Fuzzy Expert Systems Using CLIPS

Thach C. Le
The Aerospace Corporation
Los Angeles, California

Abstract

This paper describes a CLIPS-based fuzzy expert system development environment called FCLIPS and illustrates its application to the simulated cart-pole balancing problem. FCLIPS is a straightforward extension of CLIPS without any alteration to the CLIPS internal structures. It makes use of the object-oriented and module features in CLIPS version 6.0 for the implementation of fuzzy logic concepts. Systems of varying degrees of mixed Boolean and fuzzy rules can be implemented in FCLIPS. Design and implementation issues of FCLIPS will also be discussed.

I. Introduction

Production systems, better known as data-driven rule-based expert systems, attempt to encode human problem-solving knowledge. The architecture of production systems originated from research in cognitive psychology which observed that human behavior tends to be modularized and reactive in nature [1]. Consequently, the production system architecture was designed to allow for modularized and independent pieces of knowledge to be encoded in the form of if-then rules or production rules. One of the first implementations of a production system language is OPS5 (Official Production Systems) which is based on LISP. The Software Technologies Group of NASA at Houston, Texas later reimplemented a similar language called CLIPS (C-Language Integrated Production System) in C mainly to increase speed and portability. Because of CLIPS' flexibility and source code availability, it quickly became popular among expert system developers. Since its introduction several years ago, its has been frequently enhanced to embrace the latest advances in software technology.

Fuzzy systems, based on fuzzy set theory, also seek to encode human problem-solving knowledge. One of the most important contribution of fuzzy logic to the field of heuristic systems is the introduction of the concept of linguistic variables [5]. Linguistic variables are linguistic terms that are used to represent qualities of objects or concepts as in natural languages. In fuzzy systems, linguistic variables are represented by a mathematical function from which a value can be derived to indicate the similarity of an object to the ideal form. The implication is rather significant: instead of modeling the real world as a discrete world, linguistic variables allow a whole spectrum of continuous possibility to be modeled. For example, instead of speaking about a world of discrete terms such as circles, squares, and triangles, one can use linguistic variables to describe a larger set of the objects in terms of the degree to which an object is similar to circles, squares or triangles. In addition to linguistic variables, there is a set of operations in a fuzzy system. In the last few years, fuzzy systems have proven their effectiveness in solving difficult problems, particularly in the area of control [5]. Several commercial hardware and software tools have been introduced recently to build fuzzy systems.

The similar and complementary nature of fuzzy systems and expert systems have naturally suggested the idea of fuzzy expert systems in which the traditional Boolean production system architecture is extended to include the concepts of fuzzy systems. By combining the proven techniques of production systems with the fuzzy systems' representation and manipulation power, we believe the resulting system can effectively address even more challenging AI problems. Since the CLIPS source code is available at almost no cost, there have been attempts to extend CLIPS to a fuzzy expert system development environment. The most notable examples are the FuzzyCLIPS from the Knowledge Systems Laboratory of the National Research Council of Canada [6], and, also using the same name, the FuzzyCLIPS by Togai Infralogics Inc. [7]. In this paper, we will focus on our own implementation. The similarities and differences of these CLIPS-based systems will be discussed at a later time. An important quality of FCLIPS is the simplicity of its design, while still maintaining the ability to support a wide range of fuzzy expert systems.

In the following sections, we will describe the architecture of a typical Boolean production system exemplified by CLIPS, the architecture of a fuzzy system, the design and implementation issues of FCLIPS, the FCLIPS application to the cart-pole balancing problem, and future FCLIPS extensions.

II. Production Systems

Production systems are based on the principles of modularity and contextual dependent reaction in human behavior. A typical production system has three distinct modules, a *production memory* for containing production rules, a *working memory* for facts, and an *inference engine* for execution [1]. The production memory contains rules which have the form of:

if <antecedent> then <consequent>

An example of a CLIPS rule is as follows:

```
(defrule My_House
        (the house number is 123)
        (the street name is Pennsylvania)
=>
        (printout t "It is my house"))
```

Antecedent, also called the rule's left hand side (LHS), contains Boolean predicates that state the conditions for the actions to take place as specified in the consequent, or the right hand side (RHS). A rule becomes a candidate for execution if its antecedent is satisfied by facts in the working memory.

The working memory contains facts which describe the current state of a situation or a problem. The set of facts in the working memory is often changed as new data arrives or as the result of a rule being executed at each inference cycle.

The inference engine decides which rule is to be executed given the current set of facts in the working memory. It first performs matching between rules and existing facts. In CLIPS this matching is implemented using Rete net [2], known as the best pattern matching algorithm. After matching, there may be more than one matched rule. The set of matched rules is called the *conflict set.* Only one rule from the conflict set will be selected to be executed via the process called *conflict resolution.* Once a rule is selected, its RHS is

executed which often affects the working memory by adding, modifying or deleting facts from the working memory. As the result of the working memory being changed, a new conflict set is produced. The cycle goes on until the conflict set is empty or a halt instruction is encountered.

The current version of CLIPS provides several powerful features to the basic architecture of production systems. It has a built-in object-oriented system which greatly enhances its representation capability. Instead of using attribute-value predicates as in OPS5 to model problems, one can now use classes and objects, which are more natural and expressive.

Another important feature of CLIPS is modularity [8]. Rules that share some commonality can be grouped into a module. Modules allow the implementation of context-switching. At any time, only one module can be active and only its rules participate in the matching and selection process. The concept of modularity allows more efficient organization of the production memory and improves the scalability of a production system in terms of its scope and speed [3].

In general, production systems are quite effective in providing heuristic solutions to problems. They have been used in many real-world problems [5]. But while object-orientation greatly improved the representation capability of production systems, the rule predicates are still constrained to only Boolean statements which are not always natural to model real-world problems. This is an area where fuzzy systems offer an attractive alternative.

III. Fuzzy Systems

Fuzzy systems are based on fuzzy set theory [4]. They are similar to Boolean production systems in their use of if-then rules. However, they are different in two major aspects. First, the LHS and RHS predicates of a fuzzy rule are fuzzy statements, not Boolean statements. Secondly, fuzzy rules whose RHS have linguistic variables describing the same subject, are all executed and collectively contribute to the final conclusion. In a Boolean production system, rules compete to be solely executed. Example 1 shows fuzzy expert system rules:

```
Rule_1:   if    Temperature is Hot
                Pressure is High
          then
                set Change to Large

Rule_2:   if    Temperature is Cold
                Pressure is Medium
          then
                set Change to Small
```

Example 1: Fuzzy rules for a fuzzy system

The terms "Hot", "Cold", "High", "Medium", "Large" and "Small" are linguistic variables. A linguistic variable describes the quality of a concept or an object in term of *degree of truth*. The degree of truth is defined by a *membership function* that maps the domain of a concept into a degree of truth between 0 and 1. Figure 1 shows examples of membership functions of Temperature linguistic variables.

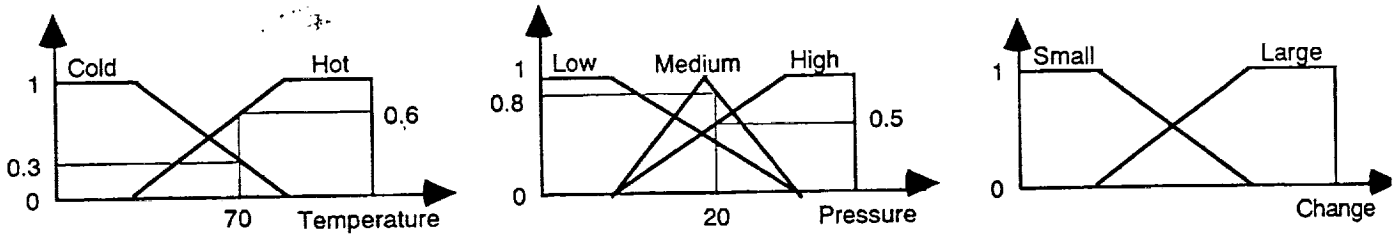

Figure 1: Membership functions for Temperature, Pressure and Change

Given a temperature value of 70, and a pressure value of 20, the statement "Temperature is Hot" in Rule 1 is evaluated to a truth value of .6. Similarly, "Temperature is Cold" is evaluated to .3, "Pressure is High" to .5, and "Pressure is Medium" to .8.

The typical method of evaluating a rule is by taking the minimum of the truth values of the LHS of the rule and applying an alpha-cut [5] to its RHS. The evaluations of Rule 1 and Rule 2 are shown in Figure 2. The results from both rules are combined using the *maximum envelop operation* and then defuzzified to give the final answer. There are several methods for defuzzification such as the centroid or the mean-of-max methods [5]. Figure 2 shows the defuzzification of the variable Change using the centroid method.

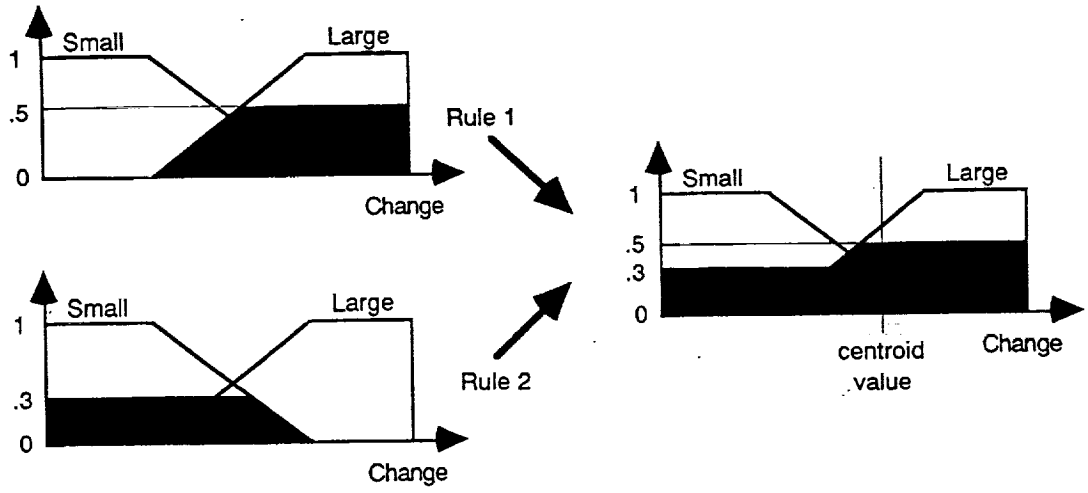

Figure 2: MIN-MAX operation and defuzzification using centroid method

As new values of Temperature and Pressure become available, the evaluation cycle repeats and a new value for Change is generated.

Fuzzy system technology has been applied successfully to many difficult control problems. It offers a straightforward way to encode human experience of solving real control problems into machine automation. The features of linguistic variables and rule contribution from fuzzy systems have proven to be very effective in modeling and solving many problems for which the continuity of variables involved is critical.

There have been several commercial tools developed to facilitate the implementation of fuzzy systems. However, they often allow only rules of common consequent and therefore, small in number. This small and monolithic characteristic of current fuzzy system implementations prevents them from addressing problems that require deep reasoning such as those in the areas of diagnosis, planning, or design.

Combining the best features of fuzzy systems and Boolean production systems will create a general system that can address more effectively a greater range of problem domains. Such systems are called fuzzy expert systems. In the following sections we will describe our fuzzy system extensions to CLIPS, including design and implementation issues.

IV. Implementation of Fuzzy Expert Systems in CLIPS

Our goal is to simply extend CLIPS to allow linguistic variables to be used in rules and to provide the necessary fuzzy operations to support them. The addition of support for linguistic variables in CLIPS is straightforward and does not pose any conflict with the original philosophy of production systems. However, the conflicting approaches of rule competition in Boolean production systems and rule cooperation in fuzzy systems needs to be resolved in a way that does not violate the original philosophies of either technique. Our approach to the CLIPS extensions is to preserve the integrity of CLIPS as much as possible and to only introduce new constructs when necessary.

Linguistic Variable and Operations

A linguistic variable is defined by a membership function. We implement a membership function as a CLIPS class called MFunc. A membership function can then be defined as an object instance of the class MFunc. Its value is specified as a piece-wise linear function which is a sequence of (x,y) coordinate pairs in the form (x1 y1 x2 y2 .... xn yn). For example, the Hot function is specified as follows:

```
(make-instance Hot of MFunc (Func 0 0 10 1 50 0))
```

Once the linguistic variable is defined, a value, for example, 8, can be tested against the membership function to give the corresponding degree of truth, or truth value, using the function "deg".

```
(deg 8 Hot)
```

The above statement will return a truth value of .8. Another very useful function is the Boolean function "is" which checks out if a value is within the domain of a membership function. It returns true if the value is within the domain and false otherwise. For examples,

(is 8 Hot) returns true, and (is -100 Hot) returns false.

In addition to the class MFunc, a fuzzy variable is represented by the class FVar. For example, "Change", an object instance of the class FVar, is first defined as follows:

```
(make-instance Change of FVar)
```

A fuzzy variable can have a value of nil or of a piece-wise linear function. This value can be combined in the MIN-MAX fashion with another linear piece-wise function to yield a new value.

Using the function "is" in the LHS, the fuzzy rule Rule_1 in section III can be rewritten in the CLIPS defrule construct as follows:

```
(defrule Rule_1
        (Temperature ?x&:(is ?x Hot))
        (Pressure ?y&:(is ?y High))
=>
        (bind ?m (fmin ?x Hot ?y High))
        (set Change Large ?m))
```

Since the LHS only contains Boolean predicates, it fits into the CLIPS defrule construct. Truth values of Temperature and Pressure are not evaluated until the rule is executed. This is designed to avoid unnecessary evaluations during the rule matching stage. If the rule is chosen for execution, the MIN-MAX operation is performed as specified in the RHS.

The function "fmin" returns the *minimum* value of the truth values in the LHS. The function "set" then uses this value to perform an alpha-cut on the membership function Large, which is then combined with the current value of the fuzzy variable Change using the *maximum* envelop operation.

With a few new classes and functions, a fuzzy rule can now be written in CLIPS format. The next section discusses issues related to the interaction of multiple fuzzy rules.

Fuzzy Processing

When multiple fuzzy rules exist in the production memory, a fuzzy cycle has to be implemented so that the contributing results to fuzzy variables from each rule can be integrated into a single value. We achieve this goal in FCLIPS by having a fuzzy inference rule of lower priority or salience value which is executed at the end of each fuzzy cycle. The set of fuzzy rules in section III can be rewritten as follows:

```
(defmodule Fuzzy_Module

        (defrule Rule_1
                (Temperature ?x&:(is ?x Hot))
                (Pressure ?y&:(is ?y High))
        =>
                (bind ?m (fmin ?x Hot ?y High))
                (set Change Large ?m))

        (defrule Rule_2
                (Temperature ?x&:(is ?x Cold))
                (Pressure ?y&:(is ?y Medium))
        =>
                (bind ?m (fmin ?x Cold ?y Medium))
                (set Change Small ?m))

        (defrule Fuzzy_Engine
                (declare (salience -10))
```

```
        ?f1 <- (Temperature ?x)
        ?f2 <- (Pressure ?y)
=>
        (retract ?f1 ?f2)
        (<check stopping condition> ?x ?y)
        (<defuzzify, apply and reset> Change)
        (<get new values for Temperature & Pressure>))
)
```

Example 2: A CLIPS-based fuzzy module

During a fuzzy cycle, the fuzzy variable Change accumulates the results from rules executed. At the end of the cycle, the Fuzzy_Engine rule is executed. Change is typically defuzzified and the result is asserted into the working memory or sent to an external environment.

To avoid potential interference from other rules in the production memory, this set of rules can be conveniently grouped into a CLIPS module. With a few extra fuzzy related classes and functions, CLIPS is now capable of supporting traditional fuzzy systems as described in section III. There are no changes necessary to the conflict resolution or rule matching modules of CLIPS. Yet, a new kind of mixed systems is also now possible. In these systems, not only can the LHS of a rule have any number of mixed Boolean or fuzzy predicates, but the RHS can also. This is a major improvement from traditional fuzzy systems whose rules' RHS have to be about the same subject.

Of course, due to the differences in the nature of production systems and fuzzy systems, there are several issues that need to be addressed for systems that allow mixed rules: (1) how to prioritize mixed rules in conflict resolution, (2) how to group fuzzy rules of different RHS' into fuzzy modules, (3) how to differentiate fuzzy cycles and Boolean cycles, and (4) how to propagate fuzzy conclusions. To illustrate these issues more clearly, we will use the examples of the following rules. The assumed truth values of antecedents are given on the right.

```
                                                    truth values
rule 1:   if    (A > 10)                                1
                (B is Small)                            .3
          then
                (set C to Large)
                (set D to Medium)

rule 2:   if    (A is Large)                            .7
                (D is Small)                            .3
          then
                (set C to Medium)
                (set E to Small)

rule 3:   if    (A > 25)                                1
                (X > 25)                                1
          then
                (set E to Large)
```

Example 3: Mixed rules with different truth values

For the first issue, based on the truth values alone, how should these rules be prioritized? We argue that they all should have the same priority despite the fact that the truth values for some fuzzy predicates are less than others. The fuzzy statement "B is Small" with truth value of .3 is no less significant than "A is Large" with truth value .7, because truth values in fuzzy logic are not the same as uncertainty values which have been used as a criteria for conflict resolution in some systems. "B is Small" with truth value .3 is a description with complete certainty of how small B is, which is indicated by the truth value of .3. Besides, in traditional fuzzy systems, a small truth value of a fuzzy predicate in a rule does not affect the rule's conclusion validity, but only the actual value of the conclusion. Consequently as long as a fuzzy predicate has a truth value of greater than 0 in FCLIPS, it is considered equivalent to a Boolean statement of value true. This interpretation is particularly significant in that it does not require extra processing for mixed rules during the rule matching stage.

The second issue points to the fact that traditional fuzzy systems only allow rules that address the same subject on their RHS. In the example above, there is no obvious way to group these three rules into fuzzy modules since their RHS' are not exactly about the same subjects. Rule 1's RHS is about C and D, rule 2 C and E, and rule 3 E. It turns out that we don't have to group them, as the following discussion will show.

The third issue concerns the interpretation of the significance of a fuzzy rule, a Boolean rule and a mixed rule. This interpretation is important for determining how conflict resolution should be changed. There are two interpretations for the significance of fuzzy rules and fuzzy modules versus a Boolean rule. In the first interpretation, a fuzzy module is considered equivalent to a Boolean rule in a Boolean production system, since all the rules in a fuzzy module can be thought of as being executed in parallel and contributing to a single conclusion. This interpretation is valid and in common use [7]. It can already be implemented in FCLIPS by using the existing defmodule construct to group fuzzy rules of the same module, as shown in Example 2.

In the second interpretation, all three kinds of rules are equivalent and compete for execution on the same grounds. This interpretation is intuitive and less restrictive, but it poses a question about when a fuzzy cycle starts and stops, i. e. when is a fuzzy variable defuzzified? In FCLIPS, this question of fuzzy cycle is resolved by introducing rules of higher salience value specifying defuzzifying conditions for each fuzzy variable mentioned in the rules' RHS. An example of a defuzzifying rule for the fuzzy variable C in Example 3 is as follows:

```
Rule DefuzzifyingC:
        if      (declare  (salience 10))
                (TimeStamp > 10 )
                (C not equal nil)
        then
                (<defuzzifying, apply and reset> C)
```

There are similar defuzzification rules for D and E. The second issue concerning how to group rules into fuzzy modules is no longer an issue, because there is no fuzzy module, only defuzzifying conditions for fuzzy variables in a rule consequent. Both interpretations are implementable under FCLIPS. The first is more traditional and of common use. The second is more general and interesting.

And finally, the fourth issue concerns the multiple-step reasoning where conclusions reached remain in fuzzy form to match the rules in the following cycles. By avoiding

defuzzification in intermediate reasoning steps, less information will be lost. Given the following two rules,

Rule 1:	if	(A is Small)
	then	(B is Large)
Rule 2:	if	(B is Small)
	then	(C is Medium)

assume that Rule 1 is executed and generates a fuzzy value X for "B is large". The truth value for "B is Small" in rule 2 will then be calculated as [15]:

truth value = 1 - MeanSquareError (X, Small)

This operation is not yet (but will be soon) implemented in FCLIPS. Currently, B has to be defuzzified after rule 1 execution before being matched into rule 2 antecedent.

In summary, FCLIPS is an extension to CLIPS to provide a fuzzy expert system development environment. Despite its simplicity, it is capable of supporting a wide range of fuzzy expert systems. An FCLIPS implementation has been fully prototyped and successfully tested in CLIPS code. It is currently being ported to C. In addition, a ToolTalk interprocess communication facility was added to FCLIPS to communicate with external processes. In the next section, an example using FCLIPS is described.

V. Example: The Cart-Pole Balancing Problem

The Cart-Pole Balancing problem, also called the Inverted-Pendulum problem, is a popular test case for fuzzy systems. In this section, we describe the structure and key components of a FCLIPS program that solves this problem. The problem is stated as follows:

A cart is moveable along 1 dimension with a pole on top as shown in Figure 3. Its state variables, which include the pole *angle*, the pole *angular velocity*, the current *position* and *velocity* of the cart, are known. Calculate the appropriate force to apply to the cart to keep the pole balanced.

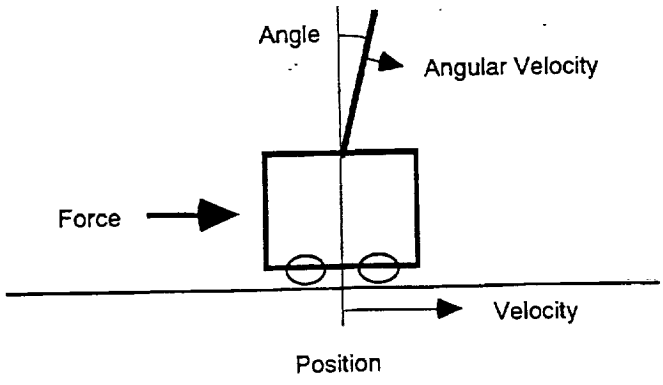

Figure 3: Cart-Pole Balancing problem

First, a function called "model" that simulates the dynamics of the cart-pole system is defined. It takes the applied force and the current state variables as arguments and generates a new set of state variables.

```
(deffunction model (?force ?polePosition ?poleVelocity
                                ?cartPosition ?cartVelocity) ...)
```

Six membership functions for the pole angle or pole position, A_NegMed, A_NegSmall, A_Zero, A_PosSmall, A_PosMed, and A_VerySmall, are defined:

```
(make-instance A_NegMed    of MFunc (Func -50 0 -20 1 -15 1 -10 0))
(make-instance A_NegSmall  of MFunc (Func -20 0 -10 1 0 0))
... (more make-instance's)
```

And similarly, for Angular Velocity: Av_NegSmall, Av_Zero, Av_PosSmall, and Av_VerySmall; for Distance: D_Neg, D_Zero, D_Pos, D_VerySmall; for Velocity: V_Neg, V_Zero, V_Pos, D_VerySmall; and for Force: F_NegMed, F_NegSmall, F_NegVerySmall, F_Zero, F_PosVerySmall, F_PosSmall, and F_PosMed.

The initial configuration of the cart-pole system is defined in the startup rule. The values can be changed for testing purposes.

```
(defrule startup
=>
        (assert (PolePosition 5))
        (assert (PoleVelocity 0))
        (assert (Position 0))
        (assert (Velocity 0)))
```

There are 9 rules devised to balance the pole position:

```
(defrule Rule1
        (PolePosition ?x&:(is ?x A_PosMed))
        (PoleVelocity ?y&:(is ?y Av_Zero))
=>
        (bind ?m (fmin ?x A_PosMed ?y Av_Zero))
        (set Force F_PosMed ?m))

(defrule Rule2
        (PolePosition ?x&:(is ?x A_PosSmall))
        (PoleVelocity ?y&:(is ?y Av_PosSmall))
=>
        (bind ?m (fmin ?x A_PosSmall ?y Av_PosSmall))
        (set Force F_PosSmall ?m))

... ( more defrules' )
```

And one deffuzzification rule:

```
(defrule engine
        (declare (salience -10))
        ?f1 <- (PolePosition ?x)
        ?f2 <- (PoleVelocity ?y)
        ?f3 <- (Position ?z)
```

```
        ?f4 <- (Velocity ?w)
=>
        (retract ?f1 ?f2 ?f3 ?f4)                          ; delete old facts
        (bind ?torq (centroid (send [Force] get-Val)))     ; defuzzification
        (bind ?vals (model ?torq ?x ?y ?x ?w))             ; new values
        (assert (PolePosition (nth$ 1 ?vals)))             ; create new facts
        (assert (PoleVelocity (nth$ 2 ?vals)))
        (assert (Position (nth$ 3 ?vals)))
        (assert (Velocity (nth$ 4 ?vals))))
```

The function "centroid" performs defuzzification using the centroid method [5]. The new values can be sent to graphs, files or simulation. In our testing, the results were sent via ToolTalk to a separate graphic simulation process for displaying the cart-pole system behavior. After some tuning of the membership functions, the system was able to balance the pole.

Conclusions

FCLIPS is a simple and straight-forward implementation of a fuzzy expert system development environment based on CLIPS. The CLIPS object-oriented and modularity features were exploited to implement fuzzy logic concepts into CLIPS. Despite the simplicity of FCLIPS, it can be used to develop systems containing a wide range of mixed Boolean and fuzzy rules. With FCLIPS, fuzzy expert system solutions can be introduced to problems which typically require deep reasoning such as those in the areas of diagnosis, planning, or design.

Bibliography

[1] Brownston, L., Farrell, R., Kant, E., and Martin, N., Programming Expert Systems in OPS5, Addison Wesley Publishing Company, 1985.

[2] Forgy, C., "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," Artificial Intelligence, vol. 19, 1982, p. 17-37.

[3] Le, T., Homeier, P., "Portable Inference Engine: An Extended CLIPS for Real-time Production Systems", Space Operations Automation and Robotics (SOAR '88), Dayton OH, July 20-23, 1988, Proceeding p. 187-192.

[4] Graham, I. and Jones, P., *Expert Systems - Knowledge, Uncertainty and Decision*, Chapman and Hall, 1988.

[5] Bezdek, J.C., Ruspini, E., Zadeh, L.A., et al, *Fuzzy Logic Inference Systems*, Short Course Notes, Intelligent Inference Systems Corp., 1993.

[6] *FuzzyCLIPS Version 6.02 User's Guide,* Knowledge Systems Laboratory, Institute for Information Technology, National Research Council of Canada. May 1994.

[7] Teichrow, S. J., Horstkotte, R. E., Fuzzy-CLIPS, The C Language Integrated Production System with Fuzzy Logic Capability. NAS9-18335, August 17, 1990.

[8] CLIPS Reference Manual, CLIPS Version 6.0, Software Technology Branch, Lyndon B. Johnson Space Center, June 2nd, 1993.