tree is not important: H is satisfied depending on the existence of some P-node *anywhere* in the tree. On the other hand, if we are interested in more complicated properties, such as beahviors where every occurrence of x is followed by a later occurrence of x', the order of traversal is important because the existence of infinitely many x and x' nodes on the tree does not imply the existence of a branch satisfying the property. In any case, discrete finite-state verification can be reduced to various search problems on finite graphs, where many techniques such as depth-first, breadth-first or heuristic serach exist.

Theoretically, the verification problem for discrete event systems (with respect to all interesting classes of

properties) is solved, and all that remains to be done is to improve performance and to extend the methodology toward systems with countable state-spaces. For continuous systems, there are various works on differential inclusions, but their nature is much less "effective". It seems that the common engineering practice, when no analytic method works, is to simulate with many instances of inputs signals, hoping that the obtained set of behaviors is a representative sample of the set of all behaviors.

Remark ["Symbolic" Verification]:

In discrete finite-state systems, one also distinguishes between *enumerative* and *symbolic* methods for doing verification. In the former, the search is performed on an explicit representation of the transition graph, while in the latter the system and the set of reachable states are encoded using some formalism, such as Boolean formulae over state variables.[16] The calculation of the reachable set is usually performed breadth-first by doing *syntactic* operations on these formulae. In many cases, symbolic techniques allow to treat systems with a number of states which is otherwise prohibitive. For infinite-state systems (and uncountable-state systems in particular) exhaustive enumeration is not an option. A differential equation is, among other things, a symbolic description of an uncountable transition relation, and a closed-form solution to the initial-value problem is a symbolic description of the set of reachable states, which can be checked for intersection with P using algebraic techniques.

5 Model III [Sketch]

5.1 Discrete Systems

Definition 6 (System III-D) *A two-input transition system is $S = (X, U, V, \delta)$ where X and U are finite sets and $\delta : X \times U \times V \to X$ is the transition function.*

The behavior of the systems in the presence of two inputs, $\psi \in U^*$ and $\eta \in V^*$, $L(S, \psi, \eta)$ can be characterized as before. The main novelty here is in the different interpretation we give to the two inputs. This model can be viewed as a game between a controller U and the external disturbances V. The overall potential behavior of a type III system can be viewed as a game tree which is the unfolding of an *alternating* And-Or automaton. After input projection the transition structure of the automaton becomes a function $\bar{\delta}$ from X to the distributive lattice generated by X, i.e. something of the form:

$$\bar{\delta}(x) = \delta(x, u_1, v_1) \wedge \delta(x, u_1, v_2) \vee \delta(x, u_2, v_1) \wedge \delta(x, u_1, v_1).$$

For such systems we are looking for a function $C : X^* \to U$ (*strategy, feed-back law*) which tells the controller which action to apply at a given stage of the game such that whatever the adversary does, the resulting behavior satisfies some property H. Pseudo-formally[17] we ask whether for *every* sequence $\eta \in V^*$ there *exists* a sequences $\psi \in U^*$, which is calculated in a causal manner, such that $L(S, \psi, \eta) \in H$. This is essentially the controller synthesis problem for discrete-event systems which can be solved by various algorithms, one of which we illustrate below.

Let H be the property of staying within $P \subseteq X$. We define the operator $\pi : 2^X \to 2^X$ as

$$\pi(Q) = \{x : \exists u \in U \ \ \forall v \in V \ \ \delta(x, u, v) \in Q\}.$$

In other words $x \in \pi(Q)$ iff from x the controller, by properly chosing u, can force the game into Q. The states from which the controller can force the game to stay at P *forever* are called the *winning* states and they are calculated iteratively by letting $P_0 = P$ and

$$P_{i+1} = P_i \cap \pi(P_i).$$

This is a sort of backward reachability calculation which is guranteed to converge for finite-state systems. The strategy for the winning states can be extracted during the iteration or after it terminates.

Many question are not discussed in this version of the paper. They include maximality of the controller (in the sense of being least-restrictive), partial observability of the state and the distinction between memoryless strategies (the choice of u depends only on the current state) and strategies which memorize some of the history.

5.2 Continuous Systems

The notion of a *differential game* involves a system defined by

$$\dot{x} = f(x, u, v)$$

and one is interested in finding a continuous control law $C : X^* \to U$ such that all trajectories satisfy a property. In many aspects these games are very similar to the discrete ones, except for the fact that they evoke some mathematical problems concerning the existence of solutions. Apart from the explicit study of differential games, many branches of control (stable designs, robust control, stochatic control) seem to study variations (mostly restrictions but some extensions) of this model.

Acknowledgments: Comments made by E. Asarin, A. Kurzhanski, J. Lygeros, G. Pappas, S. Sastry and P. Varaiya on drafts of this paper improved its political and mathematical correctness. The lack of references is due to space constraints and is balanced by the lack of self-references. This work was supported by the European Community Esprit-LTR Project 26270 VHS at VERIMAG, the US ARO under Grant DAAH-04-96-1-0341 and DARPA under grant F33615-98-C-3614 at Berkeley.

[16] These formulae are represented and manipulated efficiently using data-structures such as Binary Decision Diagrams (BDDs).

[17] The fact that ψ is computed as a feed-back from the history of the game, makes formalization cumbersome.