# Verifying a Knowledge-Based Fuzzy Controller

Miren Carranza

Technical Report 82 September 1992

Computational Logic Inc.
1717 W. 6th St. Suite 290
Austin, Texas 78703
(512) 322-9951

This work was supported in part at Computational Logic, Inc., by the Defense Advanced Research Projects Agency, ARPA Order 7406. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of Computational Logic, Inc., the Defense Advanced Research Projects Agency or the U.S. Government.

## 1. Introduction

The field of *formal methods* attempts to achieve system reliability by applying logic-based mathematical modeling in the construction of digital systems. Using rigorous, mathematically-based techniques that model programs and computing systems as mathematical entities, practitioners of formal methods attempt to prove that program models meet their specifications for all potential inputs. This approach augments more traditional, testing-based software and hardware engineering practice with *deductive* approaches to predicting system behavior and offers promise for enhancing the quality of digital systems, at least in selected applications. A subfield of formal methods is *automated reasoning* that attempts to partially mechanize the process of reasoning about system correctness. Mechanically supported formal methods have been applied to a variety of applications such as: language implementations [16, 20], operating systems [1], concurrent algorithms [8, 14], fault-tolerant systems [3], computer hardware [7], a simple real-time controller [5], and a wide variety of others.

We are attempting to apply such techniques to a new application domain, namely the correctness of fuzzy rule-based control systems. To this end, we have modeled a simple controller in the computational logic of Boyer and Moore [6] and proved some theorems about its behavior using the automatic theorem prover that supports that logic. We should note that both our fuzzy controller and the model of its behavior are quite simple. Modeling the controller within a formal logic permitted to specify, by means of theorems in the logic, some properties about its behavior. Several properties have been proved, with proofs checked by a mechanical theorem prover.

The following sections briefly outline the rationale of fuzzy control, describe a simple fuzzy control system and the modeling of that system within a formal logic, show the formalization of several correctness properties of the system, and describe the mechanically assisted proof of those properties. Appendix A describes briefly the Boyer-Moore logic and the automated proof system in which these proofs were carried out. Appendix B contains the complete formal definition of the controller and its correctness proofs.

## 2. Fuzzy Control

Conventional controllers cannot provide adequate control for systems that have time-varying parameters, unknown structures, and multiple objects. Yet numerous systems with such characteristics are controlled by humans. There have been several success stories in the construction of automated controllers based on human expertise, notably using fuzzy controllers [18]. Fuzzy rule-based control systems are similar to expert systems in that the rules embody human expert knowledge about the control operation that is being mechanized.

To illustrate, one of the first applications of fuzzy control was the automation of the operation of a cement kiln [10]. Though humans could operate the kiln efficiently, efforts to find a representative control function failed. The exact mathematical specification of the process proved too comprehensive and complex to build. A fuzzy representation of the process was developed and performed successfully. The rules that were used to represent the human reasoning were of the form:

- If the oxygen level is low, then add less coal.
- If the oxygen is adequate, then keep the coal input constant.
- If the oxygen is high, then add more coal.

In such a fuzzy system, more than one rule can be fired and, thus, more than one control action can be recommended. In the kiln controller two actions such as *keep coal feed constant* and *decrease coal* might fire at the same time. By combining these, the system arrived at the appropriate action, a slight decrease in the amount of coal supplied to the kiln. Fuzzy control theory specifies ways to combine a collection of recommended actions to obtain a single action of this sort.

A fuzzy controller properly consists of:

- linguistic variables;
- membership functions (fuzzification functions);
- a rule base;
- a fuzzy rule interpreter; and,
- a de-fuzzification function or functions.

The linguistic variables of the system are approximate terms describing measurements on the system parameters, such as *too low*, *OK*, *too slow* etc. Membership functions specify the *degree* of membership of a measurement in a fuzzy concept represented by a linguistic variable. For example, suppose we have the following membership function, where A is a linguistic variable such as *low level* and X is a crisp value describing level.

M((a)(x)) = 1 means x = a,
M((a)(x)) = 0.3 means that x = a with degree 0.3,
M((a)(x)) = 0 means x is not equal to a.

The membership functions thus determine how much a variable X belongs to different ‘‘regions’’ determined by the collection of linguistic variables. Membership functions are also called *fuzzification functions*. It is possible for X to be equal to A with degree 0.3 and also equal to B with degree 0.5, say. This must be taken into account by the rule interpreter.

The rule base is a collection of rules that collectively specify the controller. Each rule has form:

<antecedent> → <consequent>.

The antecedent of a rule gives a possible process state, in terms of linguistic variables; the conclusion specifies a control action or actions, again in terms of the linguistic variables. A sample control rule for our simple controller is:

<water-level is *high*, water-input-rate is *maximum*> → open the drain *totally*.

The rule base interpreter takes the fuzzy description of a process state and determines to what degree the antecedent of each rule matches the fuzzy representation of a process state. A crisp state might be fuzzified to more than one corresponding fuzzy process state. For example, if a sensor detects the water level of 25 and water input rate of 50, then it might be fuzzified to the two fuzzy states:

{(water-level = low, 0.7) and (water-input-rate = max, 1)};
{(water-level = med, 0.4) and (water-input-rate = max, 1)}.

This indicates that the water level falls somewhat within both the low and medium regions, though more within low. These fuzzy process states are then used to determine which rules from the rule base to apply.

In a fuzzy rule-based system all rules fire. However, rules have effect according to the degree of the match between the antecedent of the rule and the fuzzy process states. Since the fuzzification of a crisp process state may yield more than one corresponding fuzzy process state, more than one rule may match the process situation with a non-zero degree. The rule interpreter uses the degree of matching of the antecedents of the activated rules with the state to assign a weight to each of the recommended control actions. These are then converted into a crisp value that finally directs the actuator. In the example above, since the current level is more in the low than medium range, the combination of the rules may call for the drain to be closed *slightly*. There are many methods described in the literature [11, 12, 13] to combine the recommended fuzzy control actions into a single crisp control action. We describe our approach in a subsequent section.

The operation of the fuzzy controller may be summarized in the following list of steps.

1. Convert one or more crisp readings into fuzzy values using membership functions. This converts a crisp state into one or more fuzzy states.

2. Determine the degree to which each rule in the rule base applies for each of these fuzzy states. According to this degree, assign a corresponding weight to the fuzzy control action specified as the consequent of each rule.
3. By a de-fuzzification process applied to the various recommended fuzzy control actions, compute a number that represents the corresponding crisp control action.

In the following section we introduce our simple fuzzy controller and the way in which each of these steps is performed.

## 3. A Simple Fuzzy Controller

We have defined a simple fuzzy controller for a water tank with a variable input rate. The point of the controller is to keep the level of water in the tank within a certain intermediate range by adjusting the flow though a drain at the bottom of the tank. Periodically, crisp readings are taken that give the current water level (in the range [0…101]) and water input rate (in the range [0…51]). The water flow is adjusted by opening or closing the drain with settings between 0 and 65. Intuitively, a water input rate of, say, 30 and drain opening of 40 means that water is entering the tank at 30 units per minute and draining at 40 units per minute, allowing the tank level to decrease by 10 units per minute.

To fully define our controller, we must specify the linguistic variables, membership functions, rule base, fuzzy rule interpreter, and de-fuzzification scheme. We explain and illustrate each of these below.

### 3.1 Linguistic Variables and Membership Functions

The linguistic variables for our controller define the various regions for the water level, water input rate, and drain position. These are as follows:

| | |
|---|---|
| water level: | empty, low-level, med-level, high-level, full; |
| input rate: | no-flow, min-flow, med-flow, max-flow; |
| drain position: | closed, min-open, med-open, max-open, wide-open. |

Our membership functions define how particular crisp readings of water level and input rate are mapped into these categories.

We use two special classes of fuzzy numbers to define our membership functions: *triangular fuzzy numbers* and *trapezoidal fuzzy numbers*. A triangular fuzzy number `A` is defined by a triple `(A1, A2, A3)` with membership function:

$M_{tri}(A,x) = 0$, for $x < a1$,
$M_{tri}(A,x) = (x - a1)/(a2 - a1)$, for $a1 \le x \le a2$,
$M_{tri}(A,x) = (a3 - x)/(a3 - a2)$, for $a2 \le x \le a3$,
$M_{tri}(A,x) = 0$, for $x > a3$.

A trapezoidal fuzzy number is represented by a four-tuple `(A1, A2, A3, A4)`. The membership function of a trapezoidal number is characterized as:

$M_{trap}(A,x) = 0$, for $x < a1$,
$M_{trap}(A,x) = (x - a1)/(a2 - a1)$, for $a1 \le x \le a2$,
$M_{trap}(A,x) = 1$, for $a2 \le x \le a3$,
$M_{trap}(A,x) = (a4 - x)/(a4 - a3)$, for $a3 \le x \le a4$,
$M_{trap}(A,x) = 0$, for $x > a4$.

Notice that a triangular fuzzy number is a special case of a trapezoidal fuzzy number where $a2 = a3$.

The intuition behind these numbers is illustrated in Figure 1. Notice that the membership of X in the triangular number is determined by a projection onto the Y axis. The closer X is to A2, the greater its membership in the concept characterized by the fuzzy number. Membership is defined similarly for the trapezoidal number. Notice that Y has membership 1 in the concept since it falls into the plateau region between A2 and A3.

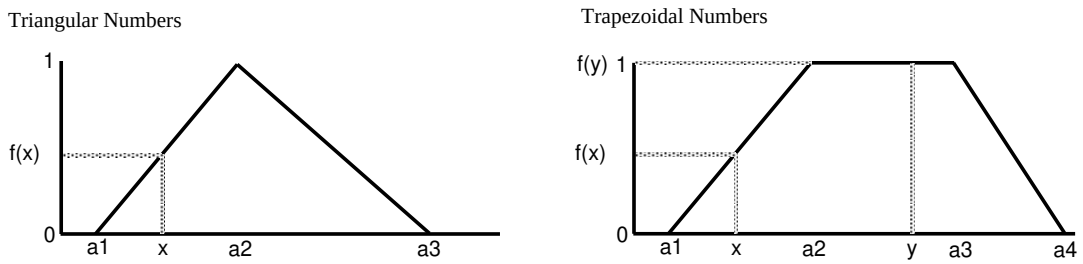


**Figure 1:** Fuzzy Numbers

For our controller, we define water level via trapezoidal numbers as follows:

| | |
|---|---|
| empty: | (0, 1, 2, 5) |
| low-level: | (3, 10, 30, 40) |
| med-level: | (30, 40, 60, 70) |
| high-level: | (60, 70, 90, 97) |
| full: | (95, 96, 100, 101) |

Thus, a crisp water level reading of 4 falls within the EMPTY region to some degree and within LOW-LEVEL to some other degree. Water input rate is defined via triangular numbers as follows:

| | |
|---|---|
| no-flow: | (0, 1, 3) |
| min-flow: | (2, 15, 25) |
| med-flow: | (20, 30, 35) |
| max-flow: | (30, 40, 51) |

Notice that defining our regions by triangles and trapezoids has a slightly odd effect at the boundaries. For example, a crisp water level value of 0 is *less empty* than a value of 2. We reconcile this by imagining that the actual bottom of the tank is at water level 2 and that no crisp values of water level below 2 are actually encountered.[1]

Now, given a pair of crisp values of water level we determine to what degree it belongs to each of the defining regions using the trapezoidal number and the membership function $M_{TRAP}$.[2] For a given crisp water level reading, the degree of membership is determined in each of the various water level regions by applying the $M_{TRI}$ membership function defined above. The result is a collection of fuzzy values listing to what degree the crisp reading falls into each of the regions. For example, for a crisp water level of 4 we have the following set of fuzzy numbers.

{(empty 33), (low-level 14), (med-level 0), (high-level 0), (full 0)}

We perform a similar computation on our crisp water input rate, using the trapezoidal numbers defining the various

[1]We could have used negative values to represent these boundary values. However, for certain technical reasons related to the Boyer-Moore logic, we wished to avoid dealing with negatives in our formalization.

[2]Notice that we are using natural number values in the range [0…100] to represent fuzzy numbers, rather than real numbers in the range [0…1]. Thus actually the numbers computed by our membership functions are scaled by 100. This is required due to the difficulty of dealing with reals in the Boyer-Moore logic. The fuzzy value, say, 52 in our representation represents the usual fuzzy value of 0.52.

input rate regions. We ignore any values for which the degrees are zero. Thus, for example, a crisp water level of 4 and input rate of 22 yields the following sets:

water level: {(empty 33), (low-level 14)}
input rate: {(min-flow 30), (med-flow 20)}

This means that a water level of 4 is in region `EMPTY` with degree 33 and in region `LOW-LEVEL` with degree 14. The crisp input rate of 22 is in the `MIN-FLOW` range with degree 30 and in the `MED-FLOW` range with degree 20.

We now combine all of the various possible fuzzy states corresponding to the single crisp input state.

{ {(empty 33), (min-flow 30)}
{(empty 33), (med-flow 20)} (*)
{(low-level 14), (min-flow 30)}
{(low-level 14), (med-flow 20)} }

This list of fuzzy states will determine the selection of rules to apply at the next step.

### 3.2 Controller Rules

In the control world it is desirable that the rules form a *complete* system. This ensures that for any process conditions (within the considered universe) the controller can compute a meaningful control action. To build a complete rule system for a controller is feasible, though it would not be feasible in areas such as medicine where we are dealing with a number of unknown variables. In this case of fuzzy systems, namely a controller for a physical device, the variables have approximate (i.e. fuzzy) values but are known.

In a number of cases, completeness is achieved by having the input (sensed) variables assuming all possible values and the rules considering all the combinations. This is achieved by fuzzy controllers by abstracting the possible values in linguistic variables such as ‘‘high,’’ ‘‘low,’’ etc. A fuzzy system with, for example three input variables, each of them with three values (min, med, max) would require $3^3$ rules to describe all possible process conditions.

In our particular controller there are $(5 \times 4)$ process conditions, given that we have two process variables (water level and water input rate) with five and four possible values, respectively. A twenty rule system will cover the system, if there are no duplicate rules.

For our system, a rule is a triple of the form:

<water-level, water-input-rate> → control-action

Our controller is specified by the following rules:

| | | | | | |
|---|---|---|---|---|---|
| <full, no-flow> | → | wide-open; | <full, min-flow> | → | wide-open; |
| <full, med-flow> | → | wide-open; | <full, max-flow> | → | wide-open; |
| <high-level, no-flow> | → | med-open; | <high-level, min-flow> | → | med-open; |
| <high-level, med-flow> | → | max-open; | <high-level, max-flow> | → | wide-open; |
| <med-level, no-flow> | → | closed; | <med-level, min-flow> | → | min-open; |
| <med-level, med-flow> | → | med-open; | <med-level, max-flow> | → | max-open; |
| <low-level, no-flow> | → | closed; | <low-level, min-flow> | → | closed; |
| <low-level, med-flow> | → | min-open; | <low-level, max-flow> | → | min-open; |
| <empty, no-flow> | → | closed; | <empty, min-flow> | → | closed; |
| <empty, med-flow> | → | closed; | <empty, max-flow> | → | closed. |

In the previous section, we illustrated how we derive a collection of the fuzzy process states from crisp inputs. We must now determine to what extent each of our rules fires, given these fuzzy states. To do this, we establish a weighting factor (firing strength) of each rule and then combine the rules according to this weighting factor to determine a final crisp action. Various approaches to this problem are discussed in [13].

To determine the firing strength of each rule we use a method derived independently by Mamdani [15] and Togai [19]. This uses the multivalued logical-implication operator: *truth {a(i) -> b(j)} = min {a(i), b(j)}*. For each rule in our database we determine the activation degree of the consequent—the recommended control action—as follows. The weighting or ''activation value'' $w(i)$ of the $i^{th}$ rule's consequent is the *minimum* of its antecedents' values. For example, suppose we have the following rule:

<low-level, med-flow> → min-open.

Suppose also that the determined degree of *low*-ness of the water-level is 14 and the degree of *medium*-ness of water input is 20. In our formulation, this is represented by the fuzzy state

{(low-level 14), (med-flow 20)}

The consequent of this rule, namely `MIN-OPEN`, will be assigned a strength of *min*(14, 20) = 14, and the firing strength of the recommended control action—set the drain at *minimally open*—will have a strength of 14.

There are several methods to combine the different fuzzy control actions considering the corresponding strengths with which each was recommended. We use the de-fuzzification method known as the Fuzzy Centroid Method [11].

Notice that our crisp input values have been fuzzified into a collection of fuzzy states such as `{(LOW-LEVEL 14), (MED-FLOW 20)}`. However, we have designed our rule set in such a way that there is exactly one rule with antecedent `<LOW-LEVEL, MED-FLOW>`, namely,

<low-level, med-flow> → min-close.

Thus we can easily select those rules that fire simply by inspecting our collection of fuzzy states.[3] Now, for the list of fuzzy states that we labeled (*) above, we extract the corresponding rules:

<empty, med-flow> → closed,
<empty, min-flow> → closed,
<low-level, med-flow> → min-open,
<low-level, min-flow> → closed.

From these rules, we assign to each consequent the minimum weight of its antecedents. This yields the list:

closed with degree 30,
closed with degree 20, (**)
closed with degree 14,
min-open with degree 14.

Now to select the final crisp action, we use the *fuzzy centroid* method on this list.

We define the regions for the control action, in this case *drain opening*, by a collection of triangles similar to those defining water input rate.

closed: (0, 1, 3)
min-open: (1, 15, 25)
med-open: (20, 30, 35)
max-open: (30, 40, 50)
wide-open: (45, 55, 65)

The centroid $C_{region}$ of a triangle is the ''peak'' of the triangle, e.g., 15 for the `MIN-OPEN` region and 40 for the `MAX-OPEN` region. The final actuator value is determined by the following formula. If we have a set of selected consequents, $\{region_i$ with degree $d_i\}$, we compute the final value as the floor of:

$$\frac{\Sigma(d_i \times C_{region_i})}{\Sigma(d_i)}$$

[3]Ostensibly, all rules fire. However, most rules have zero weight and will not affect the computation; these can be ignored.

To illustrate, for the collection of weighted consequents in (**) above, the final actuator value computed by the fuzzy controller is 3, computed from:

$$\frac{(30\times1)+(20\times1)+(14\times1)+(14\times15)}{(30+20+14+14)}$$

This number is interpreted as the crisp value for the drain setting recommended by the controller. In our formalization, this entire computation is encapsulated into a function called `FUZZY-CONTROLLER`.

## 4. Verifying the Fuzzy Controller

Given the formalization of the fuzzy controller in the preceding section, it remains to specify its correct behavior. Ideally, we would like to assert and prove that the execution of the controller over time maintains the water level of the tank within an acceptable range. To state this formally requires modeling the behavior of the tank under control of the system, i.e., modeling the *environment* in which the controller operates.

For our controller, the environment is particularly simple. The water input rate is measured at the pipe's opening which is above our tank. The water will reach the tank after a time unit. This kind of controller is known as a *read ahead controller* and it represents the simplest kind of controller to simulate. Given a current input rate and level, we can accurately predict the effect of the controller on the water level. In particular, we know that the change in level of the tank in a time unit is a very simple function `WATER-FILL` of the current level, water input rate, and water drain rate:

water-fill (current-level, water-in, water-out) ≡ (current-level + water-in) - water-out.

We encapsulate this in the function `WATER-FILL`. We can easily see that if the drain opening value is greater than the water input rate, then the water level in the tank will decrease; if the opening is smaller than the input rate, the level will increase. These are captured in the following three simple theorems.

water-out < water-in → level < water-fill (level, water-in, water-out);

water-in = water-out → water-fill (level, water-in, water-out) = level;

water-out ≥ water-in → level ≥ water-fill (level, water-in, water-out).

We formalize the simulation of our controller over time by functions in the Boyer-Moore logic. One function *water-fill-overtime* simulates the behavior of the level of water in the tank for an arbitrary sequence of water input rates. However, we are mainly interested in proving properties about the behavior of our system when water input rate is held constant. The function, *behavior-trace* carries out such simulation and returns a list of successive water level values for *n* time units, where the drain flow is adjusted at each time unit according to the (then) current water level.

behavior-trace (n, wl, rate)
≡
if n = 0, nil;
otherwise, cons (new-wl, behavior-trace (n - 1, new-wl, rate)),
where
new-wl = water-fill (wl, rate, fuzzy-controller (wl, rate)).

The function call *cons (e, l)* adds element *e* to the front of list *l*; *nil* is the empty list.

Asserting that our controller stabilizes the water level in the tank is equivalent to asserting that this behavior trace has certain characteristics. We discuss this further below.

Because functions in the Boyer-Moore logic are executable, we can ''run'' our `BEHAVIOR-TRACE` function for

specific values and observe the trace. Some sample runs are illustrated below:

behavior-trace (20, 1, 30) ⇒ [30 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45]

behavior-trace (20, 30, 45) ⇒ [60 65 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63]

behavior-trace (20, 97, 35) ⇒ [77 57 52 47 42 37 42 37 42 37 42 37 42 37 42 37 42 37 42 37]

### 4.1 From Behaviors to Theorems

In our simulations of the system using the BEHAVIOR-TRACE function and others, we observed various patterns in the system behavior and conjectured that the following were true.

- If the tank is initially full, the controller adjusts the drain in such a way that the water level decreases.
- If the tank is initially empty or nearly empty, then the water level tends to increase.
- If the water input rate remains constant for several cycles the water level either stabilizes at a single point within the desired medium range or cycles among several values within the desired range.

We generalized the observed behaviors in theorems in the logic stating precisely the conditions under which these behaviors occur.

1. If the water level is in the empty region, then the level rises *for all* input rates above 3. (Recall that a water input rate of 2 corresponds to no input.)
2. If the water level is within the full region, then the water level decreases *for all* possible values of water input rate.
3. Independently of the original water level, if the water input rate remains constant and within the ranges of 10 and 45, then the water level will either stabilize at a level within the desired boundaries, or the level will cycle among values within the desired boundaries.

We stated each of these conjectures within the Boyer-Moore logic and proved each of them using the theorem prover.

To illustrate, the third theorem is expressed in the logic as:[4]

```
(prove-lemma controller-cycles (rewrite)
  (let ((trace (behavior-trace 20 wl wir)))
     (implies (and (levelp wl)
                   (in-intervalp wir (cons 10 45)))
              (and (cycle-after-np 7 trace)
                   (in-interval-lstp
                      (some-cyclep (nthcdr 7 trace))
                      (cons 30 70))))))
```

The hypotheses of this theorem assert that the current water level has a value in the appropriate range ([0…100]) and water input rate is in the medium range of ([10…45]). The conclusion of the theorem is a conjunction of two properties. The first property asserts that the trace enters a cycle within at most 7 ‘‘ticks’’ of the clock. The second asserts that each of the elements of this cycle are within the desired medium water level range of 30 to 70.

Strictly speaking, the theorem applies only to behavior traces of length 20. However, we have also proved that once a cycle begins it repeats indefinitely. Therefore, the theorem generalizes to traces greater than 20; we have not yet formally proved this generalization. We strongly suspect that some smaller number of ticks would suffice to exhibit cycles in the traces. We also believe that the tank behavior actually always stabilizes in less than 7 ticks. However, we have not yet attempted to determine the minimum value. You will notice that all of the test traces exhibited at the end of the previous subsection satisfy the conclusion of our theorem.

[4] See Appendix B for the full formalization.

## 5. Conclusions and Future Directions

We have modeled a fuzzy controller within a formal logic and proved several properties of the system. The controller is a quite simple and the proofs were carried out mainly by ‘‘brute force.’’ However, we believe that the exercise was a useful experiment in applying formal methods to a new and important area. The behavior of controllers in general, and fuzzy controllers in particular, is subtle. Stating formally and proving rigorously behavioral properties of such controllers is one way to gain significantly enhanced assurance of their correctness. We believe that our work, though preliminary, points the way to an approach that can be applied to more realistic examples of fuzzy control.

We would like to extend this work in several ways.

- Proofs of the properties are currently quite inelegant and rely upon the simplicity of the controller. We envision developing a ‘‘reusable theory’’ [9] of fuzzy control that will allow us to prove more general and more broadly applicable theorems.
- The mechanical support in the prover for the underlying mathematics of fuzzy control is lacking. Rationals are supported in the Boyer-Moore logic, but real number arithmetic and analysis is not. Advanced work in this area may require using an alternative proof system. However, we hope to continue with the present tools and develop as much proof support as possible within our current proof paradigm.
- Our controller is unrealistically simple. We would like to formalize and prove properties of an existing fuzzy controller such as Sugeno’s predictive controller [18].

**Acknowledgements:** I want to acknowledge the work of the many researchers at Computational Logic who have contributed to the development of proof methodologies and strategies that have paved the way for the formalization and verification of systems in the Boyer-Moore logic. In particular, I appreciate the technical help of J S. Moore and Matt Kaufmann. I also want to acknowledge Warren A. Hunt, Jr., Michael K. Smith and Robert S. Boyer for their encouragement I received from them on this project. Very special thanks go to William D. Young who not only provided encouragement and technical help but also contributed in no small amount to the editing and revision of this work. I also would like to express my appreciation to Dr. Benito Fernandez of the University of Texas at Austin for his help on the topic of fuzzy systems.

# Appendix A
# The Boyer-Moore Logic and Theorem Prover

The Boyer-Moore logic [4, 6] is a quantifier-free, first-order predicate calculus with equality and induction. Logic formulas are written in a prefix-style, Lisp-like notation. Recursive functions may be defined and must be proven to terminate. The logic includes several built-in data types: Booleans, natural numbers, lists, literal atoms, and integers. Additional data types can be defined. The syntax, axioms, and rules of inference of the logic are given precisely in *A Computational Logic Handbook* [6].

The Boyer-Moore logic can be extended by the application of the following axiomatic acts: defining functions, adding recursively constructed data types, and adding arbitrary axioms. Adding an arbitrary formula as an axiom does not guarantee the soundness of the logic; we do not use this feature.

The Boyer-Moore theorem proving system (theorem prover) is a Common Lisp [17] program that provides a user with various commands to extend the logic and to prove theorems. A user enters theorem prover commands through the top-level Common Lisp interpreter. The theorem prover manages the axiom database, function and data type definitions, and proved theorems, thus allowing a user to concentrate on the less mundane aspects of proof development. The theorem prover contains a simplifier and rewriter and decision procedures for propositional logic and linear arithmetic. It also can perform structural inductions automatically.

The Boyer-Moore system also contains an interpreter for the logic that allows the evaluation of terms in the logic. Thus, the logic can be considered as either a functional programming language or an executable specification language. We often use this facility for debugging our specifications.

The Boyer-Moore prover has been used to check the proofs of some quite deep theorems. For example, some theorems from traditional mathematics that have been mechanically checked using the system include proofs of: the existence and uniqueness of prime factorizations; Gauss' law of quadratic reciprocity; the Church-Rosser theorem for lambda calculus; the infinite Ramsey theorem for the exponent 2 case; and Goedel's incompleteness theorem. Somewhat outside the range of traditional mathematics, the theorem prover has been used to check: the recursive unsolvability of the halting problem for Pure Lisp; the proof of invertibility of a widely used public key encryption algorithm; the correctness of metatheoretic simplifiers for the logic; the correctness of a simple real-time control algorithm; the optimality of a transformation for introducing concurrency into sorting networks; and a verified proof system for the Unity logic of concurrent processes. When connected to a specialized front-end for Fortran, the system has also proved the correctness of Fortran implementations of a fast string searching algorithm and a linear time majority vote algorithm. Recent work at CLI includes the mechanically checked proofs of a high-level language compiler, an assembler, a microprocessor, and a simple multi-tasking operating system. These verified components were integrated into a *vertically verified system* called the ''CLI Short Stack'' [2], the first such system of which we are aware. Many other interesting theorems have been proven as well. It is important to note that all of these proofs were checked by the same general purpose theorem prover, not a number of specialized routines optimized for specific problems.

# Appendix B
# The Fuzzy Controller Event List

This appendix contains the Boyer-Moore event list representing the specification and proof of the Fuzzy Controller. It does *not* contain the entire proof since it is built ‘‘on top of’’ a standard library of integer facts. The complete script is available on request.

# References

**1.** W.R. Bevier. "Kit and the Short Stack". *Journal of Automated Reasoning 5*, 4 (December 1989), 519-530.

**2.** W.R. Bevier, W.A. Hunt, Jr., J S. Moore, W.D. Young. "An Approach to Systems Verification". *Journal of Automated Reasoning 5*, 4 (December 1989), 411-428.

**3.** W.R. Bevier, W.D. Young. The Proof of Correctness of a Fault-Tolerant Circuit Design. Proceedings of the Second International Working Conference on Dependable Computing for Critical Applications, February, 1991, pp. 107-114.

**4.** R.S. Boyer and J S. Moore. *A Computational Logic.* Academic Press, Inc., ACM Monograph Series, Boston, 1979.

**5.** R. S. Boyer, M. W. Green and J S. Moore. The Use of a Formal Simulator to Verify a Simple Real Time Control Program. In *Beauty Is Our Business*, Springer-Verlag, 1990, pp. 55-64.

**6.** R.S. Boyer and J S. Mooer. *A Computational Logic Handbook.* Academic Press, Boston, 1988.

**7.** Bishop C. Brock, W.A. Hunt, Jr., and W.D. Young. Introduction to a Formally Defined Hardware Description Language. Proceedings of the IFIP Conference on Theorem Provers in Circuit Design, 1992, pp. 3-36.

**8.** D.M. Goldschlag. "Mechanically Verifying Concurrent Programs with the Boyer-Moore Prover". *IEEE Transactions on Software Engineering 16* (September 1990).

**9.** D.I. Good and W.D. Young. Mathematical Methods for Digital Systems Development. In *VDM '91: Formal Software Development Methods*, S. Prehn and W.J. Toetenal, Ed., Springer-Verlag Lecture Notes in Computer Science 552, 1991, pp. 406-430.

**10.** L.P. Holmblad and J.J. Ostergaard. Control of a Cement Kiln in Fuzzy Logic. In *Fuzzy Information and Decision Processes*, M. Gupta and E. Sanchez, Ed., North-Holland, Amsterdam, 1982.

**11.** B. Kosko. *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence.* Prentice-Hall, Englewood Cliffs, N.J., 1992.

**12.** G. Langari and M. Tomizuka. Fuzzy Linguistic Model Based Control. Technical Report CFL-90-004, Texas A&M University Center for Fuzzy Logic and Intelligent Systems Research, September, 1990.

**13.** C.C. Lee. "Fuzzy Logic in Control Systems: Fuzzy Logic Controller-Part I)". *IEEE Transactions on Systems, Man, and Cybernetics 20*, 2 (March-April 1990), .

**14.** C. Lengauer, C.H. Huang. A Mechanically Certified Theorem about Optimal Concurrency of Sorting Networks. Proceedings 13th ACM POPL, 1986, pp. 307-317.

**15.** E.H. Mamdani. "Application of Fuzzy Logic to Approximate Reasoning Using Linguistic Synthesis". *IEEE Transactions on Computers C-26*, 12 (December 1977), 1182-1191.

**16.** J S. Moore. "A Mechanically Verified Language Implementation". *Journal of Automated Reasoning 5*, 4 (December 1989), 493-518.

**17.** G.L. Steele, Jr. *Common LISP: The Language.* Digital Press, 1984.

**18.** M. Sugeno, editor. *Industrial Applications of Fuzzy Control.* North-Holland, Amsterdam, 1985.

**19.** ?. Togai and H. Watanabe. "Expert System on a Chip: An Engine for Realtime Approximate Reasoning". *IEEE Expert 1*, 3 (1986).

**20.** W.D. Young. "A Mechanically Verified Code Generator". *Journal of Automated Reasoning 5*, 4 (December 1989), 493-518.

# Table of Contents

1. Introduction . . . . . 1
2. Fuzzy Control . . . . . 1
3. A Simple Fuzzy Controller . . . . . 3
3.1. Linguistic Variables and Membership Functions . . . . . 3
3.2. Controller Rules . . . . . 5
4. Verifying the Fuzzy Controller . . . . . 7
4.1. From Behaviors to Theorems . . . . . 8
5. Conclusions and Future Directions . . . . . 9

Appendix A. The Boyer-Moore Logic and Theorem Prover . . . . . 10

Appendix B. The Fuzzy Controller Event List . . . . . 11

List of Figures

**Figure 1:** Fuzzy Numbers 4