Guest Editorial: Verification of Hybrid Systems

Oded Maler
CNRS-Verimag
Centre Equation
2, av. de Vignate
38610 Gières
France
Oded.Maler@imag.fr

1 Introduction

This special issue of *The European Journal of Control* is dedicated to some of the results of the European Community Esprit-LTR Project 26270 VHS (Verification of Hybrid Systems). It consists of 5 papers dealing with various aspects of one of the case studies treated in the project, the *Experimental Batch Plant* at the university of Dortmund. Before presenting the papers themselves, it is worthwhile to say something about hybrid systems in general and the VHS project in particular.

2 Hybrid Systems in General

This section is concerned with some general questions about hybrid systems models, not necessarily related to the VHS project.

2.1 Yet Another Fashionable Buzzword?

Over the last decade certain interest has been manifested in *Hybrid Dynamical Systems* by the Control and Computer Science communities, as attested by numerous workshops and special issues of scientific journals. Since the term "hybrid" can be used to denote any mixture of two different things, no wonder it has been used in diverse contexts. To avoid misunderstanding let us fix the meaning of the term to those dynamical systems exhibiting both *continuous evolution* and *discrete transitions*. In retrospect, one can identify two major reasons for this wave of interest.

The first, somewhat cynical, reason is related to the dynamics of academic life. Modern science is populated with people who are very competent in certain bodies of knowledge and techniques. In order to survive (both in terms of institutional evaluation and of self-esteem) they need to find new applications of their know-how and produce new results. For someone who knows how to do X, especially when X is part of a saturated domain of research where all the easy problems have already been solved, "hybrid" problems may provide new opportunities for results of type X', X'', etc. This type of activity may sometime lead to

creation of new fundamental theories but, more often than not, its impact beyond a temporary solution of the problems of the individual or the community is limited.

The second reason is more profound. Science is concerned with finding useful abstract models for natural (and more recently, technological) phenomena. The usefulness of a class of models is measured by its correspondence to reality, as manifested by the predictive power of the models and by the help they provide in designing systems. Successful models suggest a good tradeoff between the faithfulness of the model to reality and its complexity.[1] It is in this context that the hybrid approach for modeling control systems should be analyzed. My claim is that hybrid models have the potential of offering relatively clean modeling solutions for phenomena for which the corresponding classical models are inadequate or do not provide a good tradeoff between real-world relevance and model complexity.

2.2 Limitations of Continuous Models

The traditional formalisms for describing system dynamics for the purpose of controller synthesis are based on continuous dynamical systems, one of the most notable success stories of mathematical modeling. Continuous models were developed initially for predicting the behavior of uncontrolled systems such as the solar system and were adapted later for systems to which inputs can be injected such as steam engines and missiles. Such systems are specified by differential or difference equations, describing the evolution of the state-variables under the influence of input signals (disturbances and control signals). These models are derived partly by application of commonly-accepted physical laws such as those of mechanics, electricity or thermodynamics and partly by a procedure of system identification in which data obtained from experiments with the plant are used to estimate system parameters and to compare the estimated model with the data. Since these are models of real physical phenomena, it is well understood that they are nothing but useful approximations of a much more complicated dynamics and there is a trade-off between the faithfulness of a model and the feasibility of its analysis from both mathematical and computational standpoint.

There is a variety of methods for synthesizing controllers for systems defined using continuous dynamic models. Some of them are widely and successfully used in practice and some are of a purely theoretic interest. A crucial point for most of these methods is that the systems in question live in some rich mathematical domain such as the Euclidean space or, more generally a differentiable manifold, over the real number field where one can do subtraction and division and that the dynamics is described using "nice" functions, i.e. functions constructed from arithmetic, algebraic and trigonometric building blocks without logical elements. I will mention some of the reasons why these traditional models are insufficient for describing real systems. It should be noted that the problems mentioned below are well-known in the control community, and various theoretical and practical approaches for treating each of them were developed without resorting to explicit hybrid models.

1. The dynamics of many physical components of plants and controllers cannot be modeled in a useful manner using purely-continuous models. The behaviors of valves and switches are best modeled as discrete transitions. Moreover, even continuous sensors and actuators have saturation effects beyond certain values.

[1] This primary feature of scientific models, although it is the official raison d'être of the whole enterprise, is not the only one influencing their success. Other factors that intervene include the cognitive limitations and aesthetic preferences of researchers, the prevailing world views of the time and sociological factors including personal inertia, ideology and authority.

2. High-level "intelligent" control might involve requirements that are sequential by nature and cannot be expressed naturally in terms of continuous trajectories. Movement in physical space containing "objects" and "places" is inherently discrete and exhibits phenomena such as collision.

3. Even when there is a natural continuous model for the system dynamics, it is often the case that this model is highly non-linear. Many methods are based on a linear approximation which is valid only in some region of the state space. When the system leaves this region a different linear model should be used (and consequently a different derived controller).

4. Many control systems are part of larger systems and interact with entities other than continuous sensors, e.g. human operators or computers. Such entities may send discrete commands to the controller that will activate or suspend its execution or force it to switch to another mode of operation.

5. Theoretically, there are plants which cannot meet specific performance requirements by continuous controllers but can do so by discontinuous controllers.

Most of these facts are known to everybody who actually builds control systems. The problems that they pose to the continuous control paradigm are solved in a pragmatic manner. The essential continuous dynamics is modeled as faithfully as possible in important subsets of the state space which are viewed as different "modes" of operation. Control laws are synthesized for each of these modes and then "glued" together, this way or another, without regarding the transitions between them as part of the "official" dynamics of the system. In other words, the formal notion of a dynamical system is reserved only for the continuous modes, and all phenomena outside the scope of the continuous framework are treated as extra-modelic phenomena.[2] But behaviors consisting of discrete transitions in a state space without much structure are the bread and butter of computer scientists who have an appropriate dynamical system model for them, the automaton.

2.3 Discrete Systems and Verification

That part of Computer Science which is concerned with the formal approach to the design of "reactive" systems[3] has goals similar to Control, namely to design systems that interact with an external environment according to some requirements. A typical reactive system would be a mechanism which controls the access of clients to some shared resources. The interaction of this controller with its environment is not via measurements of physical magnitudes, but rather via *messages* and *events* such as "request" or "release" which are entities whose essence is logical and independent of the particular physical medium in which they are realized. Similarly, the state variables of such systems do not represent physical magnitudes but rather non-numerical values such as "ready", "waiting", "idle", etc. Such a state space is usually devoid of any useful metric.

[2] This statement contains some over-generalizations, but I think it reflects the spirit of mainstream control.

[3] The term "reactive" should be understood in the context of computer science and is intended to distinguish the systems under consideration from other computer programs that do not interact with the outside world between reading their input and producing their output. All control systems are reactive in this sense, and those studied under the title of Discrete Event Dynamical Systems are similar to those treated in computer science.

The dynamics of such systems is a state-transition dynamics, that is, it defines for each state and for each input event what is the *next* state, where "next" means the next *logical instance* of the system evolution. It is worth mentioning that this logical time scale, although it is order-isomorphic to $(\mathbb{N}, <)$, does not necessarily correspond to a fixed-step discretization of $\mathbb{R}$. When the state space is small, the dynamics can be written explicitly in a table. Larger systems are described using a combination of two methods. One is *composition* where interacting sub-systems are described separately, and the global system is their product whose size can be exponential in the number of components. The other complementary approach is to use an *implicit* (symbolic) description using, for example, a programming formalisms with *if..then..else* statements[4] or a network of logical functions and delay elements. The questions asked about the behaviors of such systems are similar to those asked in Control but the nature of the state space dictates quite different methods for answering them.

Although the state space of a discrete system is much smaller than that of a continuous one (it is finite or, at most, countable), the fact that discrete systems are defined on impoverished mathematical domains without inverse makes their analysis and synthesis harder. To illustrate this fact consider a system of equations $A\mathbf{x} = \mathbf{b}$ over the reals. Such a system admits a simple (and polynomial) solution procedure such as Gaussian elimination which takes advantage of division and subtraction. On the other hand, finding whether there is a Boolean vector satisfying a formula of propositional logic, written in conjunctive normal form, is an NP-hard problem for which all known solution algorithms might end up exploring all possible vectors. Likewise, if we consider dynamical systems, a lot can be said about the trajectories of a linear dynamical systems $\dot{\mathbf{x}} = A\mathbf{x}$ just by inspecting the structure of the A matrix. In discrete systems, there is usually no such "holistic" way to capture the global behavior of the system from its concise description and, consequently, following all the possible trajectories is sometimes the only way to say something meaningful about the system behavior.

These analytic difficulties, combined with the relative easiness of using trial-and-error design methods in software, delayed the introduction of mathematically-based methods for discrete systems design. However, with the growth in systems complexity, the increase in the consequences of their malfunctioning and the progress in research, a *verification* methodology for discrete systems emerged, comprising of the following ingredients:

1. Modeling formalisms for describing systems, based on interacting automata and other variants of transition systems.

2. Formalisms for specifying system requirements in terms of acceptable and unacceptable behaviors. These can be automata, *regular expressions* or formulae in *temporal logic*, a logic tailored for specifying sequences of states and events.

3. Methods to verify that a controller, composed with its environment, generates only acceptable behaviors. These methods are either algorithmic (explore the paths in the transition graph that correspond to possible behaviors induced by admissible inputs) or deductive (try to prove, under user guidance, some claims about all system behaviors).[5]

[4] It is a pity that the general public, including many programmers, is unaware of the fact that hardware and software systems have an established underlying model of a dynamical system. This can be attributed to two factors: one is the dominance of the programming language view of software and the other is the emphasis in computer science curriculum on automata as *acceptors* of strings and not as *generators* or *transducers*.

[5] We note here that in discrete verification it is more common to look at the controller *after* it has been synthesized, but there is no inherent reason for not doing automatic synthesis in this framework.

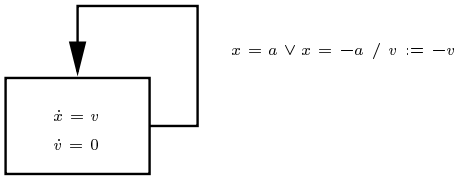

Figure 1: A hybrid automaton for collision.

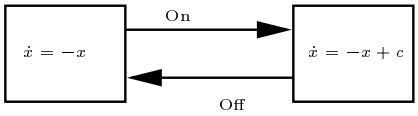

Figure 2: A heating system with On/Off inputs.

From the Computer Science point of view, hybrid systems research is an attempt to export this methodology from purely-discrete systems to discrete systems that interact with a physical environment, a situation found, for example, in digital control systems.[6] One of the most popular hybrid models is the *hybrid automaton*, a combination of an automaton and a set of differential equations.

2.4 Hybrid Automata

Rather than bombarding the reader with definitions of n-tuples, I will illustrate hybrid automata informally using some examples, giving priority to simplicity over pedantic precision. The hybrid automaton of Figure 1 models the frictionless movement of a particle in a bounded interval $[-a, a]$ of $\mathbb{R}$, subject to elastic collisions at the endpoints of the interval. As we can see, whenever the particle location x reaches one of the endpoints, a transition that changes the sign of the velocity is taken.

The automaton of Figure 2 models a heating system subject to external On/Off commands. If these commands are issued by a thermostat according to some temperature thresholds, we obtain the automaton of Figure 3. The behaviors of hybrid automata are piecewise-continuous: they consist of an alternation between a continuous phase, where the discrete state remains fixed while the continuous state variables evolve according the corresponding differential equation, and discrete transitions between states which are assumed to consume no time (see Figure 4).

[6] Personally I believe, however, that the computer is a secondary reason (many aspects of control by computers are captured by discrete-time systems) and the major motivation for hybrid systems comes from the limitations of continuous mathematics.

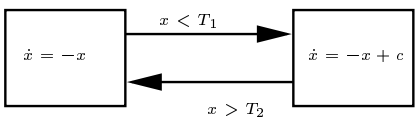

Figure 3: A heating system with a thermostat.

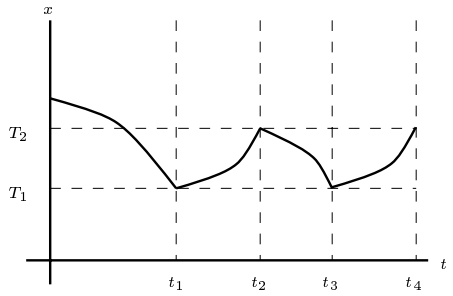

Figure 4: A typical behavior of the hybrid automaton of Figure 3 with discrete transitions occurring at times $t_1, \ldots, t_4$.

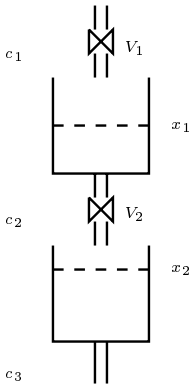

Figure 5: Two containers.

The next example is concerned with the modeling of the two containers of Figure 5. We assume that when valve V_1 is open, water flows into the first container at a fixed rate c_1. Similarly when V_2 is open, water flows from the first to the second container at rate c_2. Finally water leaves the third container at rate c_3. A simplified hybrid model, assuming containers with unlimited capacities, appears in Figure 6 where the discrete states A, B, C and D corresponds to the possible combinations of open and closed valves. The transitions to states E and F are taken when the first container becomes empty.[7]

Although the behaviors of hybrid automata are quite intuitive, the combination of continuous evolution with abrupt changes may evoke fundamental conceptual problems. For example, as careful readers might have noticed, the automaton of Figure 6 can switch back and forth between states D and F when $c_2 > c_3$. A similar phenomenon occurs in the thermostat example (Figure 3) when $T_1 = T_2$. Such behaviors where an unbounded number of discrete transitions take place in a bounded (or even zero) time interval have been investigated by control theorists under titles such as chattering and sliding mode. In the hybrid literature these are called *Zeno behaviors* in honor of the first documented presentation of problems arising from combining discrete and continuous reasoning.

In general, traditional questions concerning uniqueness and existence of solutions are harder to formulate and prove in this context and it is an interesting question whether one

[7]For simplicity, I have ignored the possible emptying of the second container.

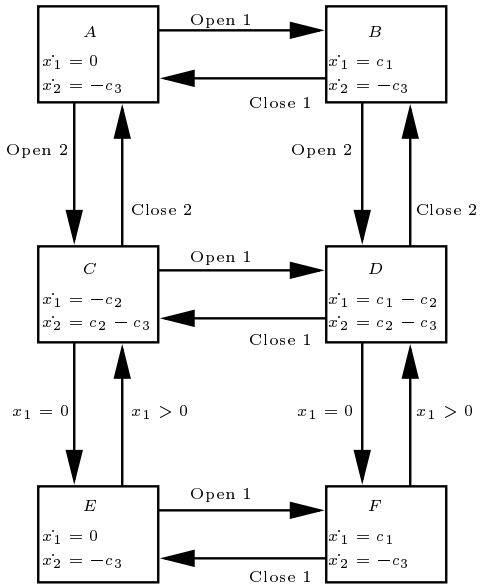

Figure 6: A simplified hybrid automaton for the two containers.

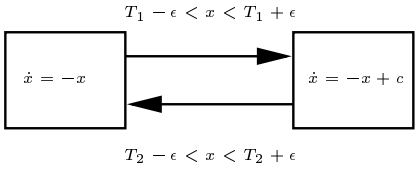

Figure 7: A heating system with a thermostat having uncertain bounds.

should worry about it. It is not clear to me whether, beyond considerations of mathematical tradition, these questions are important for the real-world problems that the equations and the automata are supposed to model. There is so much uncertainty in the models and in the influence of disturbances so that insistence on a unique solution for each initial condition seems ceremonial and irrelevant. On the contrary, in hybrid automata such an uncertainty can be explicitly stated, leading to infinitely many possible behaviors from each initial state. For example, imprecision in the thermostat measurements can be modeled by specifying non-deterministic[8] transition conditions as in the automaton of Figure 7 which admits uncountably many behaviors depending on when the transitions between the discrete states occur.

The methods and experimental tools for analyzing and synthesizing controllers for these hybrid models are beyond the scope of this introduction and the reader is invited to consult the project web page for more information.

[8] In Computer Science non-determinism is usually set-theoretic rather than probabilistic in nature, in the same sense that a relation can be viewed as a non-deterministic function having several possible results. In other words, non-deterministic dynamical systems are closer to differential inclusion rather than to stochastic differential equations.

3 The VHS project

The VHS project started in May 1998 by a consortium consisting mostly of computer scientists motivated by the potential application of verification techniques to hybrid systems. Other partners were control theorists and engineers mostly from the process industry. In addition to theoretical results and tools, the partners committed themselves to investigate several industrial and academic case-studies. Some of those have proved to be very valuable in demonstrating the limitations of the theoretical ideas and in suggesting more effective research directions.

The major lessons from the project can be summarized as follows. A typical industrial application consists of a physical plant and a controller. The plant is where hybrid processes take place. This processes can be chemical reactions, stages in the processing of steel, etc. and they have many discrete components such as switches or valves. The controller consists of a combination of human operators and computers[9] that issue commands that influence the dynamics of the plant. Both components can be modeled and analyzed at different levels of abstractions. A detailed model of the plant may involve many continuous variables subject to non-trivial differential equations — far beyond the current capabilities of hybrid verification technology. A comprehensive model of the controller needs to cover the implementation details of the digital control, including the programs, the computers and the execution environment (operating system and communication mechanisms). On the other hand, there is a very useful and tractable level of abstraction which can be captured by a special restricted class of hybrid automata, namely *timed automata*.

At this level of abstraction the continuous phases of the plant are modeled as discrete steps that take some time to complete. As an illustration consider first the hybrid automaton of Figure 8 which gives a detailed model of such a continuous step. The process starts when the continuous state variables are in some subset P of the state space, and this is modeled by a transition to state B. At that state some continuous evolution is taking place according to $\dot{\mathbf{x}} = f(\mathbf{x})$. This could model, for example, filling or emptying of a container, heating, a chemical reaction where state variables correspond to concentrations, etc. This step terminates and the automaton moves to state C when the state variables reach some part Q of the state space. Usually this corresponds to one or more variables crossing their respective thresholds, a fact which is supposedly detected by some sensors.

From a higher-level point of view, when we consider planning and sequencing, we are not so much interested in the exact details of the continuous evolution and in the intermediate values of $\mathbf{x}$. In a well-designed system, a low-level regulator takes care of fluctuations in the continuous trajectory and makes sure that the step will terminate in a reasonable time. At this level, models that can express bounds on the *duration* of each step are sufficient. Timed automata, i.e. hybrid automata where all continuous state variables are clocks that advance with derivative 1, provide a very effective modeling solution. The values of the clocks at each point in the evolution of the automaton represents the time elapsed since the beginning of the respective active steps. The step described by the hybrid automaton of Figure 8 can be modeled using the timed automaton of Figure 9 which has one clock variable y. Starting from state A, the transition to state B is accompanied by resetting the clock y to zero. The transition to state C can thus take place between t_1 and t_2 time units after the step started.

[9] For historical reasons these computers are often called Programmable Logic Controllers (PLC) or Distributed Control Systems (DCS).

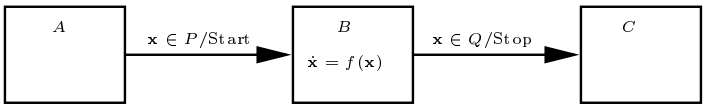

Figure 8: A hybrid automaton model for a process.

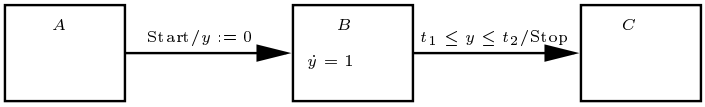

Figure 9: A timed automaton model for a continuous steps whose duration is in the interval $[t_1, t_2]$.

Such timed abstractions of continuous processes are very common in daily and technological life. We can find it in airplane schedules (rather than flight differential equations) in cooking instruction or in the specification of digital circuits (reasoning about switching delays instead of voltage patterns). Unlike more general hybrid automata, timed automata can, in principle, be verified automatically and there are experimental tools developed by the consortium members that can analyze them or generate schedules.

It should be noted that such timed abstractions are useful only under normal operation conditions. If something goes wrong, for example, a valve gets stuck or a container explodes in the middle of a heating step, the value of continuous variables such as water level or temperature are important for proving the plant safety.

When the continuous dynamics is complex and involves many variables that interact in a non-trivial way, such timed abstraction might be too coarse to be useful. However, we have observed that in practice designers tend to isolate processes into steps with more or less monotone evolutions, and with bounded temporal uncertainty. It is on the basis of such timed models that sequencing and scheduling of operations are done.

Following these observations coming from the case studies, the project evolved along three major axes, listed in an increasing order of importance.

Programming of Process Control Systems: We have looked at the practices of programming industrial computers with the hope of applying some verification techniques to this domain. We have found that this branch of programming is several generations behind the developments in more general-purpose software engineering. Notions such as structured programming, well-defined semantics for programs, compatibility between platforms, etc. have not yet penetrated into the field where most programming is done by engineers from non-computer disciplines. We have looked at the IEC-1131-3 standardization effort, investigated the languages supported by the standard and invested some work in defining their semantics and in applying verification techniques to them.

Real Hybrid Systems: This research direction was concerned with pushing the frontiers of automatic verification and controller synthesis for continuous and hybrid systems of the type described in Section 2. Several techniques based on mixed optimization and on approximate reachability analysis were developed and implemented in prototype tools. This is a new and exciting domain of research where a lot of progress has been made within the project. However, the capabilities of the developed tools do not yet match the complexity of real applications.

We intend to pursue this line of research in the forthcoming years in the framework of a follow-up project.

Timed Systems: Most of the effort in the project was directed to problems that can be treated at the level of abstraction of timed automata. One of the VHS case-studies, the *Sidmar Steel Plant*, was a major driving force behind this work. We have developed a new framework for solving scheduling problems based on modeling plants using timed automata. For classical problems such as job-shop scheduling under certainty, our techniques are not inferior to those based on constrained optimization. However, we believe that that automata-based techniques are more suitable for problems that involve uncertainty and other constraints which are not easily expressible in standard optimization models.

4 The Special Issue

After this long excursion let me describe briefly the papers that appear in this issue, centered around the experimental batch plant at Dortmund university. This is a toy (but real) plant used for educational purposes. It consists of several containers in which water is mixed with salt and then distilled again via evaporation. The plant is controlled by a soft PLC, that is, a PC programmed by special process control languages, which reads the sensors and issues commands to actuators such as valves, pumps and a heater.

The paper by *Kowalewski, Stursberg and Bauer* gives a detailed description of the plant at several levels of abstraction. The description includes the physical structure of the plant, the sensors, the continuous dynamics of the heating process, the estimation of the duration of the various steps and some of the code used to run the plant. The authors pose three classes of problems, each focusing on another description level of the plant:

- Continuous dynamics: show that in case of breakdown in the cooling system, the control algorithm guarantees that the temperature remains within safe margins. This problem requires a real hybrid model.
- Programming: prove that there are no bugs in the control software of the plant.
- Scheduling: how to produce the maximum number of batches using a timed model of the plant.

The paper by *Bemporad, Torrisi and Morari* attacks the first problem, namely verification of a hybrid model of the plant. The authors use their formalism of mixed logical dynamical systems in discrete time to encode both the continuous evolution of the plant and the switching behavior of the controller. Reachability analysis is applied to this model to prove that indeed the temperature stays in the right interval.

The paper by *Huuck, Lakhnech and Lukoschus* focuses mostly on the discrete aspects of the plant. The authors model the plant components as automata. For entities such as water tanks, this consists of abstracting the continuous process of filling the tank into a discrete transition from "empty" to "full". In addition, the control programs of the plant, are translated from SFC (sequential function charts) into automata. On this model the authors prove many properties that can be verified without a detailed continuous model, for example, "pumps are not pumping against closed valves".

The paper by *Mader, Brinksma, Wupper and Bauer* focuses on discrete aspects of the plant but it applies a somewhat different approach. The authors start with higher-level specifications of the required properties of the plant and successively refine them, by committing to more and more implementation decisions, until the control program is derived.

Finally, the paper by *Niebert and Yovine* treats the plant at the scheduling level and demonstrates how methods based on timed automata can be used to derive dynamic schedulers, that is, schedulers that observe the state of the plant and can cope with uncertainty in the duration of the processing steps. Control code can be automatically derived from the resulting scheduler.

This special issue contains quite a lot of words, formulae and code concerning a small plant that does nothing but mix water with salt. However, I believe that this case study demonstrates on a small scale many important and general problems concerning the design and implementation of complex industrial plants. I hope that this special issue will serve as an introduction to these problems and as a primal sketch of a new hybrid design methodology.

Acknowledgments: This manuscript benefited from comments by Eugene Asarin, Bruce Krogh and Amir Pnueli. I would like to thank all the partners of the VHS project for the great work they have done. More information on the project can be found on its web page:

`http://www-verimag.imag.fr/VHS/`