A Model Checking Technique for the Verification of Fuzzy Control Systems

Benedetto Intrigila
Dip. di Matematica Pura ed Applicata
Univ. di Roma "Tor Vergata", Roma, Italy
intrigil@mat.uniroma2.it

Daniele Magazzeni, Alberto Tofani
Dip. di Informatica
Univ. di L'Aquila, L'Aquila, Italy
{magazzeni,tofani}@di.univaq.it

Igor Melatti
School of Computing
Univ. of Utah, Salt Lake City, USA
melatti@cs.utah.edu

Enrico Tronci
Dip. di Informatica
Univ. di Roma "La Sapienza", Roma, Italy
tronci@di.uniroma1.it

Abstract

Fuzzy control is well known as a powerful technique for designing and realizing control systems. However, statistical evidence for their correct behavior may be not enough, even when it is based on a large number of samplings.

In order to provide a more systematic verification process, the cell-to-cell mapping *technology has been used in a number of cases as a verification tool for fuzzy control systems and, more recently, to assess their optimality and robustness.*

However, cell-to-cell mapping is typically limited in the number of cells it can explore. To overcome this limitation, in this paper we show how model checking *techniques may be instead used to verify the correct behavior of a fuzzy control system.*

To this end, we use a modified version of the Murphi verifier, which ease the modeling phase by allowing to use finite precision real numbers and external C functions. In this way, also already designed simulators may be used for the verification phase.

With respect to the cell mapping technique, our approach appears to be complementary*; indeed, it explores a much larger number of states, at the cost of being less informative on the global dynamic of the system.*

1 Introduction

1.1 Fuzzy control

Fuzzy control is well known as a powerful technique for designing and realizing control systems, especially suitable when a mathematical model is lacking or is too complex to allow an analytical treatment [16].

On the other hand, fuzzy control systems (FCS in the following) are based on qualitative fuzzy rules which have the form:

if <CONDITION> **then** <CONTROL ACTION>
where both <CONDITION> and <CONTROL ACTION> are formulated making use of the so called "linguistic variables", which have a qualitative, non mathematical character [13].

Therefore, by its very nature, a FCS leaves open a number of natural questions:

1. Does the FCS actually work?
2. How good is the FCS w.r.t. various performance measures, and in particular w.r.t. time optimality?
3. How robust is the FCS? That is, what will be the performance of the FCS when parameters vary outside the design range?

Several powerful technologies have been devised to cope with these problems.

For what concerns the first one, one has typically two strategies.

In the first strategy, fuzzy rules are devised on the basis of intuitive considerations. Then they are substantiated and tuned by means of some automatic statistical analyzer such as a neural network or a genetic algorithm [18].

In the second strategy, the very choice of fuzzy rules is guided by statistical considerations, such as in Kosko space clustering method [14] or by abstracting them from a neural network [22].

However, statistical evidence may be not enough, even when it is based on a large number of samplings. This is especially true when the controlled system displays a *critical character*, i.e. when malfunctioning may cause relevant economic losses or damage to persons. Therefore there is a need of a more systematic verification process.

To this aim, the *cell-to-cell mapping* technology has been used in a number of cases as a verification tool for FCSs and, more recently, to assess the optimality and the robustness of the FCSs.

For the sake of the reader we recall some of the main features of this approach.

1.2 Cell-to-cell mapping

Cell-to-cell mapping allows an approximated analysis of a state space, which however has a full coverage on the region of interest. The approximation stems from the fact that the continuous state space is partitioned into a finite number of disjoint cells. Thus, each variable ranges on the set of cells, instead of $\mathbb{R}$.

More in detail, suppose we are given a model of the form $x(t+1) = f(x(t))$, where the state x is described by n real-valued variables. Then, we can see x as a point on $\mathbb{R}^n$. In the cell-to-cell mapping, the n axes of the state space are partitioned into equal intervals, each denoted by an integer z_i.

These axes partitions naturally define n-dimensional *cells*. Indeed, a cell z is defined as a n-tuple of intervals $z = [z_1, \ldots, z_n]$. The union of all cells z is the cell space Z.

The main effect of the cell partition is that all the elements in a cell z are approximated with the center point $z^c = [z_1^c, \ldots, z_n^c]$ of that cell, where z_i^c is the central point of the interval z_i. Thus, the *image* or *one-step transition cell* for a cell z within a time period t is determined by computing $x = f(z^c)$, and determining the cell in which x falls. Therefore, the state space becomes the set of cells Z and the map f becomes the *cell mapping* $F : Z \rightarrow Z$.

A cell z^* is an *equilibrium cell* if $z^* = F(z^*)$. A periodic motion with period K is a sequence of K distinct cells $z^*(m)$, $m = 0, ..., K-1$, which satisfies the conditions

$$z^*(m) = F^m(z^*(0)), z^*(0) = F^K(z^*(0)).$$

An equilibrium cell is a periodic motion of period 1.

The *r-step domain of attraction* of a periodic motion is the set of all cells that are within r-steps of the periodic motion.

The cell map of a system is constructed using an *unravelling algorithm* to compute cell transitions and identify all existing periodic motions and domains of attraction. This algorithm computes an image or one-step transition cell for each cell z after a time period t_m. Cell trajectories are obtained by tracing one-step cell transitions.
Based on these trajectories, one can establish which cells converge to the set point (controllable cells) and which not (uncontrollable cells).

For detailed information about the cell mapping, the unravelling algorithm and applications, readers are referred to [4, 11].

1.3 Using Model Checking Techniques

We notice that the cell mapping technique is very similar to classical model checking techniques, extensively used in different domains such as hardware or protocols verification. Indeed, a finite state space is systematically searched by means of exhaustive algorithms.

Both techniques are plagued by the so called state explosion phenomenon, which means (for the cell-to-cell mapping) that to obtain a better approximated dynamics we need finer partitions, and this ultimately results in an exponential grow of the number of cells.

On the other hand, model checking techniques aim to an exhaustive exploration of the full state space. In this case, the state explosion phenomenon simply arises from the presence of a too large state space, w.r.t. the computational resources such as time or RAM memory. For control systems, and also for hybrid systems, the state space definition involves continuous variables, and this in turn appears to give rise to an unavoidable state explosion.

Many verification tools (*model checkers*) are available for automatic verification of hybrid systems. Examples are: HyTech [7, 1] and UPPAAL [31]. Also tools originally designed for hardware verification have been used for hybrid systems verification. E.g. in [30], SMV [17, 24] has been used for verification of chemical processing systems.

When the controller is a *fuzzy* one the analysis problem becomes even harder because of the complex dynamics involved. In this paper we show how a suitable extension [5] of the Murφ model checker can be effectively used in the automatic verification of nontrivial fuzzy control systems.

Some important points are to be noticed:

1. **Reachable States.** Of course we are interested in the states of the system that can be reached from legitimate initial states by means of the proper system dynamics. This set, called the set of *reachable states*, often turns out to be of a much smaller size than the whole state space (the cartesian product of the state parameters ranging intervals). It is worth notice that, in sharp contrast with simulation, the model checking approach is based on a full book-keeping of the visited states. So that only reachable states have to be taken into account.

2. **Memory Requirements Reduction.** Much work has been done to reduce the RAM memory required to complete a verification. The most important results have been achieved by cutting off state space portions so that the verification response is still valid (e.g., by

partial order reduction [10] and symmetry reduction [12]). Other important results have been gained by reducing the size of the states to be stored in RAM, especially by means of *hash compaction* [32, 29] (other important techniques having the same aim are described in [9, 8]). On the other hand, in an *auxiliary storage* approach one tries to exploit disk storage as well as distributed processors (network storage) to enlarge the available memory (and CPU). Examples can be found in [27, 28]. Finally, in [6] two algorithms are shown which exploit statistical properties of the state transition systems (*transition locality*), thus succeeding in verifying systems which are out of reach for the standard model checking techniques.

3. **Near the Set Point.** For control systems, one is often interested in the behavior of the system when its state is not too far from the set point. This results in a reduction of the state space and allows a systematic exploration of the states near the system initial state. Using this approach we were able to complete the automatic verification of a turbogas control system [5].

4. **Reduction of the Number of Initial States.** Another reasonable reduction of the state space can be obtained by suitably choosing the initial states. As an example, in a *car parking* problem we can assume that the controller will behave in a similar way, when starting with initial states which are "very near" one to the other.

In the present paper we propose to use *explicit model checking* for the verification of control systems. To this end, we use a modified version of the tool Murphi [3], devised by us and other researchers, that allows the use of real numbers (that are treated as *double* type of the programming language C) as well as the use of external C functions.

This simplifies the modeling phase of the controller (within the mathematical model), since allows us to use, with a few and not conceptual modifications, the controller simulators, which are typically built during the design of the controller itself, typically for testing purposes.

Note that using this approach we can obtain a systematic analysis of the state space, considering millions of states, whilst the cell mapping approach typically consider (up to) thousands of cells.

We stress the fact that this is very near to a "press-button" technology, in the sense that once the controller has been devised and formulated say as a C function, and the initial state(s) as well as the allowed parameters variations have been set, then an automatic verification of the controller takes place. This verification might not terminate (or not terminate in a reasonable amount of time) due to limitations in the computational resources. However if it terminates, then it outputs either an error trace (i.e. a path leading from an initial state to a state that the controller is unable to bring to the set point) or a "certification" that - in the considered range of variability - the controller acts properly. Often, this can be done in a reasonable amount of time.

Moreover, this state exploration makes it possible to perform various optimality analysis. For example, time optimality can be assessed by computing - for each state - the time needed to bring it to the set point.

In the following, we present three applications of our methodology. In the first one we consider the classical *car parking* problem, and we verify a fuzzy car parking controller due to [14][15]. In the second one, we consider another classical problem: *the inverted pendulum*. In this case, we verify the fuzzy controller proposed in [23]. In both cases, we limit the state space by reducing the initial states. Nevertheless, in the verification process millions of states are considered. Moreover, we were able to determine an upper bound to the number of steps needed to bring the controlled states to the set point.

Our third example is the verification of a fuzzy buck controller devised in [25]. Here, we adopt the approach of limiting the size of the perturbing stimuli. We were able to verify the controller for all the sequences of stimuli belonging to a given range. We also determined at which point the controller breaks down, so assessing the controller robustness.

These examples show that explicit model checking could be a valid technique for the verification of fuzzy controllers. Due to the great easiness of use, it seems reasonable that our tool can be used also for the tuning phase in the realization of fuzzy controllers.

With respect to the cell mapping technique, our approach appears to be *complementary*; indeed, it explores a much larger number of states, at the cost of being less informative on the global dynamic of the system.

The paper is organized as follows. In Sect. 2 we outline our approach, together with the modifications we have done to the Murphi verifier. In Sect. 3 we describe how we applied model checking techniques to our case studies, and we also give some experimental results. Sect. 4 concludes the paper.

2 Extending Murphi Input Language with Real Numbers and External C Functions

2.1 Finite Precision Real Numbers

Murphi [19] is a model checker based on an explicit enumeration of the state space (other explicit model checkers are e.g. SPIN [26], while NuSMV [20] is an example of implicit (or *symbolic*) model checker). Originally, Murphi was developed to verify protocol-like systems, especially *cache coherence* protocols. However, we have already shown [5] that it can be also used to verify hybrid systems, by adding the possibility to use *finite precision real numbers* in the modeling phase.

To do this, we added to the Murphi verifier the type `real(m, n)` of real numbers with m digits for the mantissa and $\overline{n} = log_{10}\lfloor n \rfloor + 1$ digits for the exponent. Type `real(m, n)` is finite, its cardinality is $2 \times 9 \times 10^{m-1} \times 2 \times 10^{\overline{n}} = 36 \times 10^{m+\overline{n}-1}$. Thus our extension has no impact on Murphi verification algorithms, however makes it easier to model hybrid systems as well as control systems within Murphi.

Note that, the huge cardinality of the type `real(m, n)` does not imply, *a priori*, a huge size of the set of reachable states.

2.2 External C Functions

Still, this is not enough to model *fuzzy* controllers behavior. In fact, the Murphi modeling language is very simple (being designed for cache coherency protocols), so the modeling phase may be still time consuming and error prone. Indeed, the implementation of the fuzzification phase, as well as of the system dynamics (once the controller has computed its output) requires more advanced language constructs to be described, such as the ones provided by the C/C++ language. Moreover, system simulators written in C/C++ are often available (especially in the more complex cases) for testing purposes, thus it is worth doing to reuse them in the verification phase.

To overcome these difficulties, and to reuse simulators, we added the possibility to use externally defined C/C++ functions in the modeling language. In this way, we can use the C/C++ language constructs to model the fuzzy controllers. Moreover, we can directly include (with some arrangement) in the Murphi model a simulator for the system under analysis, since a system simulator is almost always written in C/C++.

To make this possible, we had to properly modify Murphi. The modification details cannot be described here; we simply point out that the extension has been possible since Murphi uses C/C++ language to generate the successor states of a given state.

There are some limitations on the functions that may be used. The main limitation is that functions cannot have collateral effects, since the verification algorithm of Murphi does not guarantee the calls to be in the expected order. However, it is always possible to modify a C/C++ function so that it does not have collateral effects.

2.3 Verification strategy

In our case studies, we use Murphi in two different ways.

When we want to verify a controller acting in a *disturbed environment* (it is the case of the buck controller of Sect. 3.3), Murphi performs the following computation. While visiting a generic state s, it nondeterministically generates the appropriate *stimuli* (or *disturbances*) $d_1, \ldots, d_n$ to system, and calls n times the simulator to determine the $t_1, \ldots, t_n$ successor states of s (note that to each disturbance corresponds one successor state, since controllers are deterministic). Then, $t_1, \ldots, t_n$ are stored in a queue to be eventually expanded in the same way, provided that they have not been visited before. This check is done by looking for t_i in a hash table maintaining all the states visited so far. Here, the system *initial state* is a state internal to the set point, and we verify if the controller is able to maintain the system within the set point nearbies, notwithstanding the disturbances.

On the other hand, in the two remaining case studies, there are no disturbances, so each state has exactly one successor state. Here, we simply set *all the possible initial states of the system* (within a given *precision*), and verify that the controller is always able to reach the goal in a given number of steps.

Note that, in checking for already visited states, finite precision real numbers are used, so that slightly different states may be declared equal. However, the error arising from this loss of precision will be acceptable if the choice of real number representation has been adequate for the system in hand. This means that a right choice of the precision is an aspect of the verification design.

3 Case Studies

In this section we show how our Murphi based validation methodology for fuzzy controllers may be applied. To this end, we use three case studies: a *Parking Control System*, an *Inverted Pendulum Control System* and a *Buck Tension Control System* (PCS, IPCS and, respectively, BCS in the following). In each case, we begin by describing the system and the controller under analysis, then we show how we modeled them in Murphi. Finally, some experimental results are provided, where we set the number of decimal digits of mantissa of our real number precision to be 4 for PCS and BCS and 7 for IPCS. On the other hand, 2 digits are always sufficient for the exponent.

3.1 A Parking Control System

The goal of the Parking Control System (PCS in the following) is to back a car up to a parking place starting from any initial position in the parking lot [14]. We describe the car state with three real values:

- the abscissa and the ordinate of the car $x, y \in [0, 12]$, referred to the center of the rear wheels;

- the angle $\varphi \in [-90°, 270°]$ of the longitudinal axis of the car w.r.t. the horizontal axis of the coordinate system.

The PCS takes as input the car position and outputs the steering wheels angle $\theta \in [-30°, 30°]$. The goal is to move the car to a final position satisfying $x = 6, \varphi = 90°$. We have no restrictions on the y coordinate since we assume the initial position to be sufficiently far away from the parking place. Therefore, if the final position is reached, to move the car to the parking place it is sufficient to drive back without steering the wheels.

The PCS fuzzy controller we want to verify is defined in [14]. For lack of space we omit the description of fuzzy sets and rules.

In order to model the PCS in Murphi, we simply have to implement in an external C function (see Section 2) the fuzzification, inference and defuzzification process.

Moreover, we have to be able to determine the final position of the car once a decision has been output by the PCS. To this end, we use analytic equations giving $x(t+1)$, $y(t+1)$ and $\varphi(t+1)$ as functions of $\theta(t)$ (PCS output), $x(t)$, $y(t)$ and $\varphi(t)$ (preceding position), as well as of the car length (1.5m), of the constant car velocity (1.5m/s) and of the sampling period (0.1s).

We are now ready to speak about the PCS verification. Namely, we want to verify that the PCS is able to back the car up to the parking place in a maximum given number of steps (`NUMSTEPS` in the following), whatever is the starting position.

To this end, we define n starting states in the following way. We discretize the domains of the Murphi variables `x`, `y` and φ as follows:

- We divide the interval $[0, 12]$ in to 270 subintervals and consider each value $x = \frac{12}{270}i$ for $i = 0..270$ as possible initial values of the coordinates x, y; note that we choose 270 since it is 10 times greater than the value used in [15] for a cell to cell mapping division of the state space;

- We divide the interval $[-90, 270]$ in to 360 subintervals and consider each value $\varphi = -90 + i$ for $i = 0..360$ as possible initial value of the angle φ;

With these assumptions the Murphi verifier operates as follows. For each state s a suitable Murphi variable `step` counts the number of manoeuvring done to reach s and a suitable predicate `Parking` is true when the car is (approximately) in the parking place.

Therefore, a given state s will be an *error state* (i.e., the controller should not bring the system in s) if the value of `step` is greater than `NUMSTEPS` and the predicate `Parking` is not true.

Tab. 1 summarizes one of our experimental results.

Table 1. Experimental Results for the PCS

NUMSTEPS	States	Error States	States OK
60	3200424	9.46%	90.54%

3.2 An Inverted Pendulum Control System

The IPCS is a fuzzy controller for the inverted pendulum problem [23]. The goal is to bring the pendulum to equilibrium by applying a horizontal force to a maneuverable cart. We describe the pendulum state with two real values:

- the pendulum angle w.r.t the vertical axis $\theta \in [-1.5, 1.5]$;

- the angular velocity $\dot{\theta} \in [-8, 8]$rad/s.

The IPCS takes as input θ and $\dot{\theta}$ and outputs the force g to be applied to the cart. The goal is to keep θ as close as possible to 0.

The IPCS fuzzy controller we want to verify is defined in [21]. For lack of space we omit the description of fuzzy sets and rules. Moreover, as for the IPCS case, we have to be able to determine the final position of the pendulum once a decision has been output by the IPCS. To this end, we use analytic equations giving $\ddot{\theta}(t)$ as function of g (IPCS output), $\theta(t)$ and $\dot{\theta(t)}$ (preceeding position), as well as of the gravitational acceleration constant, the mass of the pendulum (0.1kg), the mass of the cart (0.9kg), is the length of the pendulum (1m).

We are now ready to speak about the IPCS verification. Namely, we want to verify that the IPCS is able to bring the pendulum to the equilibrium position whatever is the starting position. Moreover we want to determine the maximum number of steps required.

To this end, we define n starting states in the following way. We discretize the domains of the Murphi variables θ and $\dot{\theta}$ as follows:

- We divide the interval $[-1.5, 1.5]$ in to 100 subintervals and consider each value $\theta = -1.5 + \frac{3}{100} * i$ for $i = 0..100$ as possible initial value of the angle θ;

- We divide the interval $[-8, 8]$ in to 100 subintervals and consider each value $\dot{\theta} = -8 + \frac{16}{100} * i$ for $i = 0..100$ as possible initial value of the angular velocity $\dot{\theta}$;

Tab. 2 summarizes one of our experimental results.

3.3 A Buck Tension Control System

The Buck Tension Control System (BCS in the following) is a fuzzy controller for a Buck DC/DC Converter described in [25]. The goal of BCS is to maintain the Buck output voltage V_0 inside its set point when there is a variation of the Buck input resistance R. Here, following [25], we define the set point for V_0 to be the range $V_{ref} \pm d_{ref}$, where d_{ref} is a given tolerance.

In order to reach its goal, BCS takes as input the error $e_k = V_0 - V_{ref}$ at a step k (typically due to a change in the resistance R) and the change of error $ce = e_k - e_{k-1}$, and outputs the buck duty cycle value variation δ_k. Thus, the new duty cycle will be $d_k = d_{k-1} + \eta \cdot d_k$, being η a constant defining the the gain factor of the fuzzy controller. This new duty cycle should take V_0 back as close as possible to V_{ref}.

In [25] it is shown by simulation that BCS works correctly when the variations of R happen not too frequently. In this section we will show how Murphi can be used to formally prove that a given time interval t_R between the variations of R is indeed safe for BCS, and we show how it is possible to find the least t_R (`MAX_OUT` in the following) with this property.

In order to model BCS in Murphi, we again have to implement the fuzzy control behavior as well as to compute the values for the controlled Buck parameters when the FCS has made a decision.

To this end, we directly used an already available simulator [2] for the whole BCS (fuzzy control included), which is able to compute the BCS state after t seconds. The main work here has been done to adapt the BCS simulator to our needs, i.e. to understand which variables it is necessary to store in the Murphi state, in order be able to safely restart the simulator from a given intermediate position. However, this task has been carried over in 2 days, without requiring knowledge of the Buck internal working. This shows the feasibility of our approach.

Thus, our Murphi model is organized in only 3 rules. The first two rules handle the disturbances, which nondeterministically may or may not take place with a fixed frequency. In other words each state s has an internal counter (incremented at each step) such that when the counter reaches a fixed value `DistFreq` then s has two successors: s_1 and s_2; in s_1 the disturbance has taken place while in s_2 has not. In both s_1 and s_2 the counter is set to 0.

The third rule calls the simulator on a small time interval t. This is done only after having set the proper simulator variables with the Murphi state variables. The inverse operation is performed when the simulator terminates its execution, so as to allow Murphi to store the new obtained state.

Finally, we want to verify that the number of consecutive steps in which the buck is outside its set point is less than a given number of steps (`MAX_OUT`). To this end, we count the number of consecutive steps in which the buck output is outside its set point by means of the internal variable `num_steps_out`. Thus, the property to be verified states that in each state `num_steps_out <= MAX_OUT`. To determine the least value for `MAX_OUT` for which the BCS is correct, we simply run many verification, each time increasing the `MAX_OUT` value. Our results are in Tab. 3.

Table 2. Experimental Results for the IPCS

NUMSTEPS	States	Error States	States OK
188	570246	29.52%	70.48%

Table 3. Experimental results for the BCS

SimTime (s)	DistFreq	d_{ref} (V)	MAX_OUT	Res
10^{-4}	1	10^{-2}	30	FAIL
10^{-4}	2	10^{-2}	0	FAIL
10^{-4}	2	10^{-2}	1	OK

Note that for too frequent disturbances, the BCS fails to maintain the output of the buck in its set point for 3 ms, which is typically unacceptable. On the other hand, for higher values of the disturbance interarrival times, BCS succeeds in its task.

4 Conclusions

In this paper we proposed to use *model checking* techniques to verify fuzzy control systems. This allows us to explore much more states than other techniques used to this purpose, like the *cell to cell mapping*.

To prove the feasibility of our approach, we have applied it to three case studies: the classical *car parking* and *inverted pendulum* problems, plus the verification of a fuzzy controller for a buck tension system [25]. For the latter example, we was able to determine at which point the controller breaks down, so assessing the controller robustness. For the first two examples, we verified that the controller is able to take the car or the pendulum to its goal in a given number of steps. These examples show that explicit model checking could be a valid technique for the verification of fuzzy controllers. Due to the great easiness of use, it seems reasonable that our tool can be used also for the tuning phase in the design of fuzzy controllers. We plan to do this in a future work.

References

[1] A User Guide to HyTech: http://www.eecs.berkeley.edu/~tah/HyTech, 2004.

[2] Buck simulator : http://www.equars.com/ marco/poli/tesi/node76.html, 2005.

[3] Cached Murphi Web Page: www.cs.utah.edu/formal_verification/software/murphi/murphi.prob, 2004.

[4] Y. Chen and T. Tsao. A description of the dynamical behavior of fuzzy systems. *IEEE Trans. Systems Man Cybernet*, 19(4):745–755, Jul/Aug 1989.

[5] G. Della Penna, B. Intrigila, I. Melatti, M. Minichino, E. Ciancamerla, A. Parisse, E. Tronci, and M. Venturini Zilli. Automatic verification of a turbogas control system with the murφ verifier. In O. Maler and A. Pnueli, editors, *Hybrid Systems: Computation and Control, 6th International Workshop, HSCC 2003 Prague, Czech Republic, April 3-5, 2003, Proceedings*, volume 2623 of *Lecture Notes in Computer Science*, pages 141–155. Springer, 2003.

[6] G. Della Penna, B. Intrigila, I. Melatti, E. Tronci, and M. Venturini Zilli. Exploiting transition locality in automatic verification of finite state concurrent systems. *STTT*, 6(4):320–341, 2004.

[7] T. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1(1):110–122, dec 1997.

[8] G. Holzmann and A. Puri. A minimized automaton representation of reachable states. *Software Tools for Technology Transfer*, 2(3):270–278, November 1999.

[9] G. J. Holzmann. An analysis of bitstate hashing. *Form. Methods Syst. Des.*, 13(3):289–307, 1998.

[10] G. J. Holzmann and D. Peled. An improvement in formal verification. In *Proceedings of the 7th IFIP WG6.1 International Conference on Formal Description Techniques VII*, pages 197–211, London, UK, UK, 1995. Chapman &amp; Hall, Ltd.

[11] C. S. Hsu and R. S. Guttalu. An unravelling algorithm for global analysis of dynamical systems - An application of cell-to-cell mappings. *ASME Transactions Series E Journal of Applied Mechanics*, 47:940–948, Dec. 1980.

[12] C. N. Ip and D. L. Dill. Efficient verification of symmetric concurrent systems. In E. Straub, editor, *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 230–234, Cambridge, MA, Oct. 1993. IEEE Computer Society Press.

[13] J. Jin. *Advanced Fuzzy Systems Design and Applications*. Physica-Verlag, 2003.

[14] B. Kosko. *Neural Networks and Fuzzy Systems*. Prentice Hall, 1992.

[15] M. C. Leu and T.-Q. Kim. Cell mapping based fuzzy control of car parking. In *ICRA*, pages 2494–2499, 1998.

[16] H. Li and M. Gupta. *Fuzzy Logic and Intelligent Systems*. Kluwer Academic Publishers, 1995.

[17] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[18] T. Miyoshi, S. Tano, Y. Kato, and T. Arnould. Operator tuning in fuzzy production rules using neural networks. In *Proc. IEEE Int. Conf. on Fuzzy Systems 1993*, pages 641–646, San Francisco, Mar. 1993.

[19] Murphi Web Page: http://sprout.stanford.edu/dill/murphi.html, 2004.

[20] NuSMV Web Page: http://nusmv.irst.itc.it/, 2004.

[21] M. Papa, J. Wood, and S. Shenoi. Evaluating controller robustness using cell mapping. *Fuzzy Sets and Systems*, 121(1):3–12, 2001.

[22] S. Sekine, N. Imasaki, and E. Tsunekazu. Application of fuzzy neural network control to automatic train operation and tuning of its control rules. In *Proc. IEEE Int. Conf. on Fuzzy Systems 1993*, pages 1741–1746, Yokohama, 1995.

[23] S. Smith and D. Comer. An algorithm for automated fuzzy logic controller tuning. In *Proc. IEEE Internat. Conf. on Fuzzy Systems 1992*, pages 615–622, 1992.

[24] SMV Web Page: http://www-2.cs.cmu.edu/∼modelcheck/smv.html, 2004.

[25] W.-C. So, C. K. Tse, and Y.-S. Lee. Development of a fuzzy logic controller for dc/dc converters : Design, computer simulation and experimental evaluation. *IEEE Trans. on Power Electronics*, 11(1):23–32, January 1996.

[26] SPIN Web Page: http://spinroot.com, 2004.

[27] U. Stern and D. Dill. Parallelizing the murφ verifier. In O. Grumberg, editor, *Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22-25, 1997, Proceedings*, volume 1254 of *Lecture Notes in Computer Science*, pages 256–278. Springer, 1997.

[28] U. Stern and D. Dill. Using magnetic disk instead of main memory in the murφ verifier. In A. J. Hu and M. Y. Vardi, editors, *Computer Aided Verification, 10th International Conference, CAV '98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings*, volume 1427 of *Lecture Notes in Computer Science*, pages 172–183. Springer, 1998.

[29] U. Stern and D. L. Dill. Improved probabilistic verification by hash compaction. In *CHARME '95: Proceedings of the IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pages 206–224, London, UK, 1995. Springer-Verlag.

[30] A. L. Turk, S. T. Probst, and G. J. Powers. Hybrid and real-time systems. In O. Maler, editor, *Hybrid and Real-Time Systems, International Workshop. HART'97, Grenoble, France, March 26-28, 1997, Proceedings*, volume 1201 of *Lecture Notes in Computer Science*, pages 259–272. Springer, 1997.

[31] UPPAAL Web Page: http://www.docs.uu.se/docs/rtmv/uppaal/, 2004.

[32] P. Wolper and D. Leroy. Reliable hashing without collision detection. In C. Courcoubetis, editor, *Computer Aided Verification, 5th International Conference, CAV '93, Elounda, Greece, June 28 - July 1, 1993, Proceedings*, volume 697 of *Lecture Notes in Computer Science*, pages 59–70. Springer, 1993.