FuzzyCLIPS Version 6.10d

User's Guide

Integrated Reasoning Group
Institute for Information Technology
National Research Council Canada

Bob Orchard

October 2004

Abstract

FuzzyCLIPS[1]is an extended version of the CLIPS rule-based shell[2] for representing and manipulating fuzzy facts and rules. In addition to the CLIPS functionality, FuzzyCLIPS can deal with exact, fuzzy (or inexact), and combined reasoning, allowing fuzzy and normal terms to be freely mixed in the rules and facts of an expert system. The system uses two basic inexact concepts, fuzziness and uncertainty.

Résumé

FuzzyCLIPS[3] est une version améliorée du système expert vide à base de règles.[4] CLIPS utilisé pour la représentation et la manipulation de faits et de règles flous. En plus de pouvoir executer les fonctions de CLIPS, FuzzyCLIPS peut traiter le raisonnement exact, le raisonnement flou ou inexact et le raisonnement mixte permettant aux termes flous et ordinaires d'être librement intégrés aux règles et aux faits d'un système expert. Le système utilise deux concepts de base inexacts, le flou et l'incertitude.

1 Compatible with CLIPS version 6.10

2 CLIPS was developed by the Artificial Intelligence Section, Lyndon B. Johnson Space Center, NASA.

3 Compatible avec la version 6.10 de CLIPS.

4 CLIPS a été mis au point par la Artificial Intelligence Section, Lyndon B. Johnson Space Center, NASA.

Table of contents

Table of Figures

1 Introduction

This report describes a project carried out at the National Research Council of Canada (NRC) to implement a fuzzy expert system shell on top of CLIPS [1], [2]. This extended version of CLIPS is called FuzzyCLIPS. The modifications made to CLIPS contain the capability of handling fuzzy concepts and reasoning. It enables domain experts to express rules using their own fuzzy terms. FuzzyCLIPS allows any mix of fuzzy and normal terms, numeric-comparison logic controls, and uncertainties in the rules and facts. Fuzzy sets and relations deal with fuzziness in approximate reasoning, while certainty factors for rules and facts manipulate the uncertainty. The use of the above modifications are optional and existing CLIPS programs still execute correctly. Section 2 describes restrictions and conditions of use for FuzzyCLIPS. Section 3 gives information on how to install FuzzyCLIPS at a site. Section 5 describes the changes made to CLIPS to implement fuzzy expert systems and gives some theoretical background on fuzzy logic. Section 6 describes how to use the changes made at the NRC. Section 7 discusses the further changes made to CLIPS to accommodate the needs of continuously operating systems. Section 8 addresses the functionality of CLIPS within FuzzyCLIPS. Section 9 briefly discusses some of the limitations of FuzzyCLIPS and what future work could be done.

Please note that we also have a Java solution, the FuzzyJ Toolkit, for building fuzzy systems. It provides a mechanism for creating simple fuzzy rule based systems and can be used in conjunction with Jess, the expert system shell from Sandia National Labs (using the additional FuzzyJ component called FuzzyJess). For further information please see:

http://ai.iit.nrc.ca/IR_public/fuzzy/fuzzyJToolkit.html

2 Licence for NRC Software

This software is provided by The National Research Council of Canada (called NRC) whose address for communications with respect to this software is

National Research Council Canada
Institute for Information Technology
Integrated Reasoning Group
1200 Montreal Road
Ottawa, Ontario, Canada K1A 0R6
Electronic mail: bob.orchard@nrc-cnrc.gc.ca

2.1 Title and Conditions

NRC provides a fully paid up and nonexclusive licence to the software package with the following conditions:

1. The software will be used for educational and research purposes only.
2. The licence does not include the right to sublicense the software or to make it available for independent use by third parties outside the recipient organization.
3. Copies of the software may be made for use within the recipient organization; however, copyright remains with NRC.
4. All publications arising from the use of the software shall duly acknowledge such use in accordance with normal practices followed in scientific research publications.
5. The software is provided in its current state and NRC assumes no obligation to provide services, for example, maintenance or updates.
6. All users are requested to provide their name, the name of their organization, and a mailing or e-mail address so that we may track the use of the software as well as provide information to users as updates and enhancements are made.

2.2 Record of Use

Users are requested to inform NRC of any corrections, changes, or extensions to the software. NRC would also appreciate being informed of noteworthy uses.

2.3 Value of the Software

This software is considered to have no market value.

2.4 Warranty

NRC disclaims any warranties, expressed, implied, or statutory, of any kind or nature with respect to the software, including without limitation any warranty of fitness for a particular purpose. NRC shall not be liable in any event for damages, whether direct or indirect, special or general, consequential or incidental, arising from the use of the software.

2.5 Commercial Uses

Commercial use of FuzzyCLIPS is possible. Contact the Integrated Reasoning Group of the Institute for Information Technology at NRC (bob.orchard@nrc-cnrc.gc.ca) for details.

3 Installation Information

FuzzyCLIPS has been successfully compiled and tested on the following operating systems:

Windows 2000, NT (compiled with Borland C++ 4.51 and Microsoft VC++ 6.0)
HP-UX
VAX VMS
SUN Solaris

3.1 Accessing FuzzyCLIPS

FuzzyCLIPS is available via anonymous ftp from ai.iit.nrc.ca or via the World Wide Web (WWW) using the URL designation http://iit-iti.nrc-cnrc.gc.ca/projects-projets/fuzzyclips_e.html.

To access FuzzyCLIPS, follow the links to the FuzzyCLIPS web site. There you need to 'tell us who you are', providing an organization name, your name and an email address. You will then be able to select the version of FuzzyCLIPS that you want or other supporting files for download.

For a windows version the file **fzclp610dwin.zip** should be downloaded and uncompressed (extract files) preserving folder names. Then you will have the following directory structure (or something similar):

```
fzclp610d/
        Docs/                   - documentation (pdf and/or doc files) for FuzzyCLIPS
        FZEXMPL/                - the fuzzy example programs
        pc-prjct/               - project directories
          borland/              - project files + FuzzyCLIPS executables for Borland 4.5 build
            clips.hlp           - help file used by clipscmd.exe and clipswin.exe
            CLIPScmd.exe        - FuzzyCLIPS with non-graphical interface
            Clipscmd.ide        - Borland project file for clipscmd version
            clipscmdNOTE.txt    - a note about creating the command line version
            CLIPSedt.exe        - clips editor program
            Clipsedt.ide        - Borland project file for a CLIPS editor
            CLIPSwin.exe        - FuzzyCLIPS with graphical interface
            Clipswin.ide        - Borland project file for clipswin version
          editor/               - source files for the CLIPS editor program
          interface/            - source files for the CLIPS graphical interface
          VC++/                 - project files + FuzzyCLIPS executables for VC++ 6.0 build
            Fzclips/            - FuzzyCLIPS workspace for VC++
              Fzclipscmd/       - VC++ project for building command line version
                Fzclipscmd.dsp  - VC++ project file
                Release/        - holds the object files and the fzclipscmd.exe files
              Fzclipswin/       - VC++ project for building windows GUI version
                Fzclipswin.dsp  - VC++ project file
                Release/        - holds the object files and the fzclipswin.exe files
              Fzclipslib/       - VC++ project for building static library of FuzzyCLIPS
                Fzclipslib.dsp  - VC++ project file
                Release/        - holds the object files and the fzclipslib.lib files
        source/                 - source files (.c and .h) for FuzzyCLIPS
```

All of the executable files distributed with FuzzyCLIPS have been created to include fuzzy facts and reasoning, certainty factors, and runtime extensions. Any or all of these three components can be selectively removed from inclusion in the system by modifying the file setup.h and recompiling and linking FuzzyCLIPS. The following FLAGS in setup.h are set to 1 to include the feature and to 0 to exclude the feature:

```
FUZZY_DEFTEMPLATES      - for fuzzy facts and reasoning
CERTAINTY_FACTORS       - for certainty factors
EXTENDED_RUN_OPTIONS    - for extended runtime options
```

You must also set the correct flag to indicate which compiler is being used. Only one of these compiler flags must be set to 1 and all others must be set to 0. For example, when compiling with the Microsoft VC++ compiler one should have flags set something like:

```
/*************************************************************/
/* -------------------- COMPILER FLAGS ---------------------- */
/*************************************************************/

/*********************************************************************/
/* Flag denoting the environment in which the executable is to run.  */
/* Only one of these flags should be turned on (set to 1) at a time. */
/*********************************************************************/

#define GENERIC 0   /* Generic (any machine)                 */
#define VAX_VMS 0   /* VAX VMS                               */
#define UNIX_V  0   /* UNIX System V or 4.2bsd or HP Unix    */
#define UNIX_7  0   /* UNIX System III Version 7 or Sun Unix */
#define MAC_SC6 0   /* Macintosh, with Symantec C 6.0        */
#define MAC_SC7 0   /* Macintosh, with Symantec C 7.0        */
#define MAC_SC8 0   /* Macintosh, with Symantec C 8.0        */
#define MAC_MPW 0   /* Macintosh, with MPW 3.2 C             */
#define MAC_MCW 0   /* Macintosh, with Code Warrior 3.0      */
#define IBM_MCW 0   /* IBM PC, with CodeWarrior 3.0          */
#define IBM_ZTC 0   /* IBM PC, with Zortech C++ 3.1          */
#define IBM_MSC 1   /* IBM PC, with Microsoft C 6.0          */
#define IBM_TBC 0   /* IBM PC, with Borland C++ 5.0          */
#define IBM_ICB 0   /* IBM PC, with Intel C Code Builder 1.0 */
#define IBM_SC  0   /* IBM PC, with Symantec C++ 6.1         */
```

In the FZEXMPLE directory you will find some example programs:

lin1st.clp	(control of linear 1st order system)
lin1stdl.clp	(control of linear 1st order system with delay)
simpltst.clp	(basic non-sensical example)
fzcmplr.clp	(fuzzy compiler example)
fzcmpmod.clp	(fuzzy compiler example using MODULES)
shwrnoui.clp	(shower example with NO graphical user interface)

(there may also be some .dat files that are the outputs for these programs)

FuzzyCLIPS has been compiled (using Borland C++ 4.5, an ANSI compiler) and tested on a PC-compatible system running Windows 2000. It has been compiled for distribution as a 16-bit system that will run on windows 3.x systems. FuzzyCLIPS has also been compiled as a 32 bit version using VC++ 6.0. See the executables under the VC++ directory. There are also readme files in various locations that will help.

4 New Features in Recent Versions

4.1 Version 6.10d

During the past few years there have been minor bug fixes and an upgrade to version 6.10 of CLIPS (thanks to Dave Woodman). There are no plans at this time to become compatible with version 6.2 of CLIPS.

4.1.1 Added Ability to Turn Certainty Factor Calculations On and Off

This release, 6.10d, has addressed a problem that was felt to be significant enough to warrant a resolution and a new release. The problem is with certainty factors and the automatic way that they are calculated in a rule. Sometimes it is desirable to not do the automatic certainty factor calculation for all or a set of facts being asserted in a rule. Consider the following simple example:

```
(defglobal ?*count* = 4)

(defrule init
  =>
   (assert (counter ?*count*))
)

(defrule big
   (declare (CF 0.8))
   (counter ?c&:(> ?c 2))
  =>
   (assert (value BIG))
)

(defrule small
   (declare (CF 0.6))
   (counter ?c&:(<= ?c 2))
  =>
   (assert (value SMALL))
)

(defrule print-and-repeat
   (declare (salience -100))
   ?cf <- (counter ?c)
   ?vf <- (value ?v)
  =>
   (printout t "Count is " ?c " with certainty of " (get-cf ?cf) " and value is "
             ?v " with certainty of " (get-cf ?vf) t)
   (retract ?cf ?vf)
   (bind ?*count* (- ?*count* 1))
   (if (<= ?*count* 0)
    then (halt)
   )
   (assert (counter ?*count*))
 )
```

When executed this will give the, perhaps unexpected, results:

Count is 4 with certainty of 1.0 and value is BIG with certainty of 0.8
Count is 3 with certainty of 0.8 and value is BIG with certainty of 0.64
Count is 2 with certainty of 0.64 and value is SMALL with certainty of 0.384
Count is 1 with certainty of 0.384 and value is SMALL with certainty of 0.2304

Clearly the counter is not expected to have its certainty be anything other than 1. But in previous versions of FuzzyCLIPS there was no way to force this to be the case. Doing an `(assert (counter ?c) CF 1.0))` is the same as `(assert (counter ?c))`. Specifying the CF does not force the CF to be the value provided. Nor should it since the calculation of a certainty factor for a fact depends on the rule's certainty and the certainty of the matched facts in the rule. Also you'll note that the certainty of the `value` facts is not as expected either. On each iteration of the `print-and-repeat` rule the CF of the `counter` fact is becoming lower. This again is correct behaviour but not

what is wanted for this example. We need a way to disable the calculation of certainties for facts in a rule when necessary. To this end we added two new functions to control the certainty calculations when facts are asserted in rules. These are the `enable-rule-cf-calculation` and `disable-rule-cf-calculation`. One could for example turn off any rule certainty factor calculations by executing the `disable-rule-cf-calculation` before any rules are executed or in an initialization rule. By default rule cf calculations are enabled. Now in the simple example above we could make a simple change to `the print-and-repeat` rule as follows:

```
(defrule print-and-repeat
   (declare (salience -100))
   ?cf <- (counter ?c)
   ?vf <- (value ?v)
  =>
   (printout t "Count is " ?c " with certainty of " (get-cf ?cf) " and value is "
             ?v " with certainty of " (get-cf ?vf) t)
   (retract ?cf ?vf)
   (bind ?*count* (- ?*count* 1))
   (if (<= ?*count* 0)
    then (halt)
   )
   (disable-rule-cf-calculation) ;; turn off rule cf calculations
   (assert (counter ?*count*))
   (enable-rule-cf-calculation)  ;; turn back on again
)
```

Now when the program is run we get the following desired output:

Count is 4 with certainty of 1.0 and value is BIG with certainty of 0.8
Count is 3 with certainty of 1.0 and value is BIG with certainty of 0.8
Count is 2 with certainty of 1.0 and value is SMALL with certainty of 0.6
Count is 1 with certainty of 1.0 and value is SMALL with certainty of 0.6

It might be interesting to note that there could be better ways to deal with this problem. Perhaps, for example, each fact should be declared to have a CF or not to have one and the CF calculations then would only be applied to those facts with CFs. This would have required much more extensive changes, however, and might have impacted ALL existing FuzzyCLIPS programs that use certainty factors.

4.1.2 NOT patterns May Cause FuzzyCLIPS to Crash

Certain patterns in a rule with **NOT** conditions can cause FuzzyCLIPS to crash. In particular it was found that a NOT pattern with nested **AND** conditions will cause FuzzyCLIPS to run. For example the simple program:

```
(defrule crash_fuzzy
    (not (and (A)
              (B)
         )
    )
  =>
    (printout t "FuzzyCLIPS will crash now!" crlf)
    (assert (dummy fact1)) ; This assert will crash FuzzyCLIPS
)
```

will cause FuzzyCLIPS (prior to version 6.10d) to crash.

5 Fuzzy Expert Systems

In the real world there exists much fuzzy knowledge, i.e., knowledge that is vague, imprecise, uncertain, ambiguous, inexact, or probabilistic in nature. Human thinking and reasoning frequently involve fuzzy information, possibly originating from inherently inexact human concepts and matching of similar rather then identical experiences. In systems based upon classical set theory and two-valued logic, it is very difficult to answer some questions because they do not have completely true answers. Humans, however, can give satisfactory answers, which are probably true. Expert systems should not only give such answers but also describe their reality level. This level should be calculated using imprecision and the uncertainty of facts and rules that were applied. Expert systems should also be able to cope with unreliable and incomplete information and with different expert opinions. Many of today's commercial expert system building tools or shells use different approaches such as certainty factors [3] and Bayesian [4] and Dempster-Shafer's [5] models to handle uncertainty in the knowledge or data, but they cannot cope with fuzzy data, which constitute a very significant part of a natural language. Several systems such as Cadiag-2 [6], Fault [7], FLOPS [8], FRIL [9], SYSTEMZ-II [10], and FLISP [11] support some fuzzy reasoning, but they are purposely built from high-level languages for a specific domain of application. Following the main idea of CLIPS, which is to develop an expert system tool written in and fully integrated with the C language for high portability, low cost, and easy integration with external systems, this work was undertaken to extend CLIPS for representing and manipulating fuzzy facts and rules.

Fuzziness and uncertainty are the two distinct inexact concepts employed in the system. The following sections will discuss the general theory of both fuzziness and uncertainty, their implications on rule evaluation in FuzzyCLIPS, and algorithms implemented for extracting exact values from fuzzy facts.

5.1 Fuzziness

Fuzziness[5] occurs when the boundary of a piece of information is not clear-cut. For example, concepts such as *young*, *tall*, *good,* or *high* are fuzzy. There is no single quantitative value which defines the term *young*. For some people, age 25 is *young*, and for others, age 35 is *young*. In fact the concept *young* has no clean boundary. Age 1 is definitely *young* and age 100 is definitely not *young*; however, age 35 has some possibility of being *young* as well as some possibility of being *not young* and usually depends on the context in which it is being considered. The representation of this kind of information in FuzzyCLIPS is based on the concept of fuzzy set theory [14]. Unlike classical set theory where one deals with objects whose membership to a set can be clearly described, in fuzzy set theory membership of an element to a set can be partial, i.e., an element belongs to a set with a certain grade (possibility) of membership. More formally a fuzzy set A in a universe of discourse U is characterized by a membership function

$$\mu_A : U \rightarrow [0,1] \qquad (1)$$

which associates a number $\mu_A(x)$ in the interval [0,1] with each element x of U. This number represents the grade of membership of x in the fuzzy set A.

For example, the fuzzy term *young* might be defined by the fuzzy set in Table 1.

Table 1: Fuzzy Term *young*

Age	Grade of Membership
25	1.0
30	0.8
35	0.6
40	0.4
45	0.2
50	0.0

[5] For an excellent introduction to the concept of fuzziness see [12], Earl Cox's Fuzzy Systems Handbook.

Regarding equation (1), one can write

$$\mu_{young}(25) = 1, \mu_{young}(30) = 0.8, \ldots , \mu_{young}(50) = 0$$

Grade of membership values constitute a possibility distribution of the term *young*. The table can also be shown graphically (see Figure 1: Possibility distribution of *young*).

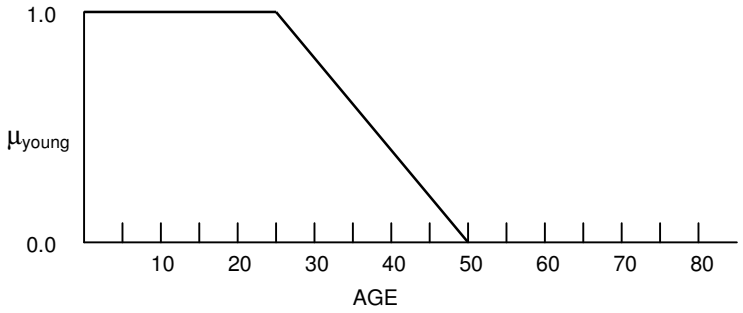


Figure 1: Possibility distribution of *young*

The possibility distribution of a fuzzy concept like *somewhat young* or *very young* can be obtained by applying arithmetic operations to the fuzzy set of the basic fuzzy term *young*, where the modifiers *somewhat* and *very* are associated with specific mathematical functions. For instance, the possibility values of each age in the fuzzy set representing the fuzzy concept *somewhat young* might be calculated by taking the square root of the corresponding possibility values in the fuzzy set of *young* (see Figure 2: Possibility distribution of *somewhat young*). These modifiers are often referred to as hedges. A more complete description of the hedges supplied with FuzzyCLIPS and how to add user-defined hedges are described in Section 5.3.

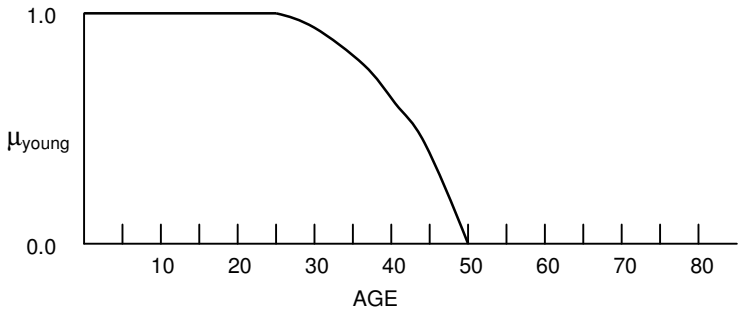


Figure 2: Possibility distribution of *somewhat young*

Fuzzy facts may be defined, matched as a pattern in a rule, and asserted in a manner similar to the ordinary *crisp* facts employed in standard CLIPS.

Example 1

```
(deftemplate age ;definition of fuzzy variable 'age'
   0 120 years
  ( (young (25 1) (50 0))
    (old   (50 0) (65 1))
  )
)
```

```
(deffacts fuzzy-fact
   (age young)                    ; a fuzzy fact
)

(defrule one ; a rule that matches and asserts fuzzy facts
     (Speed_error big)
 =>
     (assert (Throttle_change small))
)
```

where **young,** ***big*** **and** ***small*** **are fuzzy terms, and** *age, Speed_error* **and** *Throttle_change* **are linguistic (fuzzy) variables.**

Each linguistic (fuzzy) variable has an associated fuzzy term set (called *primary terms* in FuzzyCLIPS) that is the set of values that the fuzzy variable may take. For example, the fuzzy variable *water_temperature* might have the primary term set {*cold, warm, hot*}, where each primary term represents a specific fuzzy set. Figure 3: Primary terms of a *linguistic variable*, illustrates the primary term values of the fuzzy variable *water_temperature*.

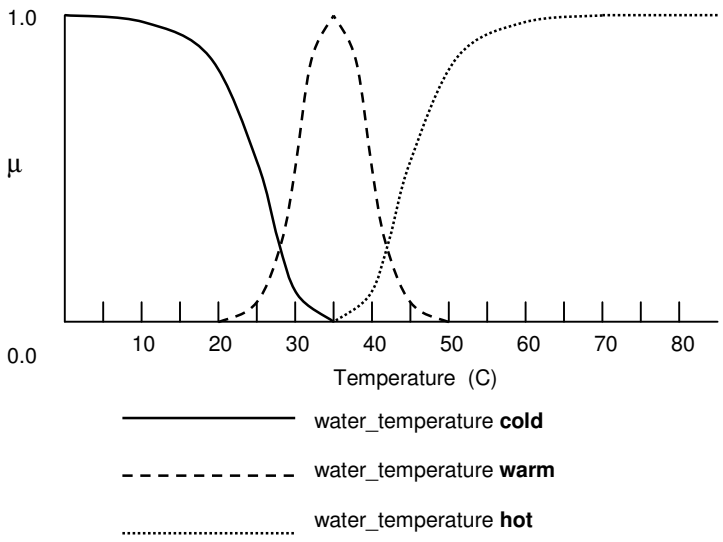


Figure 3: Primary terms of a *linguistic variable*

The syntax for defining fuzzy variables and fuzzy terms is discussed in detail in Section 6.1.

5.2 Uncertainty

Uncertainty occurs when one is not absolutely certain about a piece of information. The degree of uncertainty is usually represented by a crisp numerical value on a scale from 0 to 1, where a certainty factor[6] of 1 indicates that the expert system is very certain that a fact is true, and a certainty factor of 0 indicates that it is very uncertain that a fact is true.

In FuzzyCLIPS, a fact is composed of two parts: the fact in the sense of standard CLIPS and its certainty factor. In general a FuzzyCLIPS fact takes the following form:

6 Note that only facts have associated certainty factors. Object instances do not have certainty factors in this version of FuzzyCLIPS so that all object instances are treated as if they had certainty factors of 1.0.

(fact) [CF certainty factor]

The CF acts as the delimiter between the fact and the certainty factor and [ ] indicates an optional part. For example,

(prediction sunny) CF 0.8

is a fact that indicates that the weather will be sunny with a certainty of 80%. However, if the certainty factor is omitted, as in a normal CLIPS fact,

(prediction sunny)

then FuzzyCLIPS assumes that the weather will be sunny with a certainty of 100%.

Certainty factors may also be associated with entire rules, as illustrated by Example 2.

Example 2

```
(defrule flight-rule
     (declare (CF 0.95)) ;declares certainty factor of the rule
     (animal type bird)
 =>
     (assert (animal can fly))
)
```

represents the hypothesis that, 95% of the time, if an animal is a bird, then it can fly. Similar to facts, if the certainty factor of a rule is not declared, it is assumed to be equal to the value 1.0.

Uncertainty and fuzziness can occur simultaneously (see Example 3).

Example 3

```
(deffacts FuzzyAndUncertainFact
     (Speed_error more_or_less zero) CF 0.9
)


(defrule Uncertain_rule
     (declare (CF 0.8) )
     (Johns_age young)
 =>
     (assert (John goes to school))
)
```

where Speed_error and Johns_age are fuzzy variables, zero and young are fuzzy terms, more_or_less is a fuzzy term modifier and 0.9 and 0.8 are certainty factors associated with a fact and a rule respectively.

5.3 Inference Techniques

Rule evaluation depends on a number of different factors, such as whether or not fuzzy variables are found in the antecedent or consequent part of a rule, whether a rule contains multiple antecedents or consequents, whether a fuzzy fact being asserted has the same fuzzy variable as an already existing fuzzy fact (global contribution), and so on. In this section, the algorithms for evaluating the certainty factors and fuzzy objects of rules will be discussed. The calculation of certainty factors for fact asserted in rules can be controlled by the use of 2 functions, enable-cf-rule-calculation and disable-cf-rule-calculation (see section 6.9.4). When rule certainty factor calculations are enabled they are calculated as described in this section. When disabled, they are assigned the value specified for the fact (or set to 1.0 if no value is specified).

5.3.1 Simple Rules

Consider the simple rule of the form

if A then C	CF_r
A′	CF_f

C′	CF_c

where:

A is the antecedent of the rule

A′ is the matching fact in the fact database

C is the consequent of the rule

C′ is the actual consequent calculated

CF_r is the certainty factor of the rule

CF_f is the certainty factor of the fact

CF_c is the certainty factor of the conclusion

Three types of simple rules are defined: CRISP_, FUZZY_CRISP, and FUZZY_FUZZY. If the antecedent of the rule does not contain a fuzzy object, then the type of rule is CRISP_ regardless of whether or not a consequent contains a fuzzy fact. If only the antecedent contains a fuzzy fact, then the type of rule is FUZZY_CRISP. If both antecedent and consequent contain fuzzy facts, then the type of rule is FUZZY_FUZZY.

5.3.1.1 CRISP_ Simple Rule

If the type of rule is CRISP_, then A′ must be equal to A in order for this rule to fire. This is a "normal" CLIPS rule (actually A would be a pattern and A′ would match the pattern specification, but for simplicity we will not deal with patterns). In that case the conclusion C′ is equal to C, and

$$CF_c = CF_r * CF_f \quad (2)$$

Example 4

```
Given a rule:
(defrule crisp-simple-rule
     (declare (CF 0.7))                  ;crisp rule certainty factor of 0.7
     (light_switch off)                  ;crisp antecedent
 =>
     (assert (illumination_level dark))  ; fuzzy consequent
)                                        ; end of rule definition

and given that the fact:
     (light_switch off) CF 0.8
has been asserted.

Then the following fact will be asserted into the fact database due to the
firing of the crisp-simple-rule:
     (illumination_level dark) CF 0.56
where the certainty factor of the conclusion has been calculated as follows
     CF_c = 0.7 * 0.8
```

5.3.1.2 FUZZY_CRISP Simple Rule

If the type of rule is FUZZY_CRISP, then A′ must be a fuzzy fact[7] with the same fuzzy variable as specified in A for a match to occur and the rule to be placed on the agenda. In addition, while values of the fuzzy variables A and A′ represented by the fuzzy sets F_α and F'_α do not have to be equal, they must overlap. For example, the fuzzy facts (*temperature high*) and (*pressure high*) do not match because the fuzzy variables *temperature* and *pressure* are not the same. However, given the fuzzy facts (*pressure low*), (*pressure medium*), and (*pressure high)*, as illustrated in Figure 4: Matching of fuzzy facts, clearly (*pressure low*) and (*pressure medium*) overlap and thus match, while (*pressure low*) and (*pressure high*) do not match.[8]

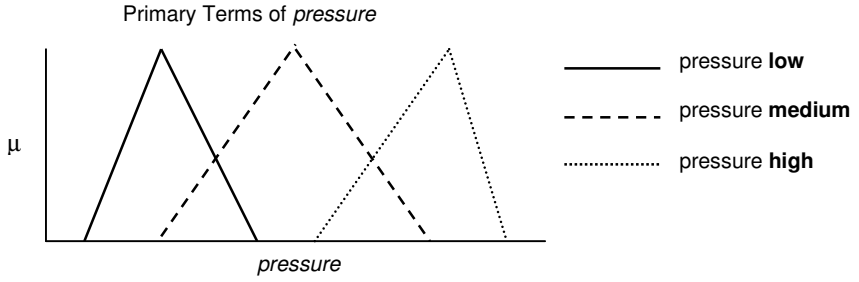


Figure 4: Matching of fuzzy facts

For a FUZZY_CRISP rule, the conclusion C′ is equal to C, and

$$CF_c = CF_r * CF_f * S$$

where S is a measure of similarity between the fuzzy sets F_α (determined by the fuzzy pattern A) and F'_α (of the matching fact A′). The measure of similarity is based upon the measure of possibility P and the measure of necessity N. It is calculated according to the following formula [13]

$S = P(F_\alpha \mid F_{\alpha'})$ if $N(F_\alpha \mid F_{\alpha'}) > 0.5$
$S = (N(F_\alpha \mid F_{\alpha'}) + 0.5) * P(F_\alpha \mid F_{\alpha'})$ otherwise

where[9]

$$P(F_\alpha \mid F'_\alpha) = \max(\min(\mu_{F_\alpha}(u), \mu_{F'_\alpha}(u))), \quad \forall u \in U$$

and

$$N(F_\alpha \mid F'_\alpha) = 1 - P(\overline{F}_\alpha \mid F'_\alpha)$$

$\overline{F}_\alpha$ is the complement of F_α described by the following membership function

---

7 In this and the next sections we deal only with fuzzy facts whose relation name is the name of a fuzzy deftemplate. However, in general a fact may contain fuzzy slots in a standard deftemplate fact. We will refer to these as fuzzy deftemplate facts and fuzzy facts, respectively (although a fuzzy deftemplate fact is also a fuzzy fact). Details are described later.

8 There is some control over this via a FuzzyCLIPS feature called the alpha-value. This is a number between 0 and 1 that is used to specify the minimum overlap required to declare a match. Normally this is set to 0.0 so that any overlap is a match.

9 min is the minimum and max is the maximum so that max(min(a,b)) would represent the maximum of all the minimums between pairs a and b.

$$\mu_{\bar{F}_\alpha}(u) = 1 - \mu_{F_\alpha}(u), \ \forall u \in U$$

Therefore, if the similarity between the fuzzy sets associated with the fuzzy pattern (A) and the matching fact (A′) is high the certainty factor of the conclusion is very close to $CF_r * CF_f$ since S will be close to 1. If the fuzzy sets are identical then S will be 1 and the certainty factor of the conclusion will equal $CF_r * CF_f$. If the match is poor then this is reflected in a lower certainty factor for the conclusion. Note also that if the fuzzy sets do not overlap then the similarity measure would be zero and the certainty factor of the conclusion would be zero as well. In this case the conclusion should not be asserted and the match would be considered to have failed and the rule would not be placed on the agenda.

Example 5

```
(defrule simple-fuzzy-crisp-rule
     (declare (CF 0.7))           ;rule has a certainty factor of 0.7
     (fuzzy-fact fact2)           ;fuzzy antecedent
 =>
     (assert (crisp-fact fact3))  ;crisp consequent
)
where (fuzzy-fact fact1) CF 0.8 is the matching fact in the fact database,
and the fuzzy sets are illustrated in Figure 5: Fact and antecedent fuzzy
sets
```

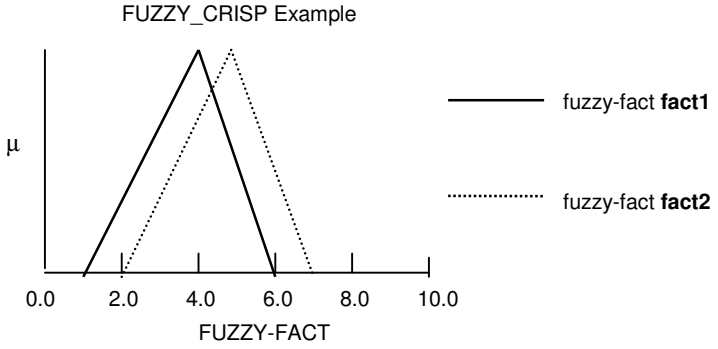


Figure 5: Fact and antecedent fuzzy sets

First, the necessity is calculated as in Figure 6.

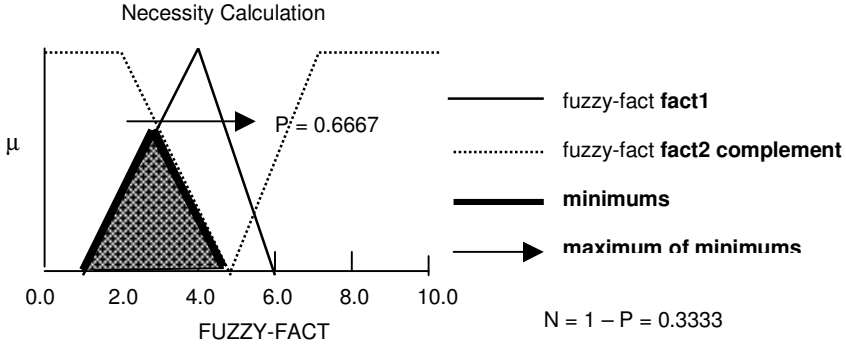


Figure 6: $N(F_\alpha|F_\alpha')$

Since the necessity is less than 0.5, S = (N(F_α | $F_{\alpha'}$) + 0.5) * P(F_α | $F_{\alpha'}$) (see Figure 7).

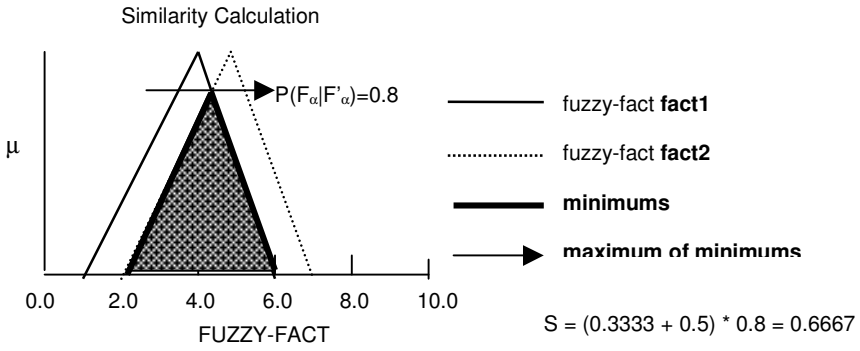


Figure 7: S=(N(F_α|F_α´)+0.5)*P(F_α|F_α´)

And thus CF_c = (0.7) * (0.8) * (0.6667) = 0.3733.

5.3.1.3 FUZZY_FUZZY Simple Rule

If the type of rule is FUZZY_FUZZY, and the fuzzy fact and antecedent fuzzy pattern match in the same manner as discussed for a FUZZY_CRISP rule, then it is shown in [16] that the antecedent and consequent of such a rule are connected by the fuzzy relation

$$R = F_\alpha * F_c$$

where

F_α is a fuzzy set denoting the value of the fuzzy antecedent pattern
F_c is a fuzzy set denoting the value of the fuzzy consequent

In the current version of the system the membership function of the relation R is calculated according to the formula

$$\mu_R(u,v) = \min(\mu_{F_\alpha}(u), \mu_{F_c}(v)), \quad \forall(u,v) \in U \times V$$

Other algorithms for forming this relation can be found in [15]. The calculation of the conclusion is based upon the compositional rule of inference [16], which can be described as follows:

$$F_c' = F_\alpha' \circ R$$

where F_c' is a fuzzy set denoting the value of the fuzzy object of the consequent. The membership function of F_c' is calculated as follows [17]

$$\mu_{F_c'}(v) = \max_{u \in U}(\min(\mu_{F_\alpha'}(u), \mu_R(u,v)))$$

which may be simplified to

$$\mu_{F_c'}(v) = \min(z, \mu_{F_c}(v))$$

where

$$z = \max(\min(\mu_{F'\alpha}(u), \mu_{F_\alpha}(u)))$$

The certainty factor of the conclusion is calculated according to (2) $CF_c = CF_r * CF_f$.

Example 6

```
(defrule fuzzy-fuzzy-rule
 ;both antecedent and consequent are fuzzy objects
     (temperature hot)
  =>
     (assert (temp_change little))
)

(temperature warm)                ; a fact in the fact database
```

A graphical illustration of the matching of the fuzzy fact with the fuzzy pattern and the generation of the fuzzy conclusion is shown below in Figure 8. Note that this type of inference method is commonly referred to as **max-min** rule of inference. The conclusion set is simply *clipped* off at the z value. Figure 9 shows the same results using a **max-prod** rule of inference. In this case the conclusion has all of its membership values scaled by the z value. The FuzzyCLIPS function set-inference-type allows the control of which method is used.

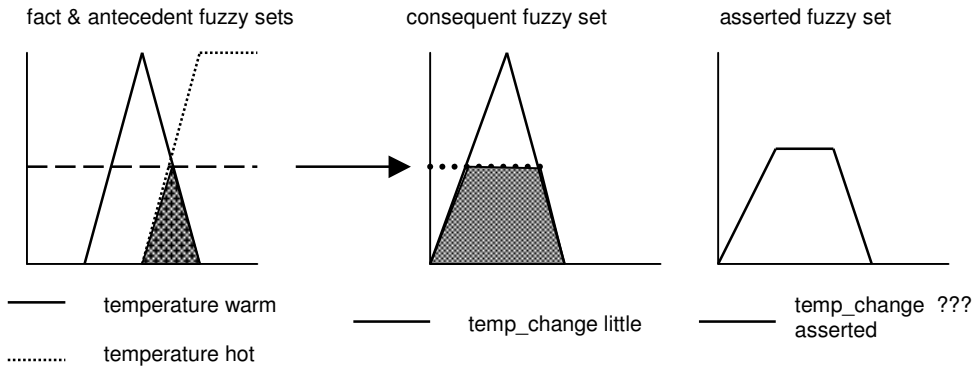


Figure 8: Compositional rule of inference[10] (max-min)

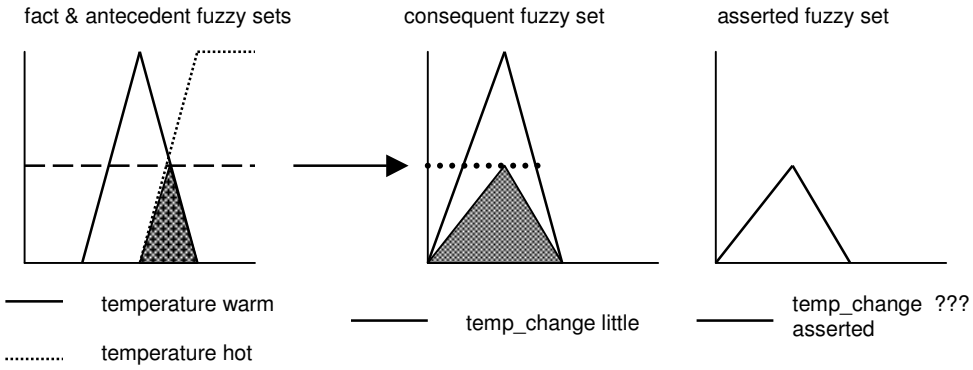


Figure 9: Compositional rule of inference (max-prod)

10 ??? is used to denote an unknown linguistic expression. The fuzzy set denoted by the linguistic expression "temp_change little" once clipped or scaled is difficult to assign a linguistic expression to.

5.3.2 Complex Rules

5.3.2.1 Multiple Consequents

In CLIPS, the consequent part of the rule can only contain multiple patterns (C_1, C_2 ,..., C_n) with the implicit **and** conjunction between them. They are treated as multiple rules with a single consequent. So the following rule:

if Antecedents then C_1 and C_2 and ... and C_n

is equivalent to the following rules:

if Antecedents then C_1
if Antecedents then C_2
...
if Antecedents then C_n

5.3.2.2 Multiple Antecedents

From the above, clearly, only the problem of multiple patterns in the antecedent with a single assertion in the consequent needs to be considered. If the consequent assertion is not a fuzzy fact, no special treatment is needed since the conclusion will be the crisp (non-fuzzy) fact. However, if the consequent assertion is a fuzzy fact, the fuzzy value is calculated using the following basic algorithm [18].

If logical **and** is used, one has

if A_1 **and** A_2 then C	CF_r
A'_1	CF_{f1}
A'_2	CF_{f2}

C′	CF_c

where A'_1 and A'_2 are facts (crisp or fuzzy) that match the antecedents A_1 and A_2 respectively. In this case the fuzzy set describing the value of the fuzzy assertion in the conclusion is calculated according to the formula

$$F'_c = F'_{c1} \cap F'_{c2}$$

where

$\cap$ denotes the intersection of two fuzzy sets[11]
F'_{c1} is the result of fuzzy inference for the fact $A_{1'}$ and the simple rule
if A_1 then C
F'_{c2} is the result of fuzzy inference for the fact $A_{2'}$ and the simple rule
if A_2 then C

In Figure 10 we see the results of a rule in which both A_1 and A_2 are fuzzy patterns. Note that if both A_1 and A_2 were crisp (non-fuzzy) facts then the conclusion would just be the fuzzy fact C since we would be dealing with two CRISP_FUZZY simple rules. If one of the patterns is crisp (say A_1) and the other is fuzzy then the conclusion is F'_{c2} since the CRISP_FUZZY simple rule would conclude C and the FUZZY_FUZZY simple rule would conclude F'_{c2}. The intersection of these two would just be F'_{c2}.

[11] A membership function of a fuzzy set C which is the intersection of fuzzy sets A and B is defined by the following formula $\mu_c(x) = \min(\mu_A(x), \mu_B(x))$, for $x \in U$

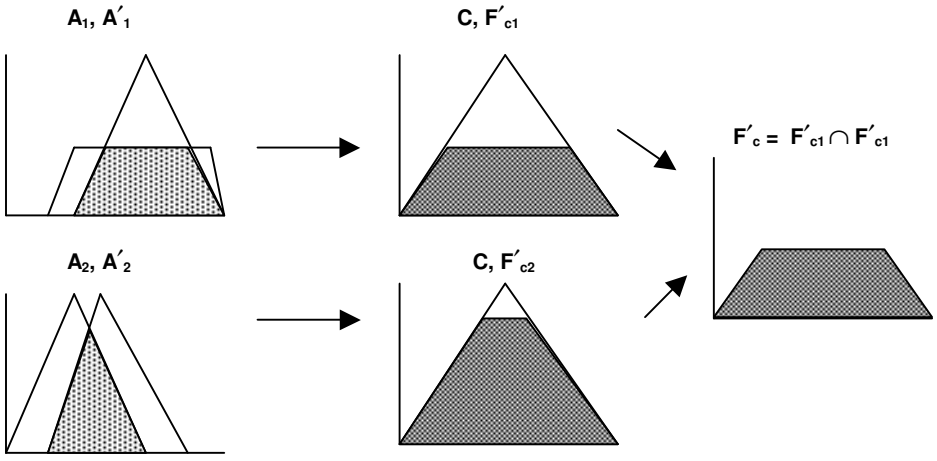


Figure 10: Compositional rule for multiple antecedents

The certainty factor of the conclusion is calculated according to MYCIN's model

$$CF_c = \min(CF'_{f1}, CF'_{f2}) * CF_r$$

where **min** denotes the minimum of the two numbers and

CF'_{f1} is the CF of the simple rule if A_1 then C given the matching fact A'_1
CF'_{f2} is the CF of the simple rule if A_2 then C given the matching fact A'_2

The above algorithm[12] can be applied repeatedly to handle any combination of antecedent patterns, i.e.,

$$F'_c = F'_{c1} \cap F'_{c2} \ldots \cap F'_{cn}$$
$$CF_c = \min(CF'_{f1}, CF'_{f2}, \ldots, CF'_{fn}) * CF_r$$

5.3.3 Global Contribution

In standard CLIPS a fact is asserted with specific values for its fields. If the fact already exists then the behaviour is as if the fact was not asserted (unless fact duplication is allowed). In such a crisp system there is never any need to re-assess the facts in the system - once they exist, they exist (unless certainty factors are being used as discussed below; then the certainty factors are modified to account for the new evidence). But in a fuzzy system, refinement of a fuzzy fact may be possible. Thus, in the case where a fuzzy fact is asserted, this fact is treated as giving contributing evidence towards the conclusion about the fuzzy variable (it contributes globally). If information about that fuzzy variable has already been asserted then this new evidence (or information) about the fuzzy variable is combined with the existing information in the fuzzy fact. The concept of fact duplication for fuzzy facts, therefore, does not apply as it does for standard CLIPS facts. See section 8.1 for details. There are many readily identifiable methods of combining evidence. In the current version of the system, the new value of the fuzzy fact is calculated in accordance with the formula

$$F_g = F_f \cup F'_c$$

where

[12] Later we will discuss fuzzy facts that are not fuzzy deftemplate facts (facts whose relation name is a fuzzy deftemplate). These facts can have fuzzy values in one or more slots of the fact.

F_g is the new value of the fuzzy fact
F_f is the existing value of the fuzzy fact
F'_c is the value of the fuzzy fact to be asserted
$\cup$ denotes the union[13] of two fuzzy sets

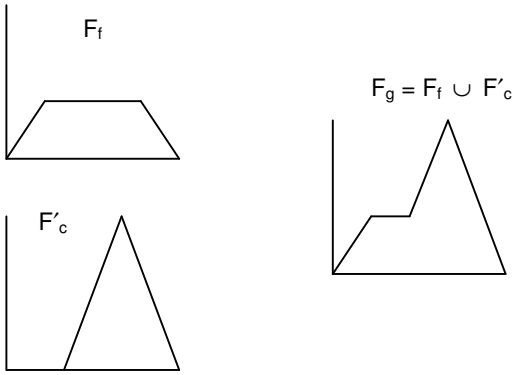


Figure 11: Union of fuzzy sets - global contribution

The uncertainties are also aggregated to form an overall uncertainty. Basically, two uncertainties are combined, using the following formula

$$CF_g = \text{maximum}(CF_f , CF_c)$$

where

CF_g is the combined uncertainty
CF_f is the uncertainty of the existing fact
CF_c is the uncertainty of the asserted fact

As an example of the importance of the global contribution to a fuzzy fact, consider the implementation of a fuzzy logic controller. In this case the user has to ensure the firing of all rules that contribute to the control action to be performed, before any other rule (usually defuzzification) fires. This can be done by attaching a suitable salience to the rules or by separating the rules that all contribute to the same control action into a separate MODULE. As an example see Appendix A and also see the example FuzzyCLIPS programs fzCmplr.clp and fzCmpmod.clp, which use salience and MODULES, respectively, to ensure all fuzzy rules that are related fire together.

5.3.4 Threshold Certainty Factors

In FuzzyCLIPS it is possible to set a threshold certainty factor value such that no rule will be fired unless the rule has a calculated certainty factor value greater than or equal to the threshold value (see also [19]). This feature may be useful in preventing a chain of rules with very low certainty and little logical contribution from firing, and thus speed up the run time. The default is to have no threshold certainty factor set (i.e., a threshold of 0.0), and for rules to be fired as usual (see also Section 6.9.4). The calculated certainty factor for a rule is

$$CF_{rule} * \min(CF_1, \ldots ,CF_n)$$

where CF_{rule} is the certainty factor for the rule and CF_i are the certainty factors for the facts that matched the n patterns on the lefthand side of the rule.

---

[13] A membership function of a fuzzy set C which is the union of fuzzy sets A and B is defined by the following formula $\mu_c(x) = \max(\mu_A(x), \mu_B(x))$, for $x \in U$

Example 7

```
(defrule   below-threshold-rule
     (declare (CF 0.5))           ;rule certainty factor of 0.5
     (fuzzy-fact antecedent-fact) ;fuzzy antecedent
 =>
     (assert (crisp-fact c1))             ;crisp consequent
     (assert (another-fuzzy-fact c2))     ;fuzzy consequent
)
```

Suppose the following fact has been asserted

```
     (fuzzy-fact fact-list-fact) CF 0.6
```

The calculated certainty factor for the rule is

```
     CF = 0.5 * 0.6 = 0.3
```

The rule will fire only if the threshold certainty factor is less than or equal to 0.3 at the time the rule is selected to fire.

The certainty factor for (*another-fuzzy-fact c2*), the fuzzy consequent in the previous rule, is

```
     CF = 0.5 * 0.6 = 0.3
```

from equation (2), as for a simple FUZZY_FUZZY rule. However, the certainty factor for (*crisp-fact c1*) is

```
     CF = 0.5 * 0.6 * S,
```

where S is the measure of similarity, as for a simple FUZZY_CRISP rule. Suppose that S=0.8. Then the following conclusions are reached on the RHS

```
     =>
          (assert (crisp-fact c1))                ;calculated CF = 0.24
          (assert (another-fuzzy-fact c2))  ;calculated CF = 0.3
```

Combining rule certainty factors, fact certainty factors, multiple patterns on the lefthand side of a rule and multiple assertions on the righthand side of a rule can lead to a complicated determination of threshold and certainty factors for the asserted facts.

Example 8

```
(defrule complex
     (declare (CF 0.9))
     (crisp1)
     (fuzzy1 very few)
  =>
     (assert (crisp2)
               (fuzzy2 hot))
)
```

with asserted facts

```
     (crisp1) CF 0.8
     (fuzzy1 few) CF 0.7
```

The calculated certainty factor for the rule is

```
     0.9 * min(0.8, 0.7) = 0.63
```

Therefore the rule will fire if the threshold value is less than or equal to

```
0.63 and the certainty factors of the two asserted facts will be (assuming
the similarity between (fuzzy1 very few) and (fuzzy1 few) is 0.6)

     CF of (crisp2) = 0.9 * min(0.8, 0.7*0.6) = 0.378
     CF of (fuzzy1 few) = 0.9 * min(0.8, 0.7) = 0.63
```

Note that the calculated certainty factor for a rule is evaluated when the rule is selected to fire and NOT when it is added to the agenda. This is because the certainty factors for the facts that match the patterns of the rule can change due to global contribution while the rule is on the agenda.

5.3.5 Certainty Factors in Assert Statements

When a fact is asserted in FuzzyCLIPS, its certainty factor may be specified. It is thus possible to assert a fact on the RHS of a rule with a specific certainty factor.

Example 9

```
(assert (some-consequent) CF 0.8)  ;asserting fact with CF = 0.8
```

Note that a certainty factor will also be calculated according to the nature of the rule (having multiple antecedents, FUZZY_FUZZY, CRISP_, or FUZZY_CRISP) and any global contribution. This calculated certainty factor will then be multiplied by the certainty factor given in the assert statement. This could be useful as a method of assigning weighted certainty factors for a rule with multiple consequents. Note that if the fact is not asserted during a rule execution (e.g., from the CLIPS command level) then only global contribution will apply in determining the certainty factor for the asserted fact (i.e., if the fact does not exist already then the certainty will be as specified in the assert and if it does exist the certainty will be modified to be the maximum of the existing certainty and the certainty specified in the assert).

Example 10

```
(defrule assert-cf-rule
     (declare (CF 0.8));rule CF is 0.8
     (fact 1)
   =>
     (assert (c1))
     (assert (c2) CF 0.7) ;assert c2 with CF 0.7
     (assert (c3))
     (assert (c4))
)

where
     (fact 1) CF 0.9     ; matching fact has CF 0.9

the conclusions reached on the RHS would be:

     (assert (c1))         ; CF = 0.8*0.9     = 0.72
     (assert (c2) CF 0.7)  ; CF = 0.8*0.9*0.7 = 0.504
     (assert (c3))         ; CF = 0.8*0.9     = 0.72
     (assert (c4))         ; CF = 0.8*0.9     = 0.72

The above rule attaches a lower CF to conclusion c2 than to the other
conclusions.
```

5.4 Defuzzification

The outcome of the fuzzy inference process is a fuzzy set, specifying a fuzzy distribution of a conclusion. However, in some cases, such as control applications, only a single discrete action may be applied, so a single point that reflects the best value of the set needs to be selected. This process of reducing a fuzzy set to a single point is known as *defuzzification*.

There are several possible methods, each one of which has advantages and disadvantages. A method which has been widely adopted is to take the center of gravity COG or moment) of the whole set. This has the advantage of producing smoothly varying controller output, but it is sometimes criticized as giving insufficient weight to rule consequents that agree and ought to reinforce each other. Another method concentrates on the values where the possibility distribution reaches a maximum, called the mean of maxima method. The mean of maxima (MOM) algorithm is criticized as producing less smooth controller output, but has the advantage of greater speed due to fewer floating-point calculations.

In FuzzyCLIPS, the user has the option of choosing either the COG or MOM algorithm when defuzzifying a fuzzy set (For details on the functions that perform defuzzification in FuzzyCLIPS see Section 6.7)

5.4.1 Centre of Gravity Algorithm

The centre of gravity method may be written formally as

$$x' = \frac{\int_{x \in U} (x \cdot f(x)) dx}{\int_{x \in U} f(x) dx}$$

where x' is the recommended, defuzzified value, and the universe of discourse is U.

In FuzzyCLIPS, a fuzzy set is defined by a set of points that are considered to be connected by straight-line segments. The integral then reduces to a simple summation,

$$x' = \frac{\sum_{i=1}^{n} x_i' \cdot A_i}{\sum_{i=1}^{n} A_i}$$

where x_i' is the local centre of gravity, A_i is the local area of the shape underneath line segment (p_{i-1}, p_i), and n is the total number of points.

Example 11

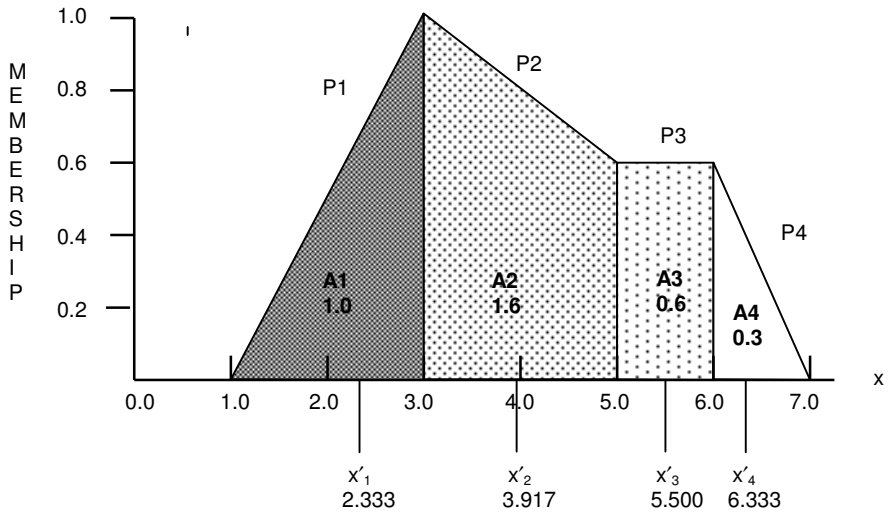


Figure 12: Example of COG defuzzification

For each shaded subsection in the figure above, the area and centre of gravity is calculated according to the shape identified (i.e., triangle, rectangle or trapezoid). The centre of gravity of the whole set is then determined:

$$x' = \frac{2.333 \cdot 1.0 + 3.917 \cdot 1.6 + 5.5 \cdot 0.6 + 6.333 \cdot 0.3}{1.0 + 1.6 + 0.6 + 0.3} = 3.943$$

5.4.2 Mean of Maxima Algorithm

The MOM algorithm returns the x-coordinate (x″) of the point at which the maximum membership (y) value of the set is reached.

Example 12

Given the fuzzy set illustrated in Figure 12: Example of COG defuzzification, the MOM result would be 3.0.

If the maximum y value is reached at more than one point, then the average of all the x″ is taken. More formally:

$$x' = \sum_{j=1}^{J} \frac{x''_j}{J}$$

where x''_j are the x-coordinates of all the maxima, and J is the total number of maxima (see Figure 13: Examples of MOM defuzzification).

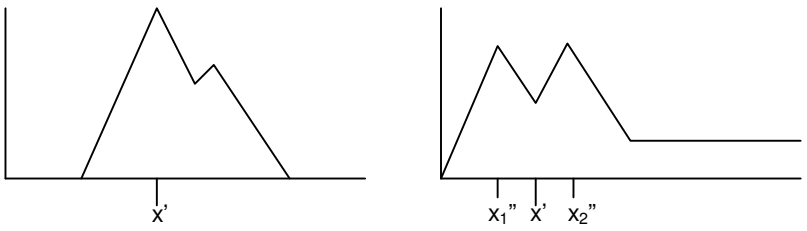


Figure 13: Examples of MOM defuzzification

Note that an ambiguity occurs when the maximum value occurs along a plateau (see Figure 14: MOM example - Ambiguity) rather than just a series of individual peaks (see Figure 13). In this case there are an infinite number of maximum points between x_1'' and x_2'' and using the average of the three points x_1'', x_2'' and x_3'' results in what may be an incorrect or perhaps an unexpected value. It is not entirely clear what the answer should be (see Section 9 for a discussion of this problem and see Section 6.7 to see how to perform defuzzification in FuzzyCLIPS using the moment-defuzzify and maximum-defuzzify functions).

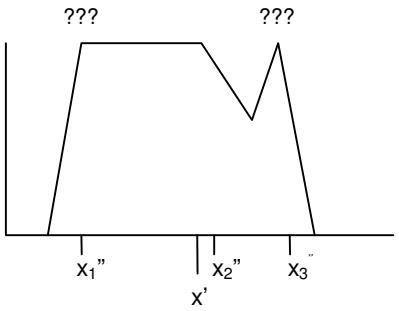


Figure 14: MOM example - Ambiguity

6 Using the FuzzyCLIPS Extensions

The following sections present the syntax for defining fuzzy variables, using fuzzy variables in LHS patterns and in facts, declaring certainty factors, changes made to the assert statement, defuzzification functions and commands for accessing fuzzy parameters, and for accessing certainty factor information.

6.1 Defining Fuzzy Variables in Deftemplate Constructs

All fuzzy variables must be predefined before use with the deftemplate statement. This is an extension of the standard deftemplate construct in CLIPS. The extended syntax of this construct is as follows:

```
(deftemplate <name> ["<comments>"]
   <from> <to> [<unit>]   ; universe of discourse
       (
          t1
          .
          .   ; list of primary terms
          .
          tn
       )
)
```

<name> is the identifier used for the fuzzy variable. The description of the universe of discourse consists of three elements. The <from> and <to> should be floating point numbers. They represent the beginning and end of the interval that describes the domain of the fuzzy variable (the universe of discourse). The value of <from> must be less than the value for <to>. The <unit> is a word that represents the unit of measurement (optional). The t_i are specifications for the fuzzy terms (such as hot, cold, warm) used to describe the fuzzy variable. These specifications describe the shape of the fuzzy set associated with the terms. Example 13 below shows an incomplete fuzzy deftemplate with only the universe of discourse specified.

Example 13

```
(deftemplate water_flow
     0 100 liters/sec
          . . .
)
```

6.1.1 Primary Terms

A primary term t_i (i=1, ..., n) has the form

(<name> <description of fuzzy set>)

where <name> represents the name of a primary term used to describe a fuzzy concept, and <description of fuzzy set> defines a membership function of the given primary term. The membership function can be described using either a singleton representation, a standard function representation, or a linguistic expression that uses terms defined previously in the fuzzy deftemplate definition.

<description of fuzzy set> ::= <singletons> | <standard> | <linguistic-expr>

6.1.1.1 Singleton Representation

The grade of membership $\mu_A(x)$ of x in fuzzy set A is a positive number and the pair ($\mu_A(x)$, x) will be called a singleton (often these pairs are represented by $\mu_A(x)/x$ or $\mu(x)/x$ for short). A fuzzy set A in a universe of discourse U can be described as follows:

$$A = \int_{x \in U} \mu_A(x) / x$$

where the integration symbol denotes the union of singletons.

If a universe of discourse U is a finite set, then A is expressed as follows:

$$A = \sum_{i=1}^{n} \mu(x_i)/x_i = \mu(x_1)/x_1 + \mu(x_2)/x_2 + \ldots + \mu(x_n)/x_n$$

In FuzzyCLIPS we consider the universe of discourse to be a range of the real number line and do not deal with finite sets for U. A singleton will be represented here as a pair $(x_i\ \mu(x_i))$. A fuzzy set A will be represented as a list of singletons

<singletons> ::= $(x_1\ \mu_1)\ (x_2\ \mu_2)\ \ldots\ (x_n\ \mu_n)$

where:

$x_i \le x_{i+1}$ for i = 1, 2, ... , n-1
x_i is an element from U
μ_i is a number denoting the grade of membership of x in the fuzzy set A

As mentioned in Section 4.1, a fuzzy set is represented by an ordered set of points joined by straight line segments. The grade of membership of an x value not listed in a list of singletons will be calculated on the basis of interpolation according to the following formula (for points that do not have multiple membership values; in these cases the membership value is defined to be the maximum of all values at that same x value):

$$\mu(x) = \mu(x_1), \quad x \le x_1$$

$$\mu(x) = \mu(x_i) + \frac{\mu(x_{i+1}) - \mu(x_i)}{x_{i+1} - x_i}(x - x_i), \quad x_i < x \le x_{i+1}$$

$$\mu(x) = \mu(x_n), \quad x_n < x$$

Example 14

Let U = {x | 0 ≤ x ≤ 9}

We may then define a fuzzy set "few" as follows

μ(0) = 0, μ(1) = 0, μ(2) = 0.3, μ(3) = 0.9,
μ(4) = 1, μ(5) = 0.8, μ(6) = 0.5, μ(7) = 0,
μ(8) = 0, μ(9) = 0

One can represent this fuzzy set by the following list of singletons

(1 0) (2 0.3) (3 0.9) (4 1) (5 0.8) (6 0.5) (7 0)

One can also show this set graphically as in Figure 15: Fuzzy Set of group *few*.

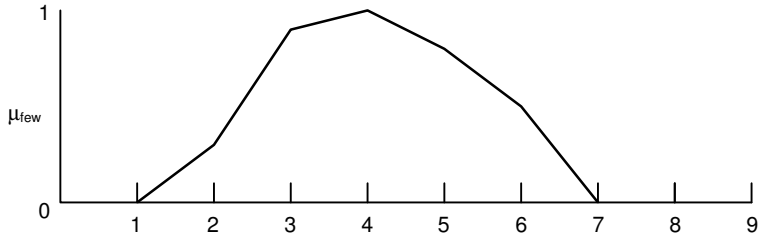


Figure 15: Fuzzy Set of group *few*

In FuzzyCLIPS one can define a linguistic variable *group* that has a universe of discourse from 0 to 9 (units unspecified) and the primary term *few* as follows:

```
(deftemplate group           ;a linguistic variable declaration
     0 9                     ;universe of discourse limits (no units)
     (                       ;start of primary term declarations
       ; a primary term few described in singleton notation
       (few (1 0) (2 0.3) (3 0.9) (4 1) (5 0.8) (6 0.5) (7 0))
     )                       ;end of primary term declarations
)                            ;end of fuzzy deftemplate
```

Note that it is possible for consecutive x values to be the same. This describes a crisp boundary in the fuzzy set (vertical line). If more than three points have the same value then only three will be kept (the fourth will always replace the third). If two points have exactly the same x and y values then one of them will be discarded. Consider the following two examples of fuzzy sets with crisp boundaries.

Example 15

The singleton set described as

```
     (three (3 0) (3 1) (3 0))
```

might represent the crisp concept 3 as a fuzzy set. It is shown graphically as

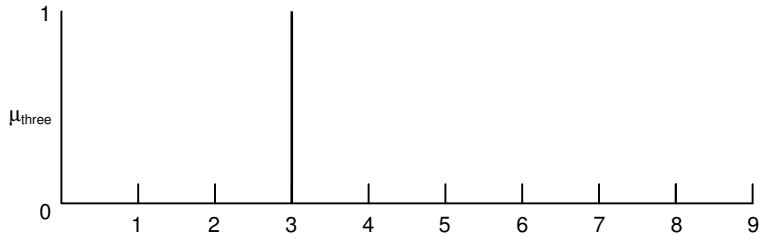


Figure 16

Example 16

Another more complex (and probably unrealistic) set might be defined with the following set of singleton values

```
(weird      (1 0) (1 1) (1 0) (3 0) (4 .25)
            (4 1) (4 .4) (4 .5) (6 1) (8 0) )
```

The graph follows. Note that in this case the point (4 .4) is discarded.

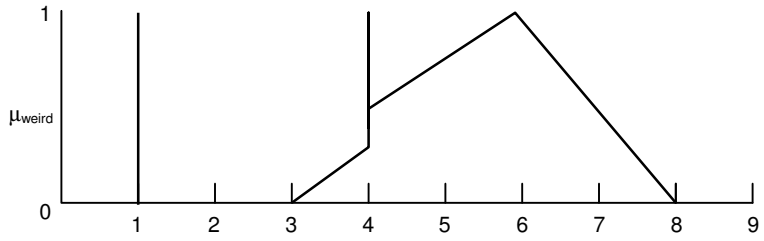


Figure 17

6.1.1.2 Standard Function Representation

Frequently, it is useful to describe a membership function using one of a set of standard functions S, Π, or Z. Parameters of these functions can be chosen, depending on applications. These functions are defined as follows and are shown graphically in Figure 18: Standard fuzzy set functions.

$$S(u,a,c) = 0, \quad u \le a,\ u \in U$$

$$S(u,a,c) = 2\left(\frac{u-a}{c-a}\right)^2, \quad a < u \le \frac{a+c}{2}$$

$$S(u,a,c) = 1 - 2\left(\frac{c-u}{c-a}\right)^2, \quad \frac{a+c}{2} < u \le c$$

$$S(u,a,c) = 1, \quad c < u$$

$$Z(u,a,c) = 1 - S(u,a,c)$$

$$\Pi(u,d,b) = S(u,b-d,d), \quad u \le b$$

$$\Pi(u,d,b) = Z(u,b,b+d), \quad b < u$$

Note that the names are suggestive of the shape of the functions.

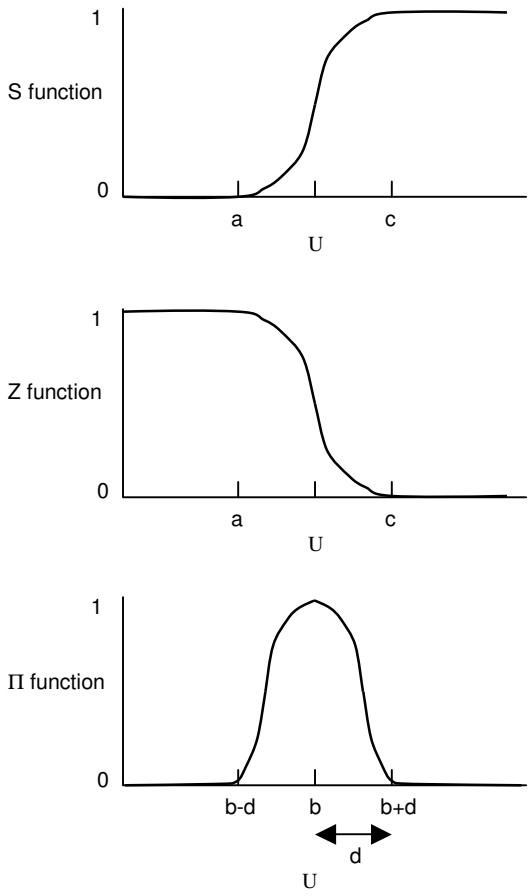


Figure 18: Standard fuzzy set functions

A standard representation of a membership function has the following format:

<standard> ::= (S a c) | (s a c) | (Z a c) | (z a c) | (PI d b) | (pi d b)

where : a, b, c, d are numbers that represent the parameters of the respective functions.

Example 17

```
(deftemplate Tx   "output water temperature"
     5 65 Celsius
     (  (cold (z 10 26))     ;standard set representation
        (OK (PI 2 36))
        (hot (s 37 60))
     )
 )
```

FuzzyCLIPS converts all standard notation to singleton representation. Nine points, equally spaced along the x axis, are selected to represent the functions (see Figure 19: Approximation of standard functions). The number of points (9) can be changed by modifiying the value of ArraySIZE, which is defined in the file fuzzypsr.h and then recompiling FuzzyCLIPS. Note, however, that increasing this size will increase the computational load during fuzzy inferencing. Also note that in many instances a simple 2 or 3 point singleton set that approximates these functions

will be acceptable and will result in less computation during inferencing.

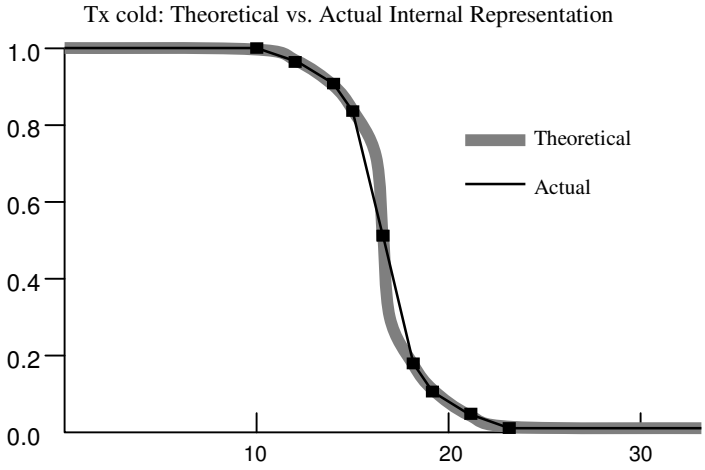


Figure 19: Approximation of standard functions

As a special note, consider the case where the parameter a is equal to c for the S or Z functions. In this case two points will be used to represent the function.

Example 18

```
(S 10 10) will be represented (10 0) (10 1)
(Z 10 10) will be represented (10 1) (10 0)
```

For the PI function, if parameter d is zero then the function will be represented by three points.

Example 19

```
(PI 0 10) will be represented (10 0) (10 1) (10 0)
```

6.1.1.3 Linguistic Expression Representation

Linguistic expressions are defined in Section 5.3 in detail, but a simple example will illustrate the usage.

Example 20

```
(deftemplate temperature
   0 100 C
     (  (cold (z 10 26))           ;standard set representation
        (hot (s 37 60))            ;standard set representation
        (warm not [ hot or cold ]) ; linguistic expression
     )
 )
```

Note that the term *warm* is described as being *not hot or cold*. It uses the terms *hot* and *cold* previously defined in this deftemplate to describe the warm concept. Only terms described in this deftemplate (before the term definition being defined) can be used (along with any available modifiers and the *and* and *or* operations).

6.2 Standard Deftemplate Definitions with Fuzzy Slots (fields)

A fuzzy deftemplate describes a fuzzy variable. One may use these deftemplates to describe fuzzy facts in patterns and assert commands. For example,

```
(defrule high-temp
    (temperature hot)
=>
   (assert (move-throttle negative-big)
   (printout t "The temperature is hot")
)
```

contains a fuzzy pattern based on the fuzzy deftemplate temperature. It also contains an assertion of a fuzzy fact based on the fuzzy deftemplate for move-throttle. Facts that have as their relation name the name of a fuzzy deftemplate will be referred to as **fuzzy deftemplate facts**. These facts in essence have a single slot that holds the fuzzy value described by the linguistic expression (or fuzzy set description). It is also possible to supplement standard CLIPS deftemplate facts by having one or more of the slots be a fuzzy value slot. In these cases the slot is associated with a fuzzy deftemplate description so that the appropriate universe of discourse and terms are known. These facts with fuzzy slots are known as fuzzy facts.[14] A fuzzy slot has the form

(fuzzy-slot> ::= (slot <slotname> (type FUZZY-VALUE <fuzzy-deftemplate-name>)

where <slotname> is the name of the slot and <fuzzy-deftemplate-name> is the name of a previously defined fuzzy deftemplate. Note that no other options are allowed in the slot specification. For example the slot cannot have multiple possible types like INTEGER and FUZZY-VALUE and the options that determine other constraints such default values or cardinality are not allowed. Fuzzy slots can only hold fuzzy values and must have a value.

Example 21

```
;; assume that the fuzzy deftemplates fz-height and
;; fz-weight have already been defined
(deftemplate person
      (slot name (type SYMBOL))
      (slot height (type FUZZY-VALUE fz-height))
      (slot weight (type FUZZY-VALUE fz-weight))
)

(defrule big-person
   (person (name ?n)
            (weight heavy)
            (height tall))
  =>
   (printout t ?n " is a big person" crlf)
)
```

The use of fuzzy slots has a big payoff in many situations. Given the last example and only fuzzy deftemplate facts it would be necessary to define a fuzzy deftemplate for each person's weight and each person's height as well as rules for each person. In effect a fuzzy deftemplate acts somewhat like a type definition for the language in which a fuzzy variable type is defined.

---

[14] In fact fuzzy deftemplate facts are also fuzzy facts. Internally a fuzzy fact is stored exactly as if it were declared as **(deftemplate tname1 (slot GenericFuzzySlot (type FUZZY-VALUE tname2)))** where tname2 is the name of a fuzzy deftemplate.

6.3 Modifiers (Hedges) and Linguistic Expressions

6.3.1 Predefined Modifiers

As mentioned in Section 4.1, a modifier may be used to further enhance the ability to describe our fuzzy concepts. Modifiers (very, slightly, etc.) used in phrases such as

very hot or *slightly* cold

change (modify) the shape of a fuzzy set in a way that suits the meaning of the word used. These modifiers are commonly referred to as **hedges**. FuzzyCLIPS has a set of predefined modifiers that can be used at any time to describe fuzzy concepts when fuzzy terms are described in fuzzy deftemplates, fuzzy rule patterns are written, or fuzzy facts or fuzzy slots are asserted.

Modifier Name	**Modifier Description**	
not	1-y	
very	y**2	
somewhat	y**0.333	
more-or-less	y**0.5	
extremely	y**3	
above	(see [12])	
below	(see [12])	
intensify2	(y**2) 1 - 2(1-y)**2	if y in [0, 0.5] if y in (0.5, 1]
plus	y**1.25	
norm	normalizes the fuzzy set so that the maximum value of the set is scaled to be 1.0 (y = y*1.0/max-value)	
slightly	intensify (norm (plus A AND not very A))	

These modifiers change the shape of a fuzzy set using mathematical operations on each point of the set. In the above table the variable y represents each membership value in the fuzzy set and A represents the entire fuzzy set (i.e., y**2 squares each membership value and very A applies the very modifier to the entire set). Note that when a modifier is used it can be used in upper or lower case (NOT or not) or even a mix of cases (NoT).

The following diagrams show the effect of each of the modifiers on a base fuzzy set. The diagrams have been produced using the *plot-fuzzy-value* function described later in the manual.

```
Fuzzy Value: base-fv
Linguistic Value: base (*)

 1.00                          *
 0.95
 0.90                        * *
 0.85                       *   *
 0.80
 0.75                      *     *
 0.70                     *       *
 0.65
 0.60                    *         *
 0.55
 0.50                   *           *
 0.45                  *             *
 0.40
 0.35                 *               *
 0.30                *                 *
 0.25
 0.20               *                   *
 0.15
 0.10              *                     *
 0.05             *                       *
 0.00*************                         *************
     |----|----|----|----|----|----|----|----|----|----|
    0.00     20.00     40.00     60.00     80.00    100.00
```

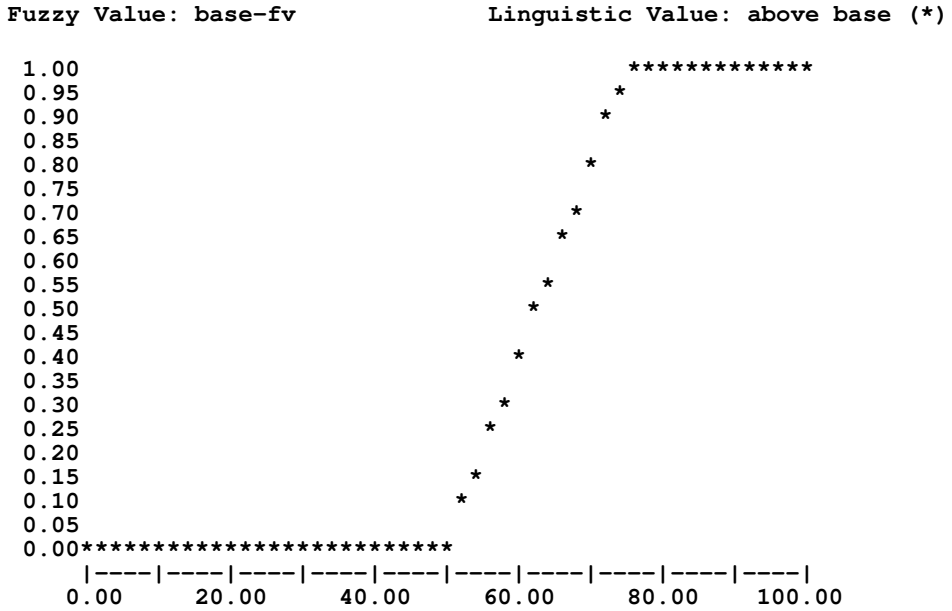
```
Fuzzy Value: base-fv             Linguistic Value: not base (*)

 1.00*************                         *************
 0.95             *                       *
 0.90              *                     *
 0.85
 0.80               *                   *
 0.75
 0.70                *                 *
 0.65                 *               *
 0.60
 0.55                   *           *
 0.50                    *         *
 0.45
 0.40                      *     *
 0.35
 0.30                       *   *
 0.25                        * *
 0.20
 0.15                         *
 0.10                        * *
 0.05                       *   *
 0.00                      *     *
     |----|----|----|----|----|----|----|----|----|----|
    0.00     20.00     40.00     60.00     80.00    100.00
```

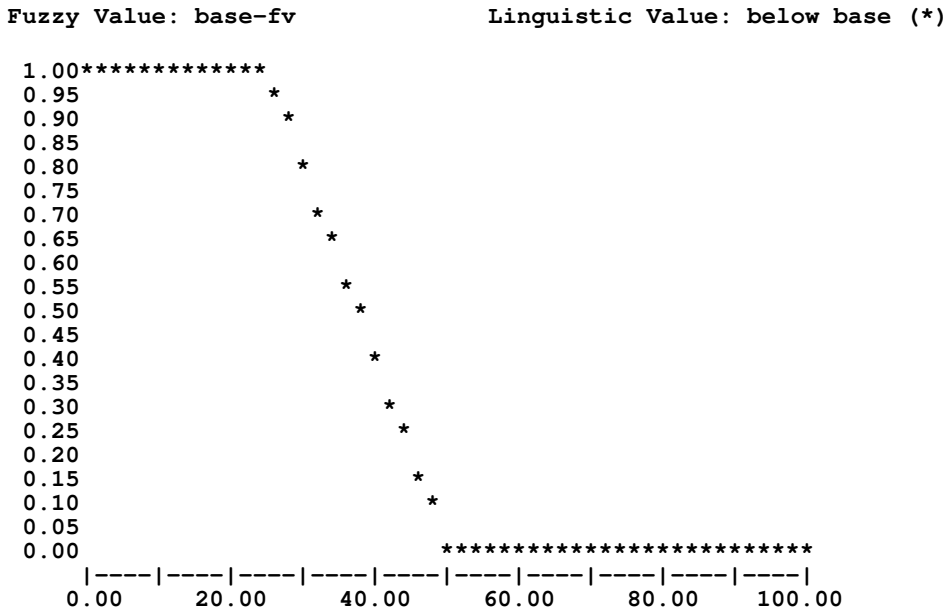
```
Fuzzy Value: base-fv              Linguistic Value: very base (*)

 1.00                          *
 0.95
 0.90
 0.85                        * *
 0.80
 0.75
 0.70                       *   *
 0.65
 0.60                      *     *
 0.55
 0.50
 0.45                     *       *
 0.40
 0.35                    *         *
 0.30
 0.25                   *           *
 0.20                  *             *
 0.15                 *               *
 0.10                *                 *
 0.05               *                   *
 0.00***************                     ***************
     |----|----|----|----|----|----|----|----|----|----|
    0.00     20.00     40.00     60.00     80.00    100.00
```

```
Fuzzy Value: base-fv              Linguistic Value: more-or-less base (*)

 1.00                          *
 0.95                        * *
 0.90                       *   *
 0.85                      *     *
 0.80                     *       *
 0.75                    *         *
 0.70                   *           *
 0.65                  *             *
 0.60                 *               *
 0.55                *                 *
 0.50
 0.45               *                   *
 0.40
 0.35              *                     *
 0.30
 0.25
 0.20
 0.15            *                       *
 0.10
 0.05
 0.00*************                       *************
     |----|----|----|----|----|----|----|----|----|----|
    0.00     20.00     40.00     60.00     80.00    100.00
```

```
Fuzzy Value: base-fv             Linguistic Value: somewhat base (*)

 1.00                         *
 0.95                       ** **
 0.90                     **     **
 0.85                   *         *
 0.80                  *           *
 0.75                 *             *
 0.70                *               *
 0.65               *                 *
 0.60              *                   *
 0.55
 0.50             *                     *
 0.45
 0.40
 0.35
 0.30
 0.25
 0.20            *                       *
 0.15
 0.10
 0.05
 0.00*************                       *************
     |----|----|----|----|----|----|----|----|----|----|
    0.00     20.00     40.00     60.00     80.00    100.00
```

```
Fuzzy Value: base-fv             Linguistic Value: slightly base (*)

 1.00                    **       **
 0.95
 0.90                   *           *
 0.85                     *     *
 0.80
 0.75
 0.70                  *             *
 0.65
 0.60
 0.55
 0.50                        *   *
 0.45                 *               *
 0.40
 0.35
 0.30
 0.25                *                 *
 0.20
 0.15                      * *
 0.10               *                   *
 0.05              *                     *
 0.00**************           *           **************
     |----|----|----|----|----|----|----|----|----|----|
    0.00     20.00     40.00     60.00     80.00    100.00
```


Fuzzy Value: base-fv
Linguistic Value: above base (*)
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.00
20.00
40.00
60.00
80.00
100.00


Fuzzy Value: base-fv
Linguistic Value: below base (*)
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.00
20.00
40.00
60.00
80.00
100.00

```
Fuzzy Value: base-fv              Linguistic Value: intensify base (*)

 1.00                          ***
 0.95                        *   *
 0.90                       *     *
 0.85
 0.80                      *       *
 0.75
 0.70                     *         *
 0.65
 0.60
 0.55                    *           *
 0.50
 0.45
 0.40                   *             *
 0.35
 0.30
 0.25                  *               *
 0.20
 0.15                 *                 *
 0.10                *                   *
 0.05               *                     *
 0.00***************                       ***************
     |----|----|----|----|----|----|----|----|----|----|
    0.00     20.00     40.00     60.00     80.00    100.00
```

```
Fuzzy Value: base-fv              Linguistic Value: extremely base (*)

 1.00                           *
 0.95
 0.90
 0.85
 0.80                         * *
 0.75
 0.70
 0.65
 0.60                       *   *
 0.55
 0.50
 0.45                      *     *
 0.40
 0.35
 0.30                     *       *
 0.25
 0.20                    *         *
 0.15                   *           *
 0.10                  *             *
 0.05                 *               *
 0.00*****************                 *****************
     |----|----|----|----|----|----|----|----|----|----|
    0.00     20.00     40.00     60.00     80.00    100.00
```

```
Fuzzy Value: base-fv2            Linguistic Value: base (+),  norm base (*)

 1.00                          *
 0.95
 0.90                        * *
 0.85                       *   *
 0.80
 0.75                      *     *
 0.70                     *   +   *
 0.65                        + +
 0.60                    *  +   +  *
 0.55                      +     +
 0.50                   * +       + *
 0.45                  *             *
 0.40                    +         +
 0.35                 * +           + *
 0.30                * +             + *
 0.25                  +               +
 0.20               *+                 +*
 0.15               +                   +
 0.10              *                     *
 0.05            *                       *
 0.00*************                       *************
     |----|----|----|----|----|----|----|----|----|----|
    0.00     20.00     40.00     60.00     80.00    100.00
```

These predefined modifiers are available for use at all times in fuzzy deftemplate definitions, fuzzy patterns specifications, fuzzy slot assert specifications, and in the fuzzy-modify function. Some examples below illustrate the use of the modifiers.

Example 22

```
(deftemplate temp
  0 100 C
  ( (cold (Z 20 40))
    (hot  (S 60 80))
    (freezing extremely cold)
  )
)
(defrule temp-rule
  (temp not hot and not cold)
=>
  (printout t "It's such a pleasant day" crlf)
)
```

6.3.2 User Defined Modifiers

The FuzzyCLIPS user may also define[15] modifiers that can be used in exactly the same manner as the predefined ones. This can be done at the FuzzyCLIPS code level or for more complicated definitions or for efficiency at the "C" code level.

Adding a new modifier at the FuzzyCLIPS level is done with the function

[15] Note that this is different than the previous version of FuzzyCLIPS in which modifiers were added in the fuzzy deftemplate definition. This handling of modifiers was felt to be much superior to the old style giving a good set of predefined modifiers and flexibility to add more complicated modifiers at the 'C' code level.

(add-fuzzy-modifier *modname modfunction*)

where

modname is the name (symbol) to be used for the modifier and *modfunction* is the name of a CLIPS function or a user defined deffunction that takes a floating point number and returns a floating point number (the function should return a number between 0 and 1 and if it does not then it will be set to 0 if it is less than 0 or to 1 if it is greater than 1; the function will be passed numbers between 0 and 1)

Example 23

```
(add-fuzzy-modifier my-somewhat sqrt)
(deffunction most-extremely-fcn (?x)
    (** ?x 5)
)
(add-fuzzy-modifier most-extremely most-extremely-fcn)
(deftemplate temp
  0 100 C
  ( (low (10 1) (50 0))
    (high (50 0) (90 1))
    (sort-of-high my-somewhat high)
    (incredibly-low most-extremely low)
  )
)
```

In Example 23 *sqrt* is a system defined function that returns the square root of a number. With this definition the modifier *my-somewhat* will act exactly like the *somewhat* modifier supplied with the system. The function *most-extremely-fcn* is a user defined deffunction that will handle the work of the modifier *most-extremely*. With this definition its behavior will be similar to that of the system supplied modifier *extremely*. The functionality of the modifiers defined this way is somewhat limited (for example, it would not be possible to define the *above*, *below,* or *slightly* modifiers this way).

It is also possible to remove any modifiers added this way with the function

(remove-fuzzy-modifier modname)

Note that only modifiers added with the add-fuzzy-modifier function can be removed.

The second method of adding modifiers requires that the source code for FuzzyCLIPS be modified and the system be recreated with these new modifiers being available as system defined modifiers. This will allow for more complex modifiers to be added and for simple ones to be added with better performance. The only source file that should need to be changed is **fuzzymod.c**. Most modifiers can be added by simply following the code for system supplied modifiers that are in this file. In general, it is only necessary to do the following things.

1. Modify the function initFuzzyModifierList to identify the name of the modifier and it's "C" code function (example for the *very* modifier is given below).

```
AddFuzzyModifier("very", veryModFunction, NULL
#if DEFFUNCTION_CONSTRUCT
          , NULL
#endif
             );
```

2. Then the function definition is added (example for the *very* modifier is given below).

```
/*************************************************************/
/* veryModFunction:                                          */
/*                                                           */
/* implements the 'very' hedge                               */
/*                                                           */
/*      each element is squared (y**2)                       */
/*                                                           */
/*************************************************************/
static VOID veryModFunction(fv)
 struct fuzzy_value *fv;
{
 concentrate_dilute(fv, 2.0);
}
```

The concentrate-dilute function handles all of the similar types of modifiers that involve an exponentiation operation on the membership values. In this function there is a call to another function called Yexpand. This code determines if any straight line segments need to be expanded into further points so that the exponentiation operation will have the desired effect. For example, if the line segment was from (10 0) to (20 1) the squaring operation of the very modifier would just return these same two points, since 0**2 = 0 and 1**2 = 1. This would not reflect the shape properly. In essence, we try to keep a compact representation of the fuzzy set when we can but expand to more points when necessary.

The figure below demonstrates the effect of the Yexpand operation on a straight line segment.

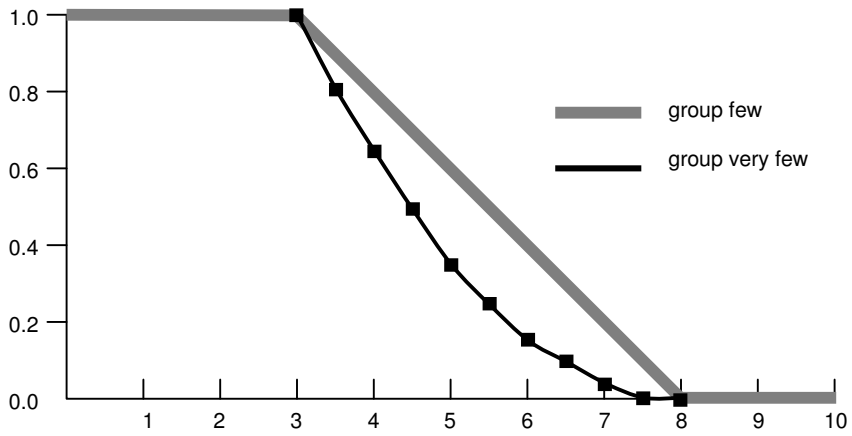


Figure 20: Modifier interpolation method (Yexpand)

If further assistance is required please contact NRC by e-mail.

6.3.3 Linguistic Expressions

The use of fuzzy primary terms and modifiers together with the binary operators *and* and *or* allow us to express the problem solutions in a more natural way. These expressions are called linguistic expressions. Expressions such as

temperature very hot or very cold

height below tall and above short

are examples of expressions that could be used to describe fuzzy variable. The BNF that describes formally the syntax of such expressions is shown below.

<LExpr> ::= <LTerm> | <LExpr> OR <LTerm>
<LTerm> ::= <modExpr> | <LTerm> AND <modExpr>
<modExpr> ::= MODIFIER <modExpr> | <element>
<element> ::= PRIMARY-TERM | [<LExpr>]

where

MODIFIER is a valid modifier (not, very, etc.)
PRIMARY-TERM is a term defined in a fuzzy deftemplate

Note that this gives AND higher precedence than OR and that modifiers are basically unary operators with the highest precedence. One can control the order of the expression evaluation through the use of brackets [and]. These brackets must be separated from other items by a space due to the nature of the CLIPS token parser.

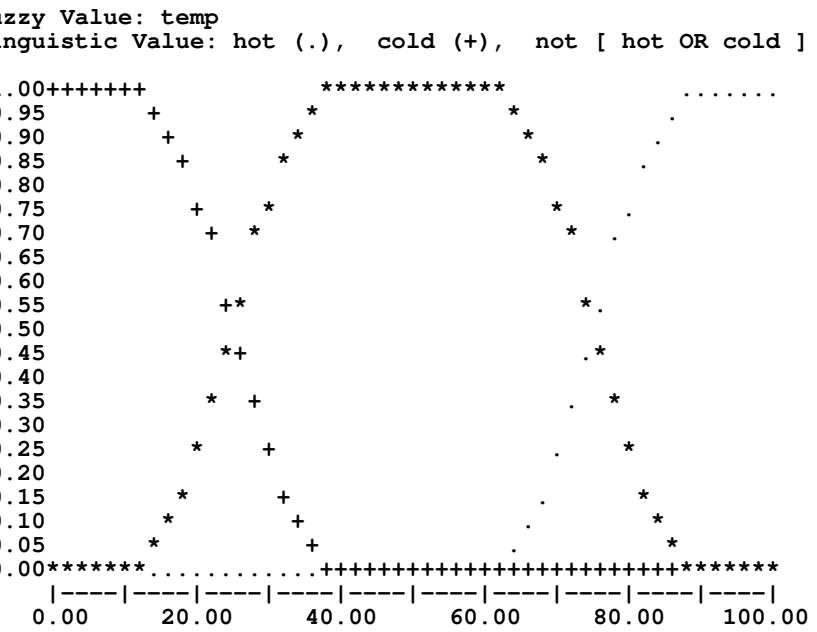
Therefore,

A or B and C or D

is the same as

A or [B and C] or D

The following graph shows an example of the fuzzy sets *temp hot*, *temp cold,* and *temp not [hot or cold]*.



6.4 Using Fuzzy Variables in LHS Patterns

A fuzzy LHS pattern is of the form

(fuzzy-variable-name <linguistic-expr>)

or

(fuzzy-variable-name ?)

or

(fuzzy-variable-name ?<var-name>)

or

(fuzzy-variable-name ?<var-name> & <linguistic-expr>)

or

(template-name <slot-description>$^{+}$)

where

+ indicates that there are one or more of the <slot-description> entries, at least one of these is a <fuzzy-slot-description>, and a <fuzzy-slot-description> is

(fuzzy-slot-name <linguistic-expr>)

or

(fuzzy-slot-name ?)

or

(fuzzy-slot-name ?<var-name>)

or

(fuzzy-slot-name ?<var-name> & <linguistic-expr>)

The <linguistic-expr> is a fuzzy set specified by a combination of primary terms, modifiers, and the logical operators NOT and OR as described in Section 5.3.3. A fuzzy-variable-name is the name of any fuzzy deftemplate. A template-name is the name of any non-fuzzy deftemplate. A fuzzy-slot-name is the name of a slot declared to have type FUZZY-VALUE (as described in Section 5.2).

The examples below show some of the ways in which fuzzy patterns might appear on the left-hand side of rules.

Example 24

```
(deftemplate group           ;declaration of fuzzy variable group
  0 20 members
     ((few (3 1) (6 0)) ;primary term few
      (many (4 0) (6 1));primary term many
     )
)

(defrule simple-LHS
     (group few)             ;a simple fuzzy LHS pattern
  ...
)
```

Example 25

```
(defrule more-complex-lhs
  ?f <- (group very few or very many)
 =>
  (printout t "We are at the extreme limits of the number of"
              (get-u-units ?f) " in our club" crlf)
)
```

Example 26

```
(deftemplate height
  0 8 feet
 ((short (Z 3 4.5))
  (medium (pi 0.8 5))
  (tall (S 5.5 6))
 )
)
```

```
(deftemplate person
  (slot name)
  (slot ht (type FUZZY-VALUE height))
)

(defrule quite-complex-lhs
  (person (name ?n) (ht ?h & very tall))
 =>
  (printout t ?n " is very tall, with a height of about "
              (maximum-defuzzify ?h) " "
              (get-u-units ?h) crlf )
)
```

In the last example ?h will become the fuzzy value that matches the *very tall* specification. It can then be used as an argument to various functions that can process fuzzy values, such as maximum-deffuzzify or get-u-units.

6.5 Using Fuzzy Variables in Deffacts Constructs

The syntax of the deffacts construct has been extended to allow fuzzy facts to be declared. Both fuzzy and non-fuzzy (crisp) facts can be declared in the same deffacts construct. Also certainty factors for crisp or fuzzy facts may be specified. Fuzzy fact specifications in a deffacts construct have the following form:

(deffacts <deffacts-name> [<comment>]
<RHS-pattern>*
)

where <RHS-pattern> has been extended as follows:

<RHS-pattern> ::= <ordered-RHS-pattern> |
<template-RHS-pattern> |
<fuzzy-template-RHS-pattern>

<ordered-RHS-pattern> ::= (<symbol> <RHS-field>$^{+}$)
[CF <certainty factor> | <certainty factor expression>]

<template-RHS-pattern> ::= (deftemplate-name> <RHS-slot>*)
[CF <certainty factor> | <certainty factor expression>]

<fuzzy-template-RHS-pattern> ::=
(<fuzzy-template-name> <description of fuzzy set>
[CF <certainty factor> | <certainty factor expression>]

and where

The certainty factor (CF) is optional (if not specified a CF of 1.0 is assumed).
The <description of fuzzy set> is
a <linguistic-expr> or
a <standard> (Section 6.1.1.2) fuzzy set specification or
a <singletons> (Section 6.1.1.1) fuzzy set description.

The ability to use global variables and function calls is as per standard deffacts statements. Also note that a RHS-slot may be a slot of type FUZZY-VALUE.

Example 27

```
(deffacts groupA "some fuzzy facts"
     (my_group (1 0) (5 1) (7 0))
     ;singleton description
     (your_group (z 4 8))           ;standard description
     (their_group (s (+ 1 1) 4))
     (person (name ralph) (ht tall))
)

(deffacts groupB "some fuzzy facts with certainty factors"
     (this_group (1 0) (5 1) (7 0)) CF 0.35
     (that_group (pi 2 (+ 3 4))) CF (+ .2 .3)
)
```

6.6 Using Fuzzy Variables in Assert Statements

The syntax of the assert construct has been expanded to allow fuzzy facts as arguments, and for the certainty factor of a crisp or fuzzy fact to be specified. The assert command (with a single fact asserted in an assert function call) is of the form

(assert
(<crisp fact> | fuzzy-variable-name <description of fuzzy set> |
template-name <slot-description>$^{+}$)
[CF <certainty factor> | <certainty factor expression>]
)

where

+ indicates that there are one or more of the <slot-description> entries, at least one of these is a <fuzzy-slot-description>, and a <fuzzy-slot-description> is

(fuzzy-slot-name <description of fuzzy set>)

<description of fuzzy set> is

<linguistic-expr> | <standard> | <singletons>

and the certainty factor is optional (if not specified a CF of 1.0 is assumed).

Some examples will illustrate the forms allowed for asserting fuzzy facts and fuzzy deftemplate facts.

Example 28

```
(assert (group few))
(assert (group (1 0) (5 1) (7 0)) )
(assert (group NOT [ very few OR many ] ))
(assert (group (z 4 8))  )
(assert (person (name john) (ht extremely tall)))
(assert (person (name dan) (ht (pi 0 5.6)))
(assert (temp (24 0) (25 1) (26 0)))
```

Example 29

```
(defrule assert-rule-1
     (zmin ?minval)      ;variable ?minval must be numeric
     (zmax ?maxval)      ;variable ?maxval must be numeric
  =>
     (assert
          (group (z ?minval ?maxval))
     )                   ;fuzzy set description with variables
)

(defrule assert-rule-2   ;asserts standard set with functions
     (zmin ?minval)
     (zmax ?maxval)
  =>
     (assert (group (z ?minval (+ ?maxval 2))))
)

(defrule assert-rule-3   ;asserts singleton set with functions
     (x1 ?x1val)
     (x2 ?x2val)
  =>
     (assert
          (group (?x1val 0) (?x2val 1) ((+ ?x2val 1) 0.6)
                      ((sqr ?x2val) 0))
     )
)
```

The ***assert-string*** function of CLIPS may also be used to assert fuzzy facts. The rules for constructing the strings are as described in the CLIPS reference manual.

Example 30

```
(assert-string "(group (z 4 8))")
(assert-string "(group (s 4 (+ 5 3)))")
(assert-string "(person (name bob) (ht medium))")
```

The ***<certainty factor>*** is a constant numeric value from 0.0 to 1.0.

Example 31

```
(assert (somefact) CF 0.8)         ;with CF of 0.8
(assert (group few) CF (+ 0.2 0.4));with CF of 0.6
```

Note that the certainty factor will be calculated as described in Section 4.3.5.

A ***<certainty factor expression>*** has the same syntax as a ***<certainty factor>***, except that the constant numeric value may be replaced by a variable or function which returns a numeric value between 0.0 and 1.0.

Example 32

```
( defrule assert-rule-4      ;illustrates various CF assertions
     (certainty-factor ?cf)  ;where ?cf is between 0 and 1
     ?f <- (somefact)
  =>
     (assert (fact1) CF ?cf)
     (assert (fact3) CF (* 0.8 ?cf))
     (assert (fact5) CF (get-cf ?f))
     ;get-cf is a function discussed in Section 5.9
)
```

6.7 Defuzzification

A crisp value may be extracted from a fuzzy set using either the centre of gravity or mean of maxima techniques developed in Section 4.4. The syntax is as follows:

(moment-defuzzify ?<*fact-var*> | integer | <fuzzy-value>) ; COG algorithm
(maximum-defuzzify ?<*fact-var*> | integer | <fuzzy-value>) ; MOM algorithm

The argument may be fact variable (?<fact-var>), which normally is bound on the LHS of a rule as described in the CLIPS Reference Manual. It may be the integer value of a fact number (i.e., the index of the fact on the fact list). It may also be a <fuzzy-value > that can be obtained (among other ways) by matching in the pattern of a fuzzy deftemplate fact or a FUZZY-VALUE slot.

These functions return a floating point number[16] which is the result of performing the defuzzification.

Example 33

```
Suppose that fact-1 (a fuzzy deftemplate fact) on the fact list is

     (temperature warm)

and that the COG method returns a value of 28, while the MOM method returns a
value of 32.5.


When the following defuzzification command is issued at the CLIPS prompt
level it will return the value 28.0 and display this value at the CLIPS
prompt level

     (moment-defuzzify 1)

When either of the following rules are executed it will print 'Temperature is
32.5'.


(defrule defuzzification-1
     ?f <- (temperature ?)                  ; ? used to assure match of the
                                            ; temperature fuzzy fact
  =>
     (bind ?t (maximum-defuzzify ?f))       ; get the value
     (printout t "Temperature is " ?t crlf) ; print 32.5
)
```

16 Moment-defuzzify may be undefined and will return by default the midpoint of the universe of discourse, or when undefined it will return the value set with the set-defuzzification-error-value command (see section 5.9.14)

```
(defrule defuzzification-2
     (temperature ?fv)            ; ?fv used to hold the fuzzy value
                                  ; of the matching fuzzy fact
     =>
     (printout t "Temperature is " (maximum-defuzzify ?fv) crlf)

)
```

6.8 Certainty Factors of Rules

This section will discuss the syntax for declaring the certainty factor of rules. The certainty factor of a rule may be declared in a manner similar to declaring the rule salience:[17]

```
(defrule some-rule
        (declare (CF <certainty factor>))
        ......
 =>
        ......
)
```

The <certainty factor> is a number in the range 0 to 1.

Example 34

```
(deffacts initial-facts
     (fact1) CF 0.8          ;fact with crisp CF of 0.8
)

(defrule some-other-rule
     (declare (CF 0.7))      ;a rule with CF of 0.7
     (fact1)
  =>
     (printout t "Hello!")
)
```

6.9 FuzzyCLIPS Commands and Functions

A number of commands supplied with CLIPS can be used to look at fuzzy facts and templates just as one looks at standard facts (see commands facts, ppdeftemplate, list-deftemplates, undeftemplate, ppdeffacts, list-deffacts, undeffacts). The following commands or functions add capability to access components of fuzzy facts, control thresholds for rule firings, set precision for fuzzy set display, set the fuzzy inference type, set a threshold for fuzzy pattern matching, create and manipulate fuzzy values, etc.

6.9.1 Accessing the Universe of Discourse (get-u, get-u-from, get-u-to, get-u-units)

Command: get-u

Syntax: (get-u ?<fact-var>) or

(get-u <integer>) or

(get-u <fuzzy-template-name>) or

[17] Note that the certainty factor may be a constant or an expression as is true of the salience value.

(get-u <fuzzy-value>)

where

?<fact-var> is a fact variable, normally bound on the LHS of a rule
<integer> is the number of a fact on the fact list (constant or expression)
<fuzzy-template-name> is a symbol that represents the name of a fuzzy deftemplate
<fuzzy-value> is an element of type FUZZY-VALUE

Purpose: Returns a string of the form: "<from> - <to> <units>", the limits of the universe of discourse and the units (if specified). If no units have been declared in the deftemplate statement, then the function returns "<from> - <to>".

Example 35

```
(get-u ?t)        ;; ?t is bound to the temp fuzzy fact
(get-u 2)         ;; 2 is a fact index
(get-u temp)      ;; temp is the name of the fuzzy deftemplate

(defrule test
  (temp ?fv)      ;; ?fv is a fuzzy value
 =>
  (printout t (get-u ?fv))
)
```

Command: get-u-from

Syntax: (get-u-from ?<fact-var>) or

(get-u-from <integer>) or

(get-u-from <fuzzy-template-name>) or

(get-u-from <fuzzy-value>)

Purpose: Returns the lower limit of the universe of discourse in floating point format.

Command: get-u-to

Syntax: (get-u-to ?<fact-var>) or

(get-u-to <integer>) or

(get-u-to <fuzzy-template-name>) or

(get-u-to <fuzzy-value>)

Purpose: Returns the upper limit of the universe of discourse in floating point format.

Example 36

```
;; Deffunction fuzzify
;;
;; Inputs:   ?fztemplate  - name of a fuzzy deftemplate
;;           ?value       - float value to be fuzzified
;;           ?delta       - precision of the value
;;
;; Asserts a fuzzy fact for the fuzzy deftemplate. The fuzzy set
;; is a triangular shape centered on the value provided with zero
;; possibility at value+delta and value-delta. Note that it
;; checks bounds of the universe of discourse to generate a fuzzy
;; set that does not have values outside of the universe range.

(deffunction fuzzify (?fztemplate ?value ?delta)

  (bind ?low (get-u-from ?fztemplate))
  (bind ?hi  (get-u-to   ?fztemplate))

  (if (<= ?value ?low)
    then
      (assert-string
         (format nil "(%s (%g 1.0) (%g 0.0))"
                 ?fztemplate ?low ?delta))
    else
      (if (>= ?value ?hi)
        then
          (assert-string
             (format nil "(%s (%g 0.0) (%g 1.0))"
                     ?fztemplate (- ?hi ?delta) ?hi))
        else
          (assert-string
             (format nil "(%s (%g 0.0) (%g 1.0) (%g 0.0))"
                     ?fztemplate (max ?low (- ?value ?delta))
                     ?value (min ?hi (+ ?value ?delta)) ))))
)

As an example the following uses the fuzzify function:
```

```
(deftemplate temp
   0 100 Degrees-F
   ( (warm (30 0) (60 1) (90 0))
   )
)

(defrule test
   (temp warm)
 =>
   (bind ?x (assert (dummy)))
   (printout t "Certainty Factor = " (get-cf ?x) crlf)
   (retract ?x)
)
```

Asserting a fuzzified fact with

```
(fuzzify temp 50 0.001)
```

and firing the rule 'test' will result in the output:

```
   Certainty Factor = 0.66667
```

Command: get-u-units

Syntax: (get-u-units ?<fact-var>) or
(get-u-units <integer>) or
(get-u-units <fuzzy-template-name>) or
(get-u-units <fuzzy-value>)

Purpose: Returns the units of the universe of discourse in string format. If no units have been specified, then the empty string " " is returned.

6.9.2 Accessing the Fuzzy Set (get-fs, get-fs-x, get-fs-y, get-fs-length, get-fs-lv, get-fs-value)

Command: get-fs

Syntax: (get-fs ?<fact-var>) or
(get-fs <integer>) or
(get-fs <fuzzy-value>)

Purpose: Returns the entire fuzzy set in singleton representation, in string format.

Command: get-fs-length

Syntax: (get-fs-length ?<fact-var>) or
(get-fs-length <integer>) or
(get-fs-length <fuzzy-value>)

Purpose: Returns the number of pairs in a fuzzy set description as an integer.

Command: get-fs-x

Syntax: (get-fs-x ?<fact-var> <i>) or

(get-fs-x <integer> <i>) or

(get-fs-x <fuzzy-value> <i>)

where <i> is an integer, variable, or function expression.

Purpose: Returns the x-coordinate of the ith pair in the fuzzy set, where the pairs are numbered left to right from 0 to (n-1) and n is the total number of pairs in the set. If the expression <*i*> evaluates to a non-integer value, then it is truncated to the nearest integer. The x-coordinate is returned as a floating point value.

Command: get-fs-y

Syntax: (get-fs-y ?<fact-var> <i>) or

(get-fs-y <integer> <i>) or

(get-fs-y <fuzzy-value> <i>)

Purpose: Returns the y-coordinate of the ith pair in the fuzzy set as a floating point value.

Example 37

```
Suppose the fuzzy fact

     (temperature ???)

was fact-2 on the fact list (the ??? indicates that the fuzzy set
is not expressible using terms and modifiers for that fuzzy
variable; it may have been defined using a singleton description
or modified by the compositional rule of inference). Suppose
also that it has a fuzzy set consisting of the following three
singletons

     ((0 0) (25 1) (40 0)).

Then (get-fs-x 2 1) at the command line would return 25.
In a rule one could do the following:

(defrule print-last-coordinate
     ?f <- (temperature ?)
     =>
     (bind ?n (get-fs-length ?f))
     (bind ?x (get-fs-x ?f (- ?n 1)))
     (bind ?y (get-fs-y ?f (- ?n 1)))
     (printout t "The last point in the set is:" ?x "," ?y crlf)
)

    OR
```

```
(defrule print-last-coordinate
      (temperature ?fv)
      =>
      (bind ?n (get-fs-length ?fv))
      (bind ?x (get-fs-x ?fv (- ?n 1)))
      (bind ?y (get-fs-y ?fv (- ?n 1)))
      (printout t "The last point in the set is:" ?x "," ?y crlf)
)
```

Command: get-fs-lv

Syntax: (get-fs-lv ?<fact-var>) or

(get-fs-lv <integer>) or

(get-fs-lv <fuzzy-value>)

Purpose: Returns the returns the linguistic value associated with the fuzzy set.

Example 38

```
if (temp very hot) is asserted and the variable ?fuzzyfact is assigned to
this fact then the function call


      (get-fs-lv ?fuzzyfact)


would return the string "very hot"
```

Command: get-fs-value

Syntax: (get-fs-value ?<fact-var> <number>) *or*

(get-fs-value <integer> <number>) *or*

(get-fs-value <fuzzy-value> <number>)

Purpose: Returns the returns the value of the fuzzy set at the specified x value (<number>). The <number> is a value that must lie between the lower and upper limits of the universe of discourse for the fuzzy set.

Example 39

```
Suppose we have defined the following


(deftemplate temp
  0 100 C
  (
     ...
    (OK (30 0) (60 1) (90 0))
     ...
  )
)


and we assert the fact (temp OK). Then if we bind that fact
to a variable ?fact we could call
```

```
(get-fs-value ?fact 50.0)
```

and it would return 0.6666667

6.9.3 Accessing the Certainty Factor (get-cf)

Command: get-cf

Syntax: (get-cf ?<fact-var>)

(get-cf <integer>) or

Purpose: Returns the certainty factor of a fact as a floating point number.

Example 40

Suppose a fact on the fact list is *(temperature cold) CF 0.7.*

```
(defrule print-cf-rule
   ?f <- (temperature ?)
 =>
  (printout t "The certainty of the temperature measurement is: "
     (get-cf ?f))
)
```

Then the above rule will print
"The certainty of the temperature measurement is: 0.7".

6.9.4 Enabling and Disabling Certainty Factor Calculations in Rules (enable-cf-rule-calculation, disable-cf-rule-calculation)

These commands control the certainty factors are calculated for facts asserted within rule executions. If the calculation is disabled the facts are assigned the certainty specified in the fact assertion (or 1.0 if not specified). If enabled then the calculations for certainty factors of facts described in sections 5.2 and 5.3 are applied.

Command: enable-cf-rule-calculation

Syntax: (enable-cf-rule-calculation)

Purpose: After executing this command any facts asserted in a rule will use the appropriate calculations to determine the certainty value for the fact.

Command: disable-cf-rule-calculation

Syntax: (disable-cf-rule-calculation)

Purpose: After executing this command any facts asserted in a rule will always have the certainty value specified for the fact and if none is specified it will have a value of 1.0. The certainty value of the rule or the matched facts will not be considered.

6.9.5 Accessing the Threshold Certainty Factor (threshold, get-threshold)

Command: set-threshold[18]

Syntax: (set-threshold <NUMBER>)

Purpose: Sets threshold certainty factor to the value of <NUMBER>.
<NUMBER> must evaluate to a floating value between 0.0 and 1.0.
By default the threshold value is 0.0.

Command: get-threshold

Syntax: (get-threshold)

Purpose: Returns the floating point value of the threshold certainty factor if threshold capability is ON. If it is OFF, then a value of 0.0 is returned.

6.9.6 Setting the Rule CF Evaluation Behaviour (set-CF-evaluation, set-CF-evaluation)

Command: set-CF-evaluation

Syntax: (set-CF-evaluation <value>)

Purpose: Sets the behavior for evaluating the CF of rules to <value>.
Value must be one of when-defined (default) or when-activated. This is similar to the set-salience-evaluation function of CLIPS. The value when-defined forces the certainty factor of the rule to be evaluated at the time of rule definition (compilation). The value when-activated forces the certainty factor of the rule to be defined at the time of rule definition and when the rule is activated (added to the agenda).

Command: get-CF-evaluation

Syntax: (get-CF-evaluation)

Purpose: Returns the current setting of the behavior for evaluating the CF of rules. Return value is either when-defined or when-activated (similar to the get-salience-evaluation function).

6.9.7 Controlling the Fuzzy Set Display Precision (set-fuzzy-display-precision, get-fuzzy-display-precision)

Command: set-fuzzy-display-precision

Syntax: (set-fuzzy-display-precision <integer>)

Purpose: When fuzzy facts are displayed the fuzzy set values are displayed in floating point format. This function allows the number of significant digits displayed after the decimal point to be set. The <integer> argument is an integer value between 2 and 16. If it is less than 2 it is set to 2 and if it is greater than 16 it is set to 16. The default value is 4. Note that *clear* will not reset this value to 4.

---

[18] For compatibility with previous versions of FuzzyCLIPS *threshold* is also accepted.

Example 41

```
(set-fuzzy-display-precision 16)
(facts)
   f-0  (speed_error more_or_less large_positive) CF 1.00
        ( (0.0 0.0)
          (0.1 0.3162277660168379) (0.2 0.4472135954999579)
          (0.3 0.5477225575051661) (0.35 0.5916079783099616)
          (0.4 0.6324555320336759) (0.5 0.7071067811865476)
          (0.6 0.7745966692414834) (0.7 0.8366600265340756)
          (0.8 0.8944271909999159) (0.9 0.9486832980505138)
          (1.0 1.0)  )

(set-fuzzy-display-precision 2)
(facts)
   f-0  (speed_error more_or_less large_positive) CF 1.00
        ( (0.0 0.0)
          (0.1 0.32) (0.2 0.45)
          (0.3 0.55) (0.35 0.59)
          (0.4 0.63) (0.5 0.71)
          (0.6 0.77) (0.7 0.84)
          (0.8 0.89) (0.9 0.95)
          (1.0 1.0)  )
```

Command: get-fuzzy-display-precision

Syntax: (get-fuzzy-display-precision)

Purpose: Returns an integer value that is the current display precision.

6.9.8 Controlling the Fuzzy Inference Method (set-fuzzy-inference-type, get-fuzzy-inference-type)

Command: set-fuzzy-inference-type

Syntax: (set-fuzzy-inference-type <inf-type>)

Purpose: Sets the current inference type to one of *max-min* or *max-prod*. The default is *max-min*. The effect of this is described in more detail in Section 4.3.1.3. Note that *clear* will not reset this value to *max-min*.

Command: get-fuzzy-inference-type

Syntax: (get-fuzzy-inference-type)

Purpose: Returns a symbol that is one of *max-min* or *max-prod* indicating the current inference type.

6.9.9 Setting the Fuzzy Pattern Matching Threshold (set-fuzzy-inference-type, get-fuzzy-inference-type)

Command: set-alpha-value

Syntax: (set-alpha-value <alpha-val>)

Purpose: When fuzzy slots are matched to fuzzy patterns on the LHS of rules, the match is considered to be successful if there is any overlap (intersection) between the two fuzzy sets involved in the

matching. This function allows the match to succeed only if the maximum of the intersection set has a membership value greater than or equal to this threshold. The default alpha-value is 0.0. When the alpha-value is 0.0 the maximum of the intersection set must be greater than 0.0. Note that a *clear* does not reset the alpha-value to 0.0.

Example 42

```
(deftemplate temp
  0 100 C
  ( (low (10 1) (50 0))
    (ok  (20 0) (50 1) (80 0))
    (high (50 0) (90 1))
  )
)
(defrule test-alpha
   (temp low)
  =>
   (printout t "Rule fired ****" crlf)
)

(set-alpha-value 0.0)
(assert (temp (pi 0 30)))
(run) ; rule should fire with alpha 0.0
Rule fired ****
(retract *)
(set-alpha-value 0.5)
(assert (temp (pi 0 30)))
(run) ; rule should fire with alpha 0.5
Rule fired ****
(retract *)
(set-alpha-value 0.55)
(assert (temp (pi 0 30)))
(run) ; rule should NOT fire with alpha 0.55
;; no output here -- match was not successful
```

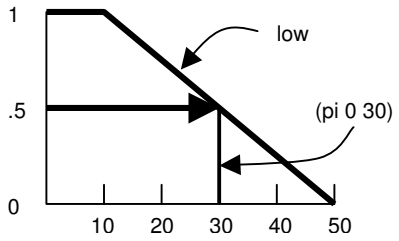


Figure 21

Command: get-alpha-value

Syntax: (get-alpha-value)

Purpose: Returns a floating point value which is the current alpha-value.

6.9.10 Fuzzy Value Predicate Function (fuzzvaluep)

Command: fuzzyvaluep

Syntax: (fuzzyvaluep <arg>)

Purpose: This function returns TRUE if the argument is of type FUZZY-VALUE, otherwise it will return FALSE.

Example 43

```
(fuzzyvaluep 45.6)
FALSE
(fuzzyvaluep "string")
FALSE
(fuzzyvaluep (create-fuzzy-value temp cold))
TRUE

(defrule check-fuzzyvaluep
  (temp ?fv & cold)
 =>
  (fuzzyvaluep ?fv)
)
(assert (temp cold))
(run)
TRUE
```

6.9.11 Creating and Operating on FUZZY-VALUEs (create-fuzzy-value, fuzzy-union, fuzzy-intersection, fuzzy-modify)

Command: create-fuzzy-value

Syntax: (create-fuzzy-value <fuzzy-deftemplate-name> <description of fuzzy set>)

Purpose: This function allows a fuzzy value to be created. A fuzzy value is a fuzzy set that is associated with a particular fuzzy deftemplate. The fuzzy deftemplate determines the universe of discourse for the fuzzy set and the terms that can be used to describe the fuzzy set. The first argument, <fuzzy-deftemplate-name>, is the name of a fuzzy deftemplate. The remaining parts describe the fuzzy set as is done for a fuzzy slot when a fuzzy fact is asserted. This can be a linguistic expression, a singleton specification, or a standard function expression (see Section 6.1).

Example 44

```
(create-fuzzy-value temp cold)
(create-fuzzy-value temp very hot or very cold)
(create-fuzzy-value temp (pi 10 20))
(create-fuzzy-value temp (s ?x (+ ?x 10)))
(create-fuzzy-value temp (10 1) (20 0))

(defrule test
  =>
    (bind ?fv (create-fuzzy-value temp cold))
    (assert (temp ?fv)) ; NOTE use of variable here!
)
```

Command: fuzzy-union

Syntax: (fuzzy-union <fuzzy-value> <fuzzy-value>)

Purpose: Returns a new fuzzy value that is the union of two other fuzzy values. Both arguments must be of type FUZZY-VALUE.

Example 45

```
(deftemplate temp
   0 100 C
  ((cold (z 20 70))
   (hot (s 30 80))
  )
)
(fuzzy-union (create-fuzzy-value temp cold)
             (create-fuzzy-value temp hot))
cold or hot

(plot-fuzzy-value t ".+*" nil nil
      (create-fuzzy-value temp cold)
      (create-fuzzy-value temp hot)
      (fuzzy-union (create-fuzzy-value temp cold)
                   (create-fuzzy-value temp hot))
)

Fuzzy Value: temp
Linguistic Value: cold (.),  hot (+),  [ cold ] OR [ hot ] (*)
 1.00*************                          *************
 0.95             **                      **
 0.90               **                  **
 0.85                 *                *
 0.80                  *              *
 0.75                   *            *
 0.70                    *          *
 0.65
 0.60                     *        *
 0.55                      *      *
 0.50
 0.45                       *    *
 0.40                        * *
 0.35                         +
 0.30                         *
 0.25                        + .
 0.20                       +   .
 0.15                      +     .
 0.10                    ++       ..
 0.05                   ++           ..
 0.00++++++++++++++++++               ..................
     |----|----|----|----|----|----|----|----|----|----|
   0.00     20.00     40.00     60.00     80.00    100.00
```

Command: fuzzy-intersection

Syntax: (fuzzy-intersection <fuzzy-value> <fuzzy-value>)

Purpose: Returns a new fuzzy value that is the intersection of two other fuzzy values. Both arguments must be of type FUZZY-VALUE.

Example 46

```
(fuzzy-intersection (create-fuzzy-value temp cold)
(create-fuzzy-value temp hot))
cold and hot

(plot-fuzzy-value t ".+*" nil nil
      (create-fuzzy-value temp cold)
      (create-fuzzy-value temp hot)
      (fuzzy-intersection (create-fuzzy-value temp cold)
                          (create-fuzzy-value temp hot))
)

Fuzzy Value: temp
Linguistic Value: cold (.),  hot (+),  [ cold ] AND [ hot ] (*)
 1.00.............                          ++++++++++++++
 0.95              ..                      ++
 0.90                ..                  ++
 0.85                  .                +
 0.80                   .             +
 0.75                    .           +
 0.70                     .         +
 0.65
 0.60                       .       +
 0.55                        .     +
 0.50
 0.45                          .   +
 0.40                           . +
 0.35                            +
 0.30                            *
 0.25                           * *
 0.20                          *   *
 0.15                         *     *
 0.10                       **       **
 0.05                     **           **
 0.00******************               ******************
     |----|----|----|----|----|----|----|----|----|----|
    0.00     20.00     40.00     60.00     80.00    100.00
```

Command: fuzzy-modify

Syntax: (fuzzy-modify <fuzzy-value> <modifier>)

Purpose: Returns a new fuzzy value that is a modification of the fuzzy value argument. The modification performed is specified by the modifier argument. This modifier can be any active modifier (very, slightly, etc.).

Example 47

```
(plot-fuzzy-value t "+*" nil nil
(create-fuzzy-value temp hot)
(fuzzy-modify (create-fuzzy-value temp hot) extremely))

Fuzzy Value: temp
Linguistic Value: hot (+),  extremely hot (*)
 1.00                                        ++++++++++++
 0.95                                      ++**
 0.90                                    ++ *
 0.85                                   +  *
 0.80                                  +
 0.75                                 +   *
 0.70                                +   *
 0.65
 0.60                               +   *
 0.55                              +
 0.50                                  *
 0.45                             +
 0.40                            +    *
 0.35                           +
 0.30                                *
 0.25                          +
 0.20                         +     *
 0.15                        +     *
 0.10                      ++     *
 0.05                    ++     **
 0.00************************
     |----|----|----|----|----|----|----|----|----|----|
    0.00     20.00     40.00     60.00     80.00    100.00
```

6.9.12 Accessing a Fuzzy Slot in a Fact (get-fuzzy-slot)

Command: get-fuzzy-slot

Syntax: (get-fuzzy-slot ?<fact-var> [<slot-name>])

(get-fuzzy-slot <integer> [<slot-name>])

Purpose: This function will retrieve the fuzzy value associated with a fuzzy slot in a fact. The first argument can be a variable that is associated with a fact address or an integer that is the fact number for a fact. If the fact is a fuzzy deftemplate fact (one whose relation name is a fuzzy deftemplate name) then the second argument is not needed since the only slot for the fact is the fuzzy value. If the fact is a standard deftemplate fact with fuzzy slots, then the second argument is a symbol that identifies the slot to access. (Note that the slot of fuzzy deftemplate facts is always name 'GenericFuzzySlot' and it could be accessed using that name.)

Example 48

```
(defrule test1-get-fuzzy-slot
  ?f <- (temp hot)
 =>
  (plot-fuzzy-value t * nil nil (get-fuzzy-slot ?f))
)

(defrule test2-get-fuzzy-slot
  ?f <- (system (name sysA) (t-outflow hot))
 =>
  (plot-fuzzy-value t * nil nil (get-fuzzy-slot ?f t-outflow))
)
```

6.9.13 Displaying a Fuzzy Value in a Format Function

This is not a function or command but is an addition to CLIPS to allow the formatting of fuzzy values in a format function. The specifier %F is used.

Example 49

```
(deftemplate temp
   0 100 C
 ((cold (z 20 50)))
)
(assert (temp cold))
<Fact-0>
(format t "Value is '%F'%n" (get-fuzzy-slot 0))
Value is 'cold'
```

6.9.14 Plotting a Fuzzy Value (plot-fuzzy-value)

Command: plot-fuzzy-value

Syntax: (plot-fuzzy-value <logicalName> <plot-chars> <low-limit> <high-limit> <fuzzy-value>$^{+}$)

Purpose: This function is used to plot fuzzy sets. The arguments are:

<logicalName> is any open router to direct the output to (e.g. t for the standard output)

<plot-chars> specifies the characters to be used in plotting (e.g., * or "*+.")

- for each fuzzyvalue specified a corresponding character from the string or symbol is used as the plotting symbol -- if more fuzzyvalues than symbols are specified then the last symbol is used for the remaining plots

<low-limit> is a numeric value that specifies the lowest x value to be displayed in the plot OR if it is not a numeric value then it will default to the low limit of the universe of discourse

<high-limit> is a numeric value that specifies the highest x value to be displayed in the plot OR if it is not a numeric value then it will default to the high limit of the universe of discourse

<fuzzy-value> is one of three things

- an integer that identifies a fuzzy deftemplate fact (in this case the fuzzy value from the fact is extracted and used)

- a variable with a fuzzy deftemplate fact address (in this case the fuzzy value from the fact is extracted and used)

- a variable with a fuzzy value

The + identifies that one or more <fuzzy-value> arguments may be present.

The fuzzy deftemplate associated with ALL fuzzy values to be plotted on the same plot must be the same one. This is required since the x axis must have the same meaning.

The <high-limit> and <low-limit> values allow a *window* of the universe of discourse to be displayed and provides for scaling the graph in the x axis.

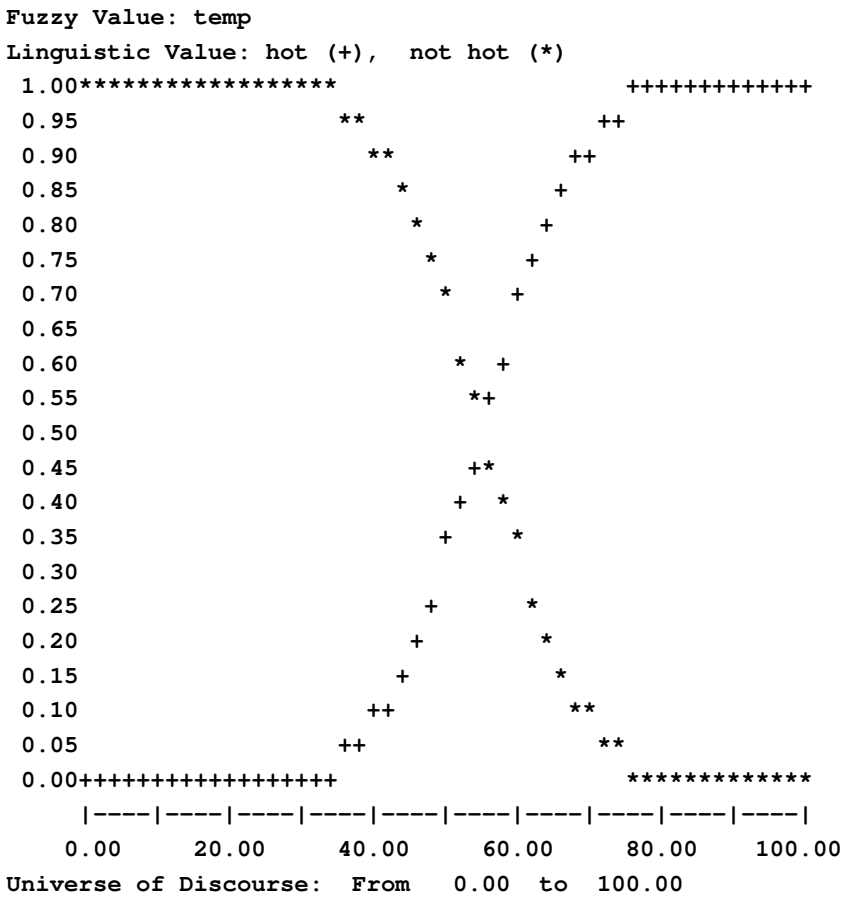
Example 50

```
(deftemplate temp
   0 100 C
  ((cold (z 20 70))
   (hot (s 30 80))
  )
)

(plot-fuzzy-value t * nil nil (create-fuzzy-value temp hot))

Fuzzy Value: temp
Linguistic Value: hot (*)
 1.00                                        *************
 0.95                                      **
 0.90                                    **
 0.85                                   *
 0.80                                  *
 0.75                                 *
 0.70                                *
 0.65
 0.60                               *
 0.55                              *
 0.50
 0.45                             *
 0.40                            *
 0.35                           *
 0.30
 0.25                         *
 0.20                        *
 0.15                       *
 0.10                     **
 0.05                   **
 0.00*******************
     |----|----|----|----|----|----|----|----|----|----|
    0.00     20.00     40.00     60.00     80.00    100.00
Universe of Discourse:  From   0.00  to  100.00
```

```
(plot-fuzzy-value t * 20 70  (create-fuzzy-value temp hot))

Fuzzy Value: temp
Linguistic Value: hot (*)
 1.00
 0.95
 0.90                                                  ***
 0.85                                                **
 0.80                                              **
 0.75                                            **
 0.70                                          **
 0.65                                         *
 0.60                                        *
 0.55                                      **
 0.50                                     *
 0.45                                   **
 0.40                                  *
 0.35                                **
 0.30                               *
 0.25                             **
 0.20                           **
 0.15                         **
 0.10                       ***
 0.05                 *****
 0.00***************
     |----|----|----|----|----|----|----|----|----|----|
   20.00     30.00     40.00     50.00     60.00     70.00
Universe of Discourse:  From   0.00  to  100.00
```

Example 51

```
(deftemplate system
     (slot name)
     (slot t-outflow (type FUZZY-VALUE temp))
)

(assert (system (name sysA) (t-outflow not hot)))
<Fact-1>

(plot-fuzzy-value t "+*" nil nil
     (create-fuzzy-value temp hot)
     (get-fuzzy-slot 1 t-outflow)
)
```



6.9.15 Controlling the Result of Defuzzification

The defuzzifcation functions are defined to return values that depend on the fuzzy value that they are provided with. The moment-defuzzify function (see section 5.4.1) is undefined in the case of a fuzzy set with area equal to zero. This can occur when the fuzzy set is a crisp value or the fuzzy set is flat at membership value of 0.0. In these cases, by default, the function will return the midpoint of the universe of discourse. To allow a user to detect these situations, a special function is provided.

Command: is-defuzzify-value-valid

Syntax: (is-defuzzify-value-valid)

Purpose: This function is used to check if the value returned by the last defuzzify function is valid or not. It returns TRUE if a valid value was be returned or FALSE if not. In the current implementation only the moment-defuzzify function can return an invalid default value.

If no defuzzify functions have yet been called, the return value is TRUE.

Normally one will use this function immediately after the defuzzify function is called. For example:

Example 52

```
(defrule defuzzify-temperature
   ?f <- (temperature ?)
 =>
   (bind ?temperature-value (moment-defuzzify ?f))
   (if (is-defuzzify-value-valid)
      then
          ... do something with the defuzzified value
      else
          ... perhaps use the maximum-defuzzify function
   )
)
```

6.10 Simple Example

This section describes the output of watching facts and rules fire for the example found in the file simplTst.clp. Please note that this was not intended to be a real example of the use of FuzzyCLIPS. For that see the other examples included in the distribution.

```
CLIPS>(load "simplTst.clp")
Defining deftemplate: speed_error
Defining deftemplate: speed_change
Defining deffacts: my_facts
Defining defrule: speed-too-fast +j
Defining defrule: speed-ok +j
Defining defrule: get-crisp-value-and-print-rslt +j
TRUE
CLIPS> (reset)
==> f-0     (initial-fact) CF 1.00
==> f-1     (speed_error zero) CF 0.90   ;linguistic description of fuzzy set
       ( (0.0 1.0) (0.11 0.0)  )         ;singletons describe fuzzy set in detail
==> Activation 0      speed-ok: f-1
==> Activation 0      speed-too-fast: f-1
CLIPS> (run)
FIRE    1 speed-too-fast: f-1
==> f-2     (speed_change ???) CF 0.63   ;CF = 0.9 * 0.7 for fuzzy-fuzzy rule
       ( (0.1 0.0) (0.1495 0.0991)  )
==> Activation -1     get-crisp-value-and-print-rslt: f-2
FIRE    2 speed-ok: f-1
<== f-2     (speed_change ???) CF 0.63   ;retraction of fuzzy fact and ...
       ( (0.1 0.0) (0.1495 0.0991)  )
<== Activation -1     get-crisp-value-and-print-rslt: f-2
==> f-3     (speed_change ???) CF 0.63   ;reassertion as fact is modified
       ( (0.0 1.0) (0.1 0.1) (0.1333 0.06667) (0.1495 0.0991)  )
==> Activation -1     get-crisp-value-and-print-rslt: f-3
FIRE    3 get-crisp-value-and-print-rslt: f-3

Change speed by a factor of:  0.3553202565269306

3 rules fired        Run time is 0.06400000000212458 seconds.
46.87499999844391 rules per second.
3 mean number of facts (3 maximum).
1 mean number of instances (1 maximum).
1 mean number of activations (2 maximum).
CLIPS>
```

7 Continuous Systems

This section describes further extensions made to CLIPS to take care of the needs of continuously operating systems.

7.1 The Run Command

Normally, CLIPS terminates when the agenda is empty. For real-time systems (or any continuously operating system) there is need for a mechanism that allows the inference engine to idle, waiting for events to occur. In FuzzyCLIPS the run command is extended to receive any of the following parameters:

n	(a positive integer) FuzzyCLIPS will run until n rules have executed or until the agenda is empty, whichever comes first, e.g., (run 10)
-1	FuzzyCLIPS runs until the agenda is empty, e.g., (run -1)
-2	FuzzyCLIPS runs forever. Control-C interrupts the execution, e.g., (run -2)
-n	(a negative integer less than -2). FuzzyCLIPS runs until n rules have executed, e.g., (run -10)

The *halt* function can be called at any time to terminate the run.

7.2 Runstart and Runstop Functions

CLIPS allows users to call external functions that are executed at the end of each cycle of the inference engine (i.e., after each rule firing). This is done by calling the AddRunFunction routine of CLIPS to include the function in the list of exec functions. In certain cases, however, it is useful to be able to execute special routines on entry or exit from the run command. The runstart and runstop functions of FuzzyCLIPS allow this. This could be useful in situations where a simulated clock is used to keep track of time. When the system is stopped (with (run -n) or control-C), one would want the simulated clock to stop too. When the system is resumed, one would want the clock to resume from where it left off when the system was stopped (i.e., without advancing during the stopped interval).

A function is added to the list of functions called when the run command is executed by calling the AddRunStartFunction. It can be removed from this list by calling the RemoveRunStartFunction. Similarly, a function is added to the list of functions called when the run command is terminated by calling the AddRunStopFunction. It can be removed from this list by calling the RemoveRunStopFunction. (Note: these external functions must have been previously defined as user functions.)

The following are examples of calls to these four functions:

```
AddRunStopFunction("haltTimer",haltTimer,1);
AddRunStartFunction("continueTimer",continueTimer,1);
RemoveRunStopFunction("haltTimer");
RemoveRunStartFunction("continueTimer");
```

The AddRunStopFunction and AddRunStartFunction functions have three arguments: a string name of the function to be added, a pointer to a function to be executed, and a priority for the function.

8 CLIPS Functionality within FuzzyCLIPS

8.1 Modifying and Duplicating Facts

The CLIPS functions modify and duplicate are different for fuzzy facts. These functions will always return FALSE when used with deftemplate fuzzy facts (this is as for standard CLIPS facts whose relation name is not a deftemplate name). When we have a fuzzy fact (CLIPS fact with fuzzy slots) the behaviour for modify is the same as a normal modify, the existing fact is retracted and the new fuzzy fact is asserted. However, for duplicate the fuzzy slots in the fuzzy fact are aggregated (global contribution) creating a new fact, the existing fact is retracted and the new fact is asserted. There is no duplicate fact created. This happens even if fact-duplication is turned off.

8.2 Load, Save, Bload, Bsave, Load-facts, Save-facts

The CLIPS functions load, save, bload, bsave, load-facts, and save-facts should all work correctly with FuzzyCLIPS programs that include fuzzy facts, fuzzy deftemplates, and certainty factors.

8.3 Constructs-to-c

Creating a runtime version of a FuzzyCLIPS program can be done as for a normal CLIPS program using the constructs-to-c function and following the instructions in the CLIPS manuals.

8.4 CreateFact, GetFactSlot, PutFactSlot

Functions defined in the advanced user's guide like CreateFact, GetFactSlot, and PutFactSlot will be usable with FuzzyCLIPS; however, the exact details for creating and accessing Fuzzy Values will require some further detailed knowledge of the internal structures of FuzzyCLIPS. Contact NRC if these are needed.

Other functions to support embedded applications and user extensions to FuzzyCLIPS may be needed and suggestions will be entertained. Contact NRC with details.

9 Limitations and Future Work

The current version of FuzzyCLIPS has been tested with some diligence but there may be omissions or errors that are discovered. Future extensions/modifications to FuzzyCLIPS will depend on user feedback. As experience is gained using FuzzyCLIPS, it may be that users will find things that work in unexpected or undesirable ways. Please feel free to comment on such things in the interest of promoting a useful tool.

One example of an area that might provoke some discussion is the maximum-defuzzify function that is used to defuzzify fuzzy facts. As pointed out in Section 4.4.2 this is a situation that is difficult to decide how best to handle. One solution is to perform the average of all discrete maximum values (as we have done in this implementation). Perhaps a better method would be to consider points that define a continuous range of x values all at the maximum value to be represented by the mid-point of that range plus a weight equal to the difference between the upper and lower x values of that range. A single point maximum would be represented by the x value plus a weight of zero. Then the defuzzified value would be calculated as:

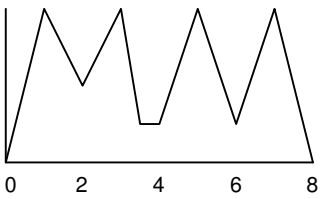
1. if weights are all zero

$$\frac{\sum x_{max}}{n}$$

2. if not all weights are zero

$$\frac{\sum x_{max} \times weight}{\sum weight}$$

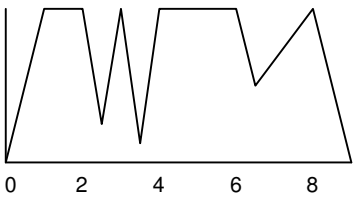
Consider the following two examples:



max values at 1, 3, 5, and 7
(all weights 0)

Calculate defuzzified value as:
(1+3+5+7)/4 = 4.0

same as current method



max values at 1.5 with weight 1, 3 with weight 0
5 with weight 2 and 8 with weight 0

Calculate defuzzified value as:
(1.5*1+3*0+5*2+8*0)/(1+0+2+0) = 3.833

current method would give
(1+2+3+4+6+8)/6 = 4.0

Another possible source of discussion is the calculation of certainty factors. This can be quite difficult to understand, especially when complex rules are used. Could this be simplified and still retain the intensions as described in this document?
A third possible problem area is the handling of truth maintenance via the logical construct in rules. Consider the following 2 rules:

where temp is a fuzzy deftemplate (fuzzy variable). If both of these rules are fired then the temp fact will exist as

(temp low OR medium)

due to global contribution. If both facts (x) and (y) are retracted then the logical support for the temp fact will be lost and the fact will be retracted as expected. However, when one of the logically supporting facts is retracted the other still supports the fact and it remains asserted as (temp low OR medium). The effect of global contribution is not undone. For example, should the fact (x) be retracted then perhaps the effect of the assertion (temp low) should be undone leaving the fact as (temp medium). This is difficult to do without retaining a lot of extra information to assist in removing the effect of that assertion. However, it would be possible to do this in another way. Instead of applying the effects of global contribution as fuzzy facts are asserted it could be delayed until required. That is when the two rules fire two facts are asserted.

(temp low) with logical dependence on fact (x)
(temp medium) with logical dependence on fact (y)

This is similar to standard clips facts being asserted with the same template name but different content in the slots. If the fact (x) is retracted then the fuzzy fact (temp low) is retracted but the fact (temp medium) would still remain. A function such as

(combine-evidence temp)

could then be executed to create a single fact from all of the facts with the template temp. It would replace all of these facts with a single one and would need to deal with the dependencies of these facts in some way. Or, rather than having such a function and actually doing any combination of the facts into a single fact it could be that the defuzzification function(s) do this combining of evidence internally and produce a result that reflects this. We could in this case provide two types of functions for defuzzification: one that operates on individual fuzzy facts (no combining of evidence) and one that operates on fuzzy templates (combining evidence of all facts for the template before doing the defuzzification). There may be other suggestions and possibilities.

A third area that needs investigation is the idea of making FUZZY-VALUES a standard type in CLIPS (like integers, symbols, etc.). The current implementation has proceeded towards this goal and this may be a next logical step. This would allow FUZZY-VALUES to be stored in objects as well as in facts. Considerable effort is still needed to go this extra step and assistance from NASA or others would be appreciated.

Bug reports and any suggestions for modifications and further extensions are welcome. Many, perhaps most of the enhancements from 6.02 to 6.10d were a result of feedback from users. Let’s keep the communication going.

10 Acknowledgments

The author would like to acknowledge the contributions of the following people: Zenon Sosnowski, a visiting researcher from the Technical University of Bialystok in Poland, initiated the concept of FuzzyCLIPS and created the first version for CLIPS 4.3; Christina Lam, a student from the University of Toronto, reworked some of the code and the user guide for a later version of FuzzyCLIPS based on CLIPS 4.3; Jadwiga Sienkowicz, a student from Concordia University, created some example programs using a older version of FuzzyCLIPS as well as a version that included the graphics extensions available in wxCLIPS from the University of Edinburgh; Gary Riley of NASA provided copies of CLIPS 6.0 and the CLIPS test routines to assist in the development of FuzzyCLIPS; Reg Shevel of the Institute for Information Technology at NRC provided a great deal of assistance in the validation of the system.

11 References

1 Artificial Intelligence Section, CLIPS Reference Guide Volume I Basic Programming Guide, CLIPS Version 6.0, Lyndon B. Johnson Space Center, June 2 1993.

2 Artificial Intelligence Section, CLIPS Reference Guide Volume II Advanced Programming Guide, CLIPS Version 6.0, Lyndon B. Johnson Space Center, June 2 1993.

3 E.H. Shortliffe. Computer-based medical consultation: MYCIN. American Elsevier. New York 1976.

4 B.G. Buchanan and E.H. Shortliffe. Rule-Based Expert Systems. Addison-Wesley. Reading, MA 1984.

5 G. Shafer. A Mathematical Theory of Evidence. Princeton University Press. Princeton, NJ 1976.

6 K.P. Adlassing and G. Kolarz. Representation and semiautomatic acquisition of medical knowledge in Cadiag-1 and Cadiag-2. Computers and Biomedical Research, 19:63-79; 1988

7 T. Whalen, B. Schott, and F. Ganoe. Fault diagnosis in fuzzy network, Proceeding of the 1982 International Conference on Cybernetics and Society, IEEE Press, New York. 1982.

8 J. Buckley and W. Siler. Fuzzy Operators for possibility interval sets. Fuzzy Sets and Systems, 22:215-227; 1987.

9 J.F. Baldwin. Evidential support logic programming. Fuzzy Sets and Systems, 24:1-26; 1987.

10 K.S. Leung, W.S.F. Wong, and W. Lam. Application of a novel fuzzy expert system shell. Expert Systems, 6(1):2-10; 1989.

11 Z.A. Sosnowski. FLISP - a language for processing fuzzy data. Fuzzy Sets and Systems, 37:23-32; 1990.

12 Earl Cox. The Fuzzy Systems Handbook. AP Professional. 1995.

13 M. Cayrol, H. Farency and H. Prade. Fuzzy pattern matching, Kybernetes, 11:103-106; 1982.

14 L.A. Zadeh. Fuzzy sets. Information and Control, 8:338-383; 1965.

15 M. Mizumoto, S. Fukami, and K. Tanaka. Some Methods of Fuzzy Reasoning. In Advances in Fuzzy Set Theory and Applications, M.M. Gupta, R.K. Ragade, and R.R. Yager, eds. North-Holland, Amsterdam. 1979. pp.117-136.

16 L.A. Zadeh. The Concept of a Linguistic Variable and its Application to Approximate Reasoning. New York. 1973.

17 Tzi-cker Chiueh. Optimization of fuzzy logic inference architecture. Computer, May:67-71; 1992.

18 T. Whalen and B. Schott. Issues in fuzzy production systems. International Journal of Man-Machine Studies, 19:57; 1983.

19 A. Kaufmann and M.M. Gupta. Fuzzy Mathematical Models in Engineering and Management Science. North-Holland. 1988.

20 Z.A. Sosnowski. A Linguistic Variable in FLISP Programming Language. The Second Joint IFSA-EC and EURO-WG Workshop “Progress in Fuzzy Sets in Europe”, Vienna, Austria, April 6-8, 1988, pp.71-74.

21 A. Kaufman and M.M. Gupta. Introduction to Fuzzy Arithmetic. Theory and Applications. Van Nostrand Reinhold. 1985.

22 K.S. Leung and W.Lam. Fuzzy Concepts in Expert Systems, IEEE, September 1988, pp.43-56.

23 Giarratano and Riley. Expert Systems: Principles and Programming, PWS-KENT Publishing Company, 1989, p.270.

Appendix A: Shower Example

The purpose of the shower example is to simulate the flow and temperature of water leaving a shower head as a function of time and to build a fuzzy controller to keep the flow and temperature within some required ranges.

A.1 Shower Model

1. The volumes of the pipes are zero.
2. Water mixes perfectly at point X (see Figure 22: Shower).
3. The water pipe is a perfect thermal insulator.
4. There is no heat transfer in the water as it travels from point X to the shower head.

As a consequence of the above model, the flow out of the shower head (Fx) at any time t is exactly equal to the flow of cold water (Fc) at time t plus the flow of hot water (Fh) at time t. Any change to a valve position (vc, vh) or to the water pressure (Ph, Pc) is immediately reflected in the flow from the shower head. The temperature of the water leaving the shower head (Tx) at time t is equal to the temperature of water at point X. See equations below.

A.2 Shower Model Equations

$Tc \in [0, 35]$ °C
$Th > Tc$ $Th \in [0, 100]$ °C
$Tx = (FcTc + FhTh)/Fx$
$Fx = Fc + Fh$
$Px = 30$ kPa (atmospheric pressure)
$Fc = vc(Pc - Px)$
$Fh = vh(Ph - Px)$
$vc \in [0,1]$ - cold water valve position
$vh \in [0,1]$ - hot water valve position
$Pc = f$(neighbor watering lawn,clothes washing machine, etc.)
$Ph = f$(dish washing machine water consumption,etc.)

Changes of Pc and Ph are simulated.

A.3 Shower Control Objectives

1. The temperature of the water leaving the shower head must “never” exceed 45°C. This is generally impossible.
2. The water temperature should rarely be less than 15°C.
3. The water temperature should have a mean value of 36°C and have a small variance.
4. The water flow should have a mean value of 12 L/min and have a small variance.
5. The hot and cold water valve actuators should be moved infrequently. (The control computer wants to spend more time washing than adjusting the taps).

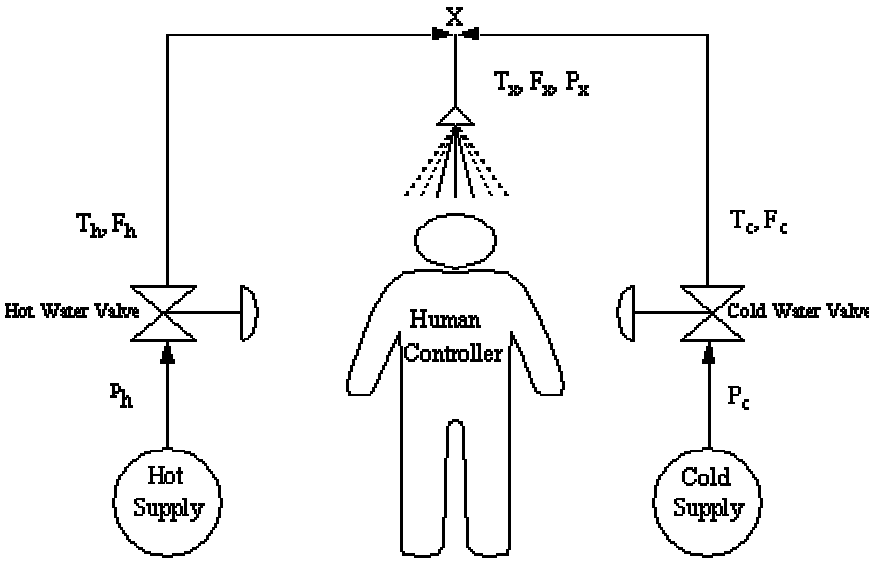


Figure 22: Shower

A.4 Fuzzy Control Loop

Below we show a schematic of a control loop with a fuzzy logic controller.

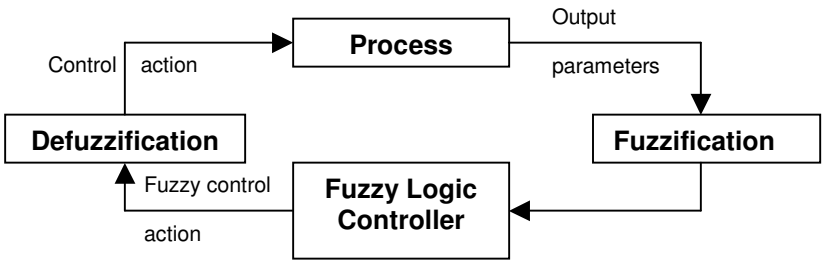


Figure 23: Fuzzy Control Loop

A.5 Text Based Version (No Graphical Interface - shwrNOUI.clp)

A.5.1 Steps to follow to run fuzzy shower example (UNIX version)

1. Start a version of FuzzyCLIPS.
2. Load the shower example (load "shwrNOUI.clp" from the fuzzy examples directory)
3. Run an example [(reset) and (run)] The program will ask for the values of parameters for temperature, pressure and valve positions. It stops when it reaches a flow between 11 and 13 L/min and water temperature between 34 and 38°C. The values of certain parameters will be printed after each set of fuzzy rules and defuzzification has taken place.
4. You will be prompted to enter further values or to quit.

A.5.2 Acknowledgment

The shower example was suggested by Robert Spring of the Noranda Technology Centre.

Index