FuzzyCLIPS Version 6.02

User's Guide

Knowledge Systems Laboratory
Institute for Information Technology
National Research Council Canada

May 1994

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.

1. Compatible with CLIPS version 6.02

2. CLIPS was developed by the Artificial Intelligence Section, Lyndon B. Johnson Space Center, NASA and is available from COSMIC, The University of Georgia, 382 Broad Street, Athens, GA 30602, USA.

Table of Contents

Table of Contents

1.0 Introduction

This report describes changes made at the National Research Council 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. It 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 2.0 describes restrictions and conditions of use for FuzzyCLIPS. Section 3.0 gives information on how to install FuzzyCLIPS at your site. Section 4.0 describes the changes made to CLIPS to implement fuzzy expert systems, and gives some theoretical background on fuzzy logic. Section 5.0 describes how to use the changes made at the NRC. Section 6.0 discusses the further changes made to CLIPS to accommodate the needs of continuously operating systems. Section 7.0 describes changes between the current version of FuzzyCLIPS and the previous version of FuzzyCLIPS built around CLIPS version 4.3. Section 8.0 briefly discusses some of the limitations of FuzzyCLIPS and what future work will be done.

2.0 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:

Knowledge Systems Laboratory
Institute for Information Technology
National Research Council Canada
Ottawa, Ontario Canada K1A 0R6
Electronic mail: fzclips@ai.iit.nrc.ca

2.1 Title and Conditions

NRC provides a fully paid up and non exclusive 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 sublicence 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 email 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.

3.0 Installation Information

FuzzyCLIPS has been successfully compiled and tested on the following machines:

Sun Sparc workstations (using cc compiler)
IBM PC DOS machines (using Borland C++ 4.0)
Macintosh II (using the MPW C 3.3 compiler)

FuzzyCLIPS is available via anonymous ftp from ai.iit.nrc.ca and via the World Wide Web (WWW) using URL designation http://ai.iit.nrc.ca/fuzzy/fuzzy.html. After connecting to the system with ftp do the following:

1. cd pub/fzclips	
2. binary	(binary transfers are required)
3. get fzclips.tar.Z	(unix version - compiled for sun workstation - tar'd and compressed)
or	
get fzclips.sea.hqx	(Macintosh version - self extracting archive format created with Stuffit Deluxe then converted to binhex format)
or	
get fzclips.exe	(PC version - self extracting archive format created with PAK)
or	
get fzdocs.ps	(FuzzyCLIPS documentation - postscript file)

The documentation is only included with the unix version (fzclips.tar.Z - both postscript format and framemaker format). If the framemaker files are required by others then they can be made available on request.

If using the WWW to access FuzzyCLIPS you will be prompted to provide your name and address of your organization and then will be able to access one or all of the documents.

On a unix system the following commands are used to install FuzzyCLIPS.

uncompress fzclips.tar.Z (only required if accessed from ftp; WWW will automatically uncompress the file)
tar -xvf fzclips.tar

On a Macintosh system use a Binhex program to extract the fzclips.sea file then double-click on the fzclips.sea file and it will automatically uncompress and install the system. Note that if you have a version of the Stuffit file compression program you could extract only the parts that you need instead of the entire system.

On a PC system the program fzclips.exe when executed will uncompress and install the FuzzyCLIPS system. Execute the program with the following commands:

mkdir /fzclp602	- create a directory for the files to be unpacked into; the fzclips.exe file should be in this directory
cd /fzclp602	
fzclips /path	- the /path is necessary to recreate the correct directory structure; if omitted then all files will be placed in the current directory without any sub-directories.

If you have a version of the PAK archiving/compression program then you could extract only the parts that you need instead of the entire system.

The following is a list of files included for all systems. Those marked with an asterisk (*) are CLIPS files that have been modified to support FuzzyCLIPS. Those marked with an exclamation mark (!) are new files that have been added to CLIPS to support Fuzzy-CLIPS. Those files marked with a number sign (#) reflect files replaced to advance from CLIPS 6.0 to CLIPS 6.01. Those marked with a tilde (~) identify files that have been replaced or modified to advance to CLIPS 6.02. Those marked with a dollar sign ($) are modified to fix some minor bug not fixed in 6.01 or 6.02.

(*#)agenda.c	($*#)analysis.c	(*)argacces.c	bload.c
bmathfun.c	(*)bsave.c	(!)cfdef.c	classcom.c
classexm.c	classfun.c	(~)classinf.c	classini.c
classpsr.c	(~)clsltpsr.c	(*~)commline.c	(*)conscomp.c
(#)constrct.c	(#)constrnt.c	crstrtgy.c	cstrcbin.c
(#)cstrccom.c	(#)cstrcpsr.c	(#)cstrnbin.c	(#)cstrnchk.c
(#)cstrncmp.c	(#)cstrnops.c	cstrnpsr.c	(#)cstrnutl.c
default.c	defins.c	(*)developr.c	dffctbin.c
dffctbsc.c	dffctcmp.c	dffctdef.c	dffctpsr.c
dffnxbin.c	dffnxcmp.c	dffnxexe.c	dffnxfun.c
dffnxpsr.c	dfinsbin.c	dfinscmp.c	drive.c
edbasic.c	edmain.c	edmisc.c	edstruct.c
edterm.c	emathfun.c	(*~)engine.c	(*~)evaluatn.c
(~)expressn.c	(*)exprnbin.c	(#)exprnops.c	exprnpsr.c
extnfunc.c	factbin.c	(*)factbld.c	factcmp.c
(*)factcom.c	(*#)factgen.c	(*)facthsh.c	(#)factlhs.c
(*#)factmch.c	(*#)factmngr.c	(*)factprt.c	(*)factrete.c
(*#)factrhs.c	filecom.c	filertr.c	(!)fuzzycom.c
(!)fuzzydef.c	(!)fuzzylhs.c	(!)fuzzypsr.c	(!)fuzzyrhs.c
(!)fuzzyutl.c	generate.c	genrcbin.c	genrccmp.c
genrccom.c	genrcexe.c	genrcfun.c	genrcpsr.c
globlbin.c	globlbsc.c	globlcmp.c	globlcom.c
(#)globldef.c	globlpsr.c	immthpsr.c	incrrset.c
inherpsr.c	(#)inscom.c	insfile.c	(~)insfun.c

($)insmngr.c	insmoddp.c	insmult.c	inspsr.c
(~)insquery.c	(~)insqypsr.c	iofun.c	lgcldpnd.c
(*)macmain.c	(*)main.c	($~)memory.c	miscfun.c
modulbin.c	modulbsc.c	modulcmp.c	moduldef.c
modulpsr.c	modulutl.c	msgcom.c	msgfun.c
(#)msgpass.c	msgpsr.c	(~)multifld.c	(~)multifun.c
objbin.c	objcmp.c	objrtbin.c	(~)objrtbld.c
objrtcmp.c	objrtfnx.c	objrtgen.c	(~)objrtmch.c
(#)pattern.c	pprint.c	prccode.c	(#)prcdrfun.c
prcdrpsr.c	prdctfun.c	(*~)prntutil.c	(#)reorder.c
(#)reteutil.c	retract.c	router.c	(*)rulebin.c
rulebld.c	rulebsc.c	(*~)rulecmp.c	rulecom.c
rulecstr.c	ruledef.c	(*)ruledlt.c	(*)rulelhs.c
(*#)rulepsr.c	(*)scanner.c	strngfun.c	strngrtr.c
(*)symblbin.c	(*)symblcmp.c	(*)symbol.c	(*)sysdep.c
textpro.c	(*)tmpltbin.c	tmpltbsc.c	(*)tmpltcmp.c
(*)tmpltcom.c	(*)tmpltdef.c	tmpltfun.c	(*)tmpltlhs.c
(*~)tmpltpsr.c	tmpltrhs.c	utility.c	watch.c

(*)agenda.h	(*)analysis.h	argacces.h	bload.h
bmathfun.h	bsave.h	(!)cfdef.h	classcom.h
classexm.h	classfun.h	classinf.h	classini.h
classpsr.h	clips.h	(~)clipsmem.h	clsltpsr.h
commline.h	conscomp.h	(*)constant.h	constrct.h
(#)constrnt.h	crstrtgy.h	cstrcbin.h	cstrccmp.h
cstrccom.h	cstrcpsr.h	cstrnbin.h	cstrnchk.h
cstrncmp.h	cstrnops.h	cstrnpsr.h	cstrnutl.h
default.h	defins.h	developr.h	dffctbin.h
dffctbsc.h	dffctcmp.h	dffctdef.h	dffctpsr.h
dffnxbin.h	dffnxcmp.h	dffnxexe.h	dffnxfun.h
dffnxpsr.h	dfinsbin.h	dfinscmp.h	drive.h
ed.h	(*)engine.h	evaluatn.h	(~)expressn.h
exprnbin.h	(#)exprnops.h	exprnpsr.h	(*$)extnfunc.h
extobj.h	factbin.h	factbld.h	factcmp.h
factcom.h	factgen.h	facthsh.h	factlhs.h
factmch.h	(*)factmngr.h	(*) factprt.h	(*)factrete.h
factrhs.h	filecom.h	filertr.h	(!)fuzzycom.h
(!)fuzzydef.h	(!)fuzzylhs.h	(!)fuzzylv.h	(!)fuzzypsr.h
(!)fuzzyrhs.h	(!)fuzzyutl.h	(!)fuzzyval.h	generate.h
genrcbin.h	genrccmp.h	genrccom.h	genrcexe.h
genrcfun.h	genrcpsr.h	globlbin.h	globlbsc.h
globlcmp.h	globlcom.h	globldef.h	globlpsr.h
immthpsr.h	incrrset.h	inherpsr.h	inscom.h

insfile.h	insfun.h	insmngr.h	insmoddp.h
insmult.h	inspsr.h	insquery.h	insqypsr.h
lgcldpnd.h	match.h	miscfun.h	modulbin.h
modulbsc.h	modulcmp.h	moduldef.h	modulpsr.h
modulutl.h	msgcom.h	msgfun.h	msgpass.h
msgpsr.h	(~)multifld.h	multifun.h	network.h
objbin.h	objcmp.h	object.h	objrtbin.h
objrtbld.h	objrtcmp.h	objrtfnx.h	objrtgen.h
(~)objrtmch.h	pattern.h	pprint.h	prccode.h
(*)prcdrfun.h	(*)prcdrpsr.h	prntutil.h	reorder.h
(#)reteutil.h	retract.h	router.h	(*)rulebin.h
rulebld.h	rulebsc.h	(*)rulecmp.h	rulecom.h
rulecstr.h	(*)ruledef.h	ruledlt.h	(*)rulelhs.h
(*)rulepsr.h	(*)scanner.h	(*~)setup.h	(*)shrtlnkn.h
strngfun.h	strngrtr.h	(*)symblbin.h	(*)symblcmp.h
(*)symbol.h	sysdep.h	(*)tmpltbin.h	tmpltbsc.h
tmpltcmp.h	tmpltcom.h	(*)tmpltdef.h	tmpltfun.h
tmpltlhs.h	tmpltpsr.h	tmpltrhs.h	utility.h
watch.h	cmptblty.h		

(*)clips.hlp - this file may appear in a different directory from those above

All of the executable files distributed with FuzzyCLIPS have been created to include fuzzy facts and reasoning, certainty factors, and runtime extensions (see section 6.0). 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

The next 3 sections describe the file structures and contents of directories for each of the unix (Sun), Macintosh, and PC versions of FuzzyCLIPS.

3.1 Unix Version (Sun)

The main directory FZ__CLIPS.6.01 has all of the files (*.c and *.h) listed above plus the following files and directories.

makefile.x (File used to create FZ_XCLIPS_shower, FZ_XCLIPS, FZ_CLIPS or FZ_CLIPS_shower. Note that for X versions the flag WINDOW_INTERFACE in setup.h must be set to 1 and for non X versions it must be 0. Also note that the flag ShowerProblemUI in main.c or xmain.c must be set to 1 for versions with the graphical interface for the shower problem included and to 0 for those without it.)

FZ_XCLIPS_shower	(executable FuzzyCLIPS system with shower problem included) (includes the code for X development interface)
FZ_CLIPS_shower	(executable FuzzyCLIPS system with shower problem included) (generic - NO X development interface)

(files that support the X windows development interface - % indicates some minor changes made at NRC to better support the Restart option when quitting clips [xmanin.c, xmenu_file.c] and to allow Control-C to halt execution of a CLIPS program [xclipstext.c])

xclips.c	(%)xmain.c	(%)xmenu_file.c	xmenu_wind.c
(%)xclipstext.c	xmenu.c	xmenu_opt.c	
xedit.c	xmenu_exec.c	xmenu_watch.c	
xclips.h	xmain.h	xmenu_wind.h	
xedit.h	xmenu.h	xsetup.h	

DOCS: (directory - framemaker documentation files + postscript output of docs)

FuzzyCLIPS.doc	FuzzyCLIPSTitle.doc
FuzzyCLIPSTOC.doc	FuzzyCLIPS.book
FuzzyCLIPS.doc.ps	snapshot.shower

ShowerProblem: (directory - files to support shower problem - see Appendix B)

EventNames.h	Events.h
Shower.G	Shower.c
Shower.po	ShowerCallbacks.c
ShowerGraph.par	ShowerSimulate.c
ShowerSimulate.h	Shower_audio.c
Simulate.c	Simulate.h
audio_device.h	audio_errno.h
audio_hdr.h	flush.au
formatData	formatData.c
libaudio.a	libaudio.h
showerUI25.clp	showerUI8.clp

src: (Directory)

Shower.ps	Shower_I.ps
Shower_c.cps	Shower_m.c
Shower_p.h	

tmp: (Directory)

Shower.PS	Shower.h
Shower_C.c	Shower_C.cps
Shower_C.h	Shower_c.c
Shower_c.h	Shower_s.c

fzexamples: (Directory - sample of examples in FuzzyCLIPS)

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)
lin1st.par (used with xvgr program to plot results of lin1st.clp or lin1stdl.clp)

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

FZ_XCLIPS_shower executable file has been created to include the code required to support the Shower Problem simulation (graphics user interface version). To run the simulation you will need to be using Openwindows version 3.0 or later (Note it will NOT run under Solaris 2 since it uses the NeWS system which has been removed in that version of the operating system). The ShowerProblem directory contains all of the files needed to modify the simulation if required and the details are presented in Appendix B. A second version of the shower problem (shwrNOUI.clp) that is text-based is provided in the directory ‘fzexamples’. It requires no special features and is also described in Appendix B.

FuzzyCLIPS has been compiled (using cc, a non-ANSI compiler) and tested on a Sun Sparc system running SunOS version 4.1.x.

3.2 IBM PC version

The main directory FZCLP602\SOURCES has all of the files (*.c and *.h) listed above. The following files and directories are also contained in the directory FZCLP602 (assuming that this is the name of the directory in which the distributed system was placed).

clips.hlp (file used by the CLIPS help command)
clips6.hlp (file used by the CLIPS help command)
clipscmd.exe (FuzzyCLIPS command line version - runs under windows)
clipswin.exe (FuzzyCLIPS windows version - development GUI included)

(Note that there are restrictions on the size of the runtime stack due to a 64K limit placed on the stack/ local heap/static data segment. Currently there is a stack size of about 10000 bytes for the clipscmd.exe version and about 7600 for the clipswin.exe version. See clipswin.def or clipscmd.def files for exact values. This may cause some programs to report stack errors or other fatal errors. There are ways to overcome these limitations but at this time we have not explored them vigorously - all suggestions are welcome. Also note that to recreate (i.e. recompile/relink) either of these versions it may be necessary to modify the setup.h file to reflect the features to be included. In particular for clipswin versions the flag WINDOW_INTERFACE in setup.h must be set to 1 and for clipscmd versions it should be 0. Also note that the flag ShowerProblemUI in main.c (SOURCES directory) or main.c (PCINTFAC directory) must be set to 0 or there will be unresolved externals. This flag is only applicable on Sun workstations. Both versions have been created with stack checking ON. This may cause programs to run slower but the check was felt to be useful and perhaps even necessary. Finally note that the project files were set to create and use precompiled header files. This will cause quite large .CSM files to be created but does speed compilations.)

FZEXMPLE (Directory with FuzzyCLIPS 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)

PCINTFAC (Directory containing files used to create the GUI interface for developers)

1.cur	2.cur	3.cur	4.cur
5.cur	6.cur	7.cur	8.cur
9.cur	wait.cur	clipsdde.h	dialog.h
menucmds.c	dialog1.c	menucmds.h	clipswin.def
clipswin.ico	dialog2.c	query.cur	clipswin.rc
status.c	clipswin.res	ids.h	status.h
clipswin.rws	joe.bmp	main.c	system.c
main.h	clipscmd.def	diag_win.c	clipsdde.c
diag_win.h	menu.c		

PCPROJ (Directory with project files for Borland C++ 4.0)

clips6.hpj	clipscmd.ide	clipswin.dsw	ide_clps.prj
clips6.rtf	clipscmd.prj	clipswin.ide	clipscmd.csm
clipswin.prj	clipscmd.dsw	clipswin.csm	clipswin.mak
clipscmd.mak			

FuzzyCLIPS has been compiled (using Borland C++ 4.0, an ANSI compiler) and tested on a PC compatible system running DOS 6.2 and Windows 3.1. Note that the PC version caused more problems during development and testing (by far) than either the unix (sun) version or the Macintosh version.

3.3 Macintosh version

The main directory 'FZ__CLIPS 6.02:FZ_CLIPS source' has all of the files (*.c and *.h) listed above. The following files and directories are also contained in the directory 'FZ__CLIPS 6.02'.

MPW C Project Files (Directory - contains the makefiles for creating FuzzyCLIPS)

> FZ_CLIPS.6.02.make   (for making FuzzyCLIPS with Interfaces version)
> FZ_CLIPS.6.02G.make(for making FuzzyCLIPS command line version)
> FZ_CLIPS.6.02 (executable version of FuzzyCLIPS with Interfaces)
> FZ_CLIPS.6.02.r (FZ_CLIPS resource file)
> clips.hlp (help file used by the CLIPS help command)

(Note that to recreate (i.e. recompile/relink) either of these versions it may be necessary to modify the setup.h file to reflect the features to be included. In particular for the interface version the flag, WINDOW_INTERFACE, in setup.h must be set to 1 and for the non-interface version it should be 0. Also note that the flag, ShowerProblemUI, in main.c and macmain.c must be set to 0 or there will be unresolved externals. This flag is only applicable on Sun workstations.)

fzexamples (Directory - contains the example programs and associated files)

> 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)

Mac Interface Source (Directory - contains the Interface source files)

ALPHA.C	EDITW.C	MACINIT.C	RULEDLOG.C
CLIPSMNU.C	EXECDLOG.C	MACMAIN.C	SCRAP.C
CLIPSSTW.C	FACTDLOG.C	MAINLOOP.C	SEARCH.C
COOLDLOG.C	FILEMENU.C	MENUCMDS.C	STATUSW.C
DISPLAYW.C	FONTPICK.C	MENUHNDL.C	STNRDMAC.C

DLOGMNGR.C	GNRCDLOG.C	PRCDDLOG.C	UNDO.C
EDITDLOG.C	MACFUNCS.C	PRINT.C	WINDOW.C
EDITMENU.C			

alpha.h	editmenu.h	interface.h	print.h
clipsmnu.h	editw.h	interface_scrap.h	ruledlog.h
clipsstw.h	execdlog.h	macinit.h	search.h
common.h	factdlog.h	mainloop.h	statusw.h
cooldlog.h	filemenu.h	menucmds.h	stnrdmac.h
displayw.h	fontpick.h	menuhndl.h	undo.h
dlogmngr.h	gnrcdlog.h	prcddlog.h	window.h
editdlog.h			

> [Note that 4 files in the interface sources (WINDOW.C, EXECDLOG.C, EDITDLOG.C and MENUHNDL.C) must be compiled with the -d dangerousPattern switch set]

FuzzyCLIPS has been compiled (using MPW 3.3, an ANSI compiler) and tested under Mac OS System 7.x. The distributed version is created with a minimum size set to 1.5K and a preferred size of 4.0K. These values can be changed although it is possible that some of your programs will not run with less memory.

4.0 Fuzzy Expert Systems

In the real world there exists much fuzzy knowledge, i.e., knowledge which 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 which 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], 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.

4.1 Fuzziness

Fuzziness 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* 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 [12]. 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 with each element x of U a number $\mu_A(x)$ in the interval [0,1] which 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

Regarding (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).

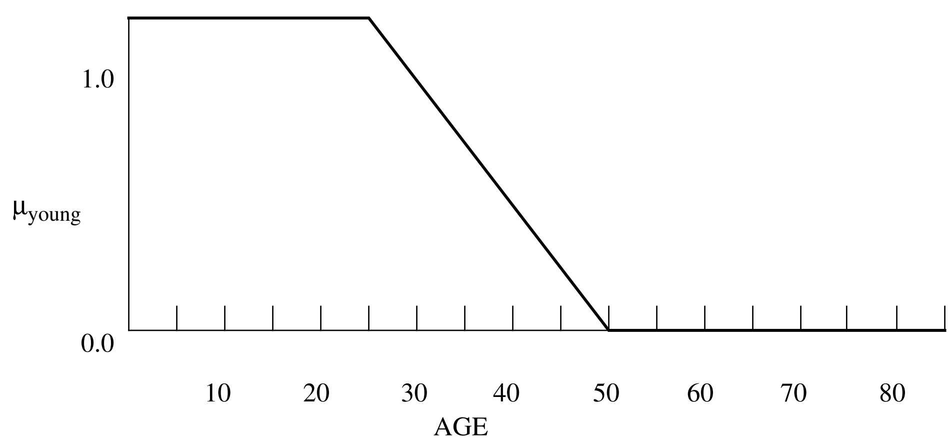


Figure 1: Possibility distribution of *young*

The possibility distribution of a fuzzy concept like *more or less young* or *very young* can be obtained by applying arithmetic operations to the fuzzy set of the basic fuzzy term *young*, where the modifiers *more or less* and *very* are associated with specific mathematical functions. For instance, the possibility values of each age in the fuzzy set representing the fuzzy concept *more or less young* might be calculated by taking the square root of the corresponding possibility values in the fuzzy set of *young* (see Figure 2).

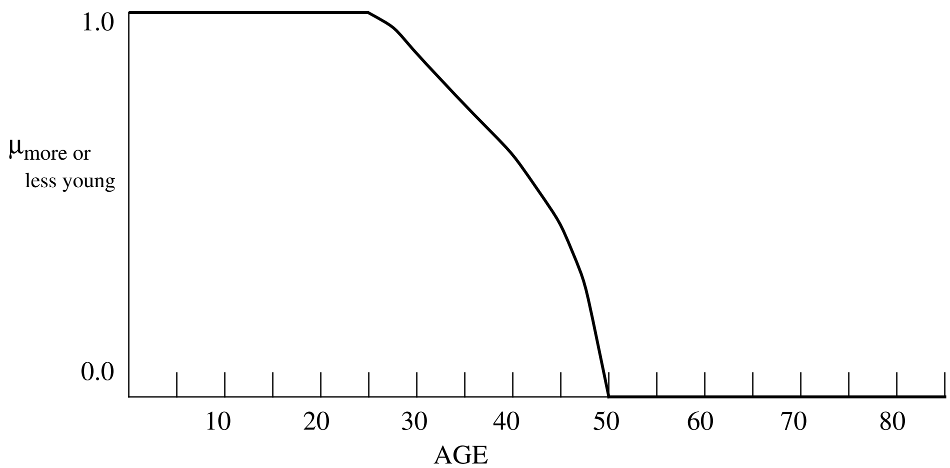


Figure 2: Possibility distribution of *more or less young*

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

```
(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) which 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 illustrates the primary term values of the fuzzy variable *water_temperature*.

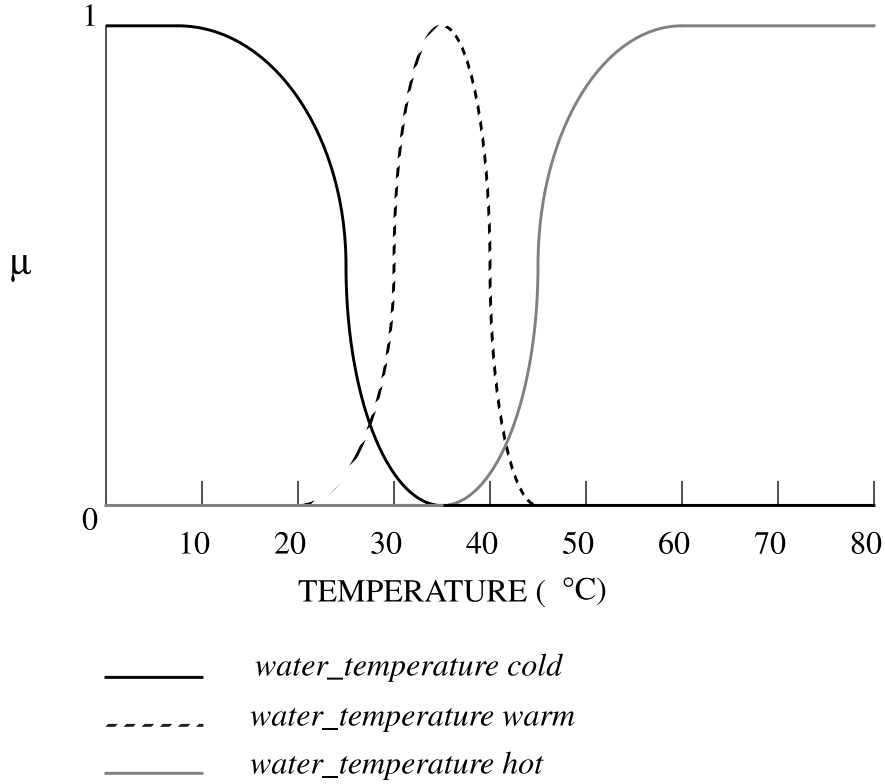


Figure 3: Primary terms of a linguistic variable

The syntax for defining fuzzy variables, fuzzy terms, and fuzzy term modifiers is discussed in detail in 5.1 on page 30.

4.2 Uncertainty

Uncertainty occurs when one is not absolutely certain about a piece of information. The degree of certainty is usually represented by a crisp numerical value on a scale from 0 to 1, where a certainty factor 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:

(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 which 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.

4.3 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.

4.3.1 Simple rules

Consider the simple rule of form

if A then C $\quad CF_r$
A' $\quad CF_f$

C' $\quad CF_c$

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.

4.3.1.1 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 may 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

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

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
```

$CF_c = 0.7 * 0.8$

4.3.1.2 FUZZY_CRISP Simple Rule

If the type of rule is FUZZY_CRISP, then A′ must be a fuzzy fact with the same fuzzy variable as specified in A in order 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, clearly (*pressure low*) and (*pressure medium*) overlap and thus match, while (*pressure low*) and (*pressure high*) do not match.

.

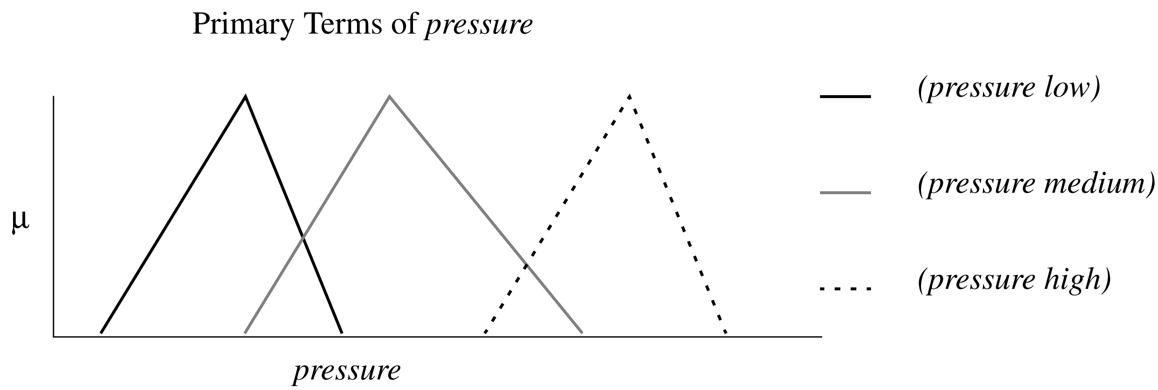


Figure 4: Matching of fuzzy facts

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 [17]

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

where[1]:

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

and

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

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

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

1. 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.

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 0.8
    (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:

FUZZY_CRISP EXAMPLE

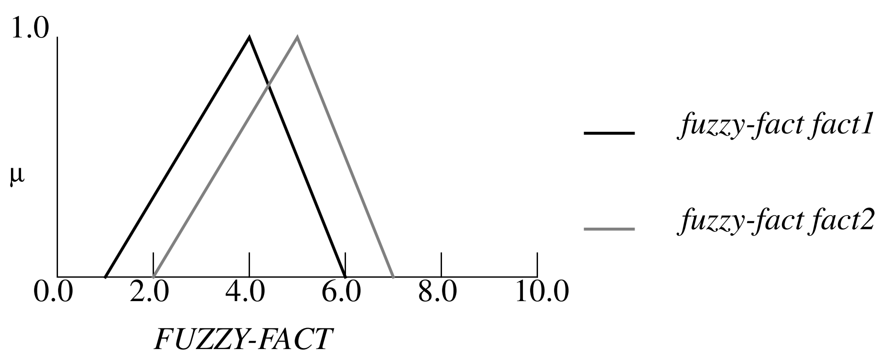


Figure 5: Fact and antecedent fuzzy sets

First, the necessity is calculated as in Figure 6.

NECESSITY CALCULATION

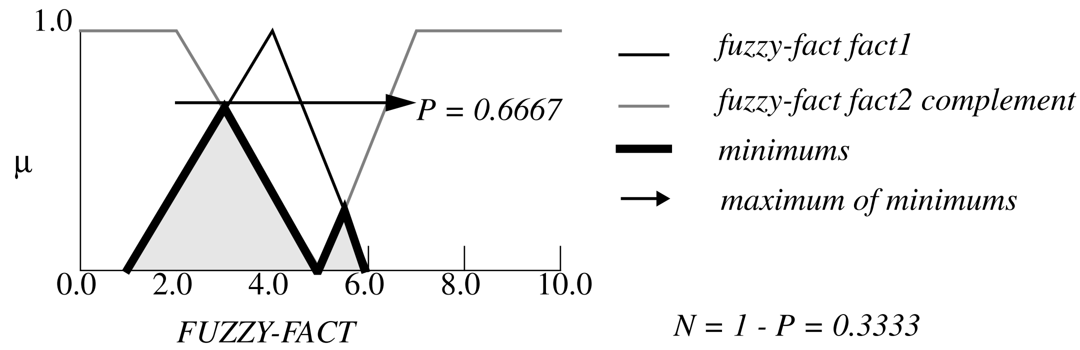


Figure 6: $N(F_\alpha|F'_\alpha)$

Since the necessity is less than 0.5, $S = (N(F_\alpha \mid F'_\alpha) + 0.5) * P(F_\alpha \mid F'_\alpha)$ (see Figure 7).

SIMILARITY CALCULATION

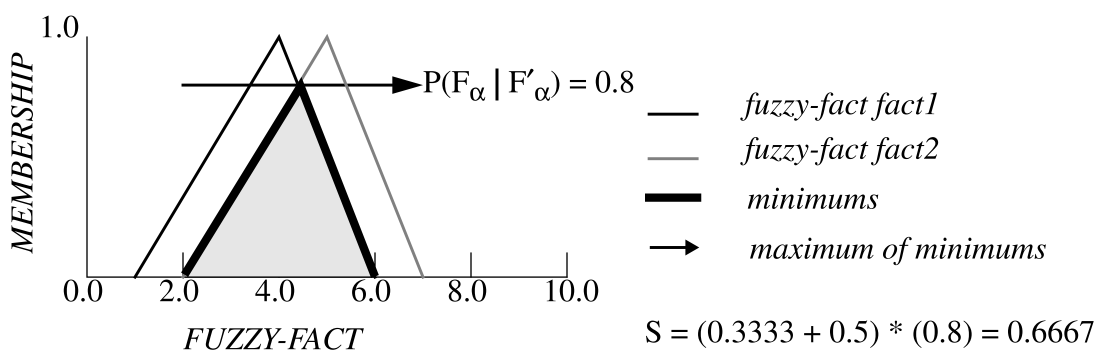


Figure 7: $S=(N(F_\alpha|F'_\alpha)+0.5)*P(F_\alpha|F'_\alpha)$

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

4.3.1.3 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 [13] 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 [18]. The calculation of the conclusion is based upon the compositional rule of inference [13], 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 [19]

$$\mu_{F'_c}(v) = max_{u \in U}(min(\mu_{F'_\alpha}(u), \mu_R(v)))$$

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
```

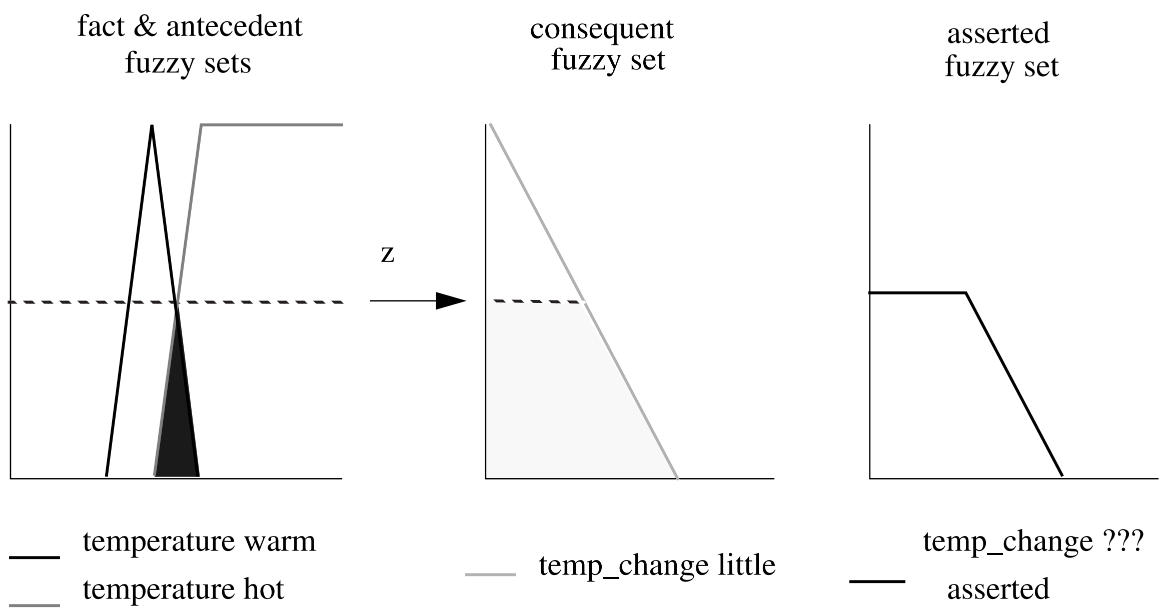


Figure 8: Compositional Rule of Inference

<u>4.3.2 Complex rules</u>

4.3.2.1 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

4.3.2.2 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 [20].

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) which 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[1]

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 9 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}

1. A membership function of a fuzzy set C which is the intersection of fuzzy sets A and B is defined by the following formula $\mu_c(x) = \min(\mu_A(x), \mu_B(x)),\quad \text{for } x \in U$

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}.

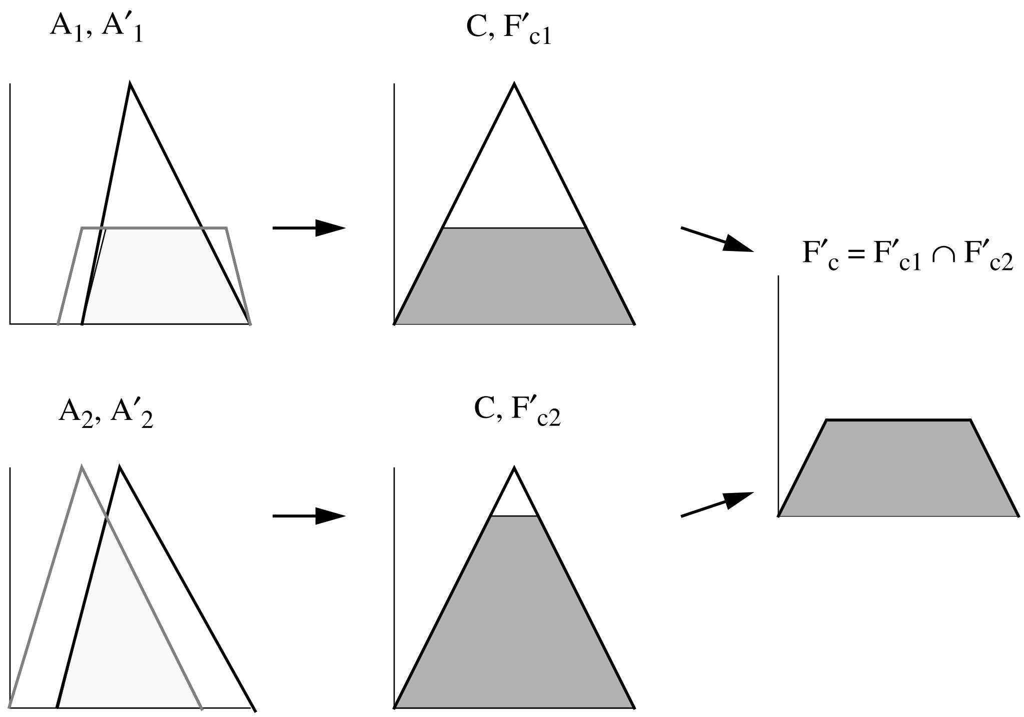


Figure 9: Compositional rule for multiple antecedents

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 can be applied repeatedly to handle any combination of antecedent patterns.

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

4.3.3 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. 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:

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[1] of two fuzzy sets

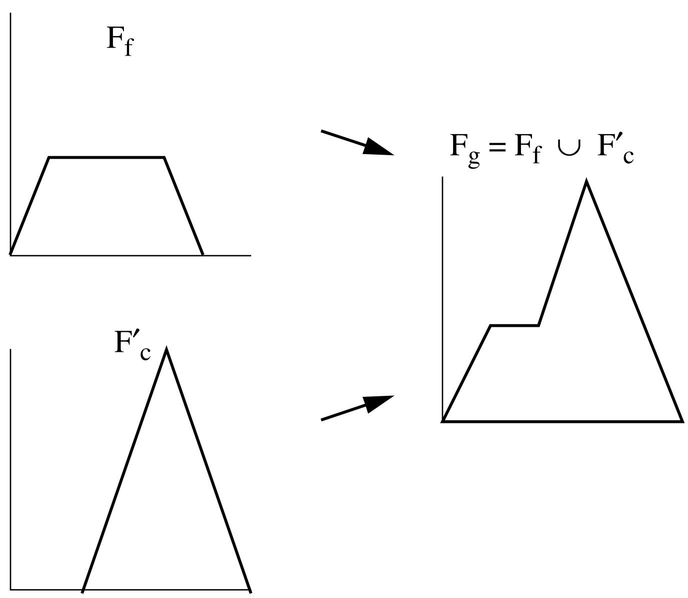


Figure 10: union of fuzzy sets - global contribution

1. 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 $x \in U$

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 which 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 B and also see the example clips programs fzCmplr.clp and fzCmpmod.clp which use salience and MODULES respectively to ensure all fuzzy rules that are related fire together.

4.3.4 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 [20]). 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 5.7.4). The calculated certainty factor for a rule is:

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

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.

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 conse-
quent 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.

4.3.5 Certainty factors in assert statements

When a fact is asserted in FuzzyCLIPS, its certainty factor may be specified (see 5.4.3 for details). 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.
```

4.4 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 which 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.

4.4.1 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 which 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 and 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

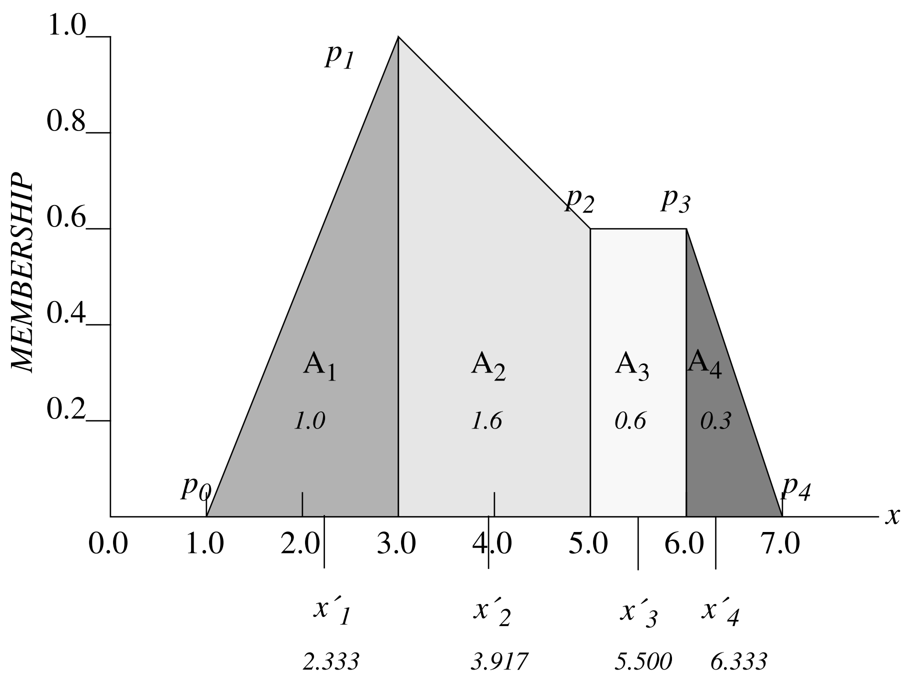


Figure 11: Example of COG defuzzification

`For each shaded subsection above, the area and centre of gravity is calculated according to the shape identified (i.e. triangle, rect-`

angle or trapezoid). The centre of gravity of the whole set is then determined:

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

4.4.2 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 11, 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.

Example 13

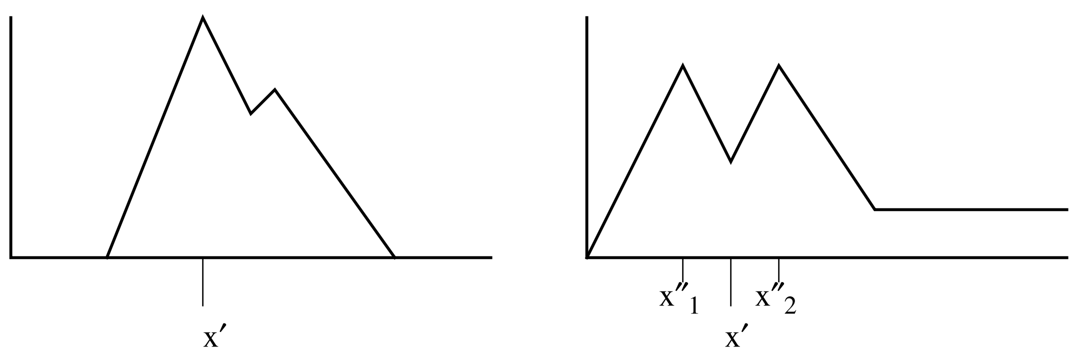


Figure 12: Example of MOM defuzzification

Note that an ambiguity occurs when a maxima is a plateau 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 3 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 [9.0] for a discussion of this problem and see [5.5] to see how to perform defuzzification in FuzzyCLIPS using the moment-defuzzify and maximum-defuzzify functions).

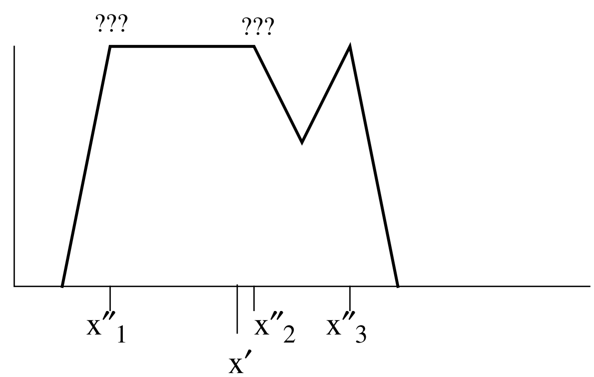


Figure 13: MOM example - Ambiguity

5.0 Steps to follow when using the fuzzy extension

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.

5.1 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
    (
      t_1
      .
      . ; list of primary terms
      .
      t_n
)
    [(
      m_1
      .
      . ; list of modifiers
      .
      m_k
)]
)

<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 begin and end of the interval which 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** which 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. The m_i are modifiers (such as very, more_or_less) used to modify the terms. Note that the [] around the list of modifiers identifies that they are optional.

Example 14

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

5.1.1 Primary terms

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

(<name> <description of fuzzy set>)

where a <name> represents the name of a primary term used to describe a fuzzy variable, and <description of fuzzy set> defines a membership function of the given primary term. The membership function can have a singleton or a standard function representation.

<description of fuzzy set> ::= <singletons> | <standard>

5.1.1.1 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

$$A = \sum_{i=1}^{n} \mu(x_i)/x_i = \mu_1/x_1 + \mu_2/x_2 + \ldots + \mu_n/x_n$$

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: $x_i < x_{i+1}$ 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 4.4.1, 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:

$$\mu(x) = \begin{cases} \mu(x_1), & x \le x_1 \\ \dfrac{\mu(x_{i+1}) - \mu(x_1)}{x_{i+1} - x_i}x, & x_i < x \le x_{i+1} \\ \mu(x_n), & x_n < x \end{cases}$$

Example 15

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 14.

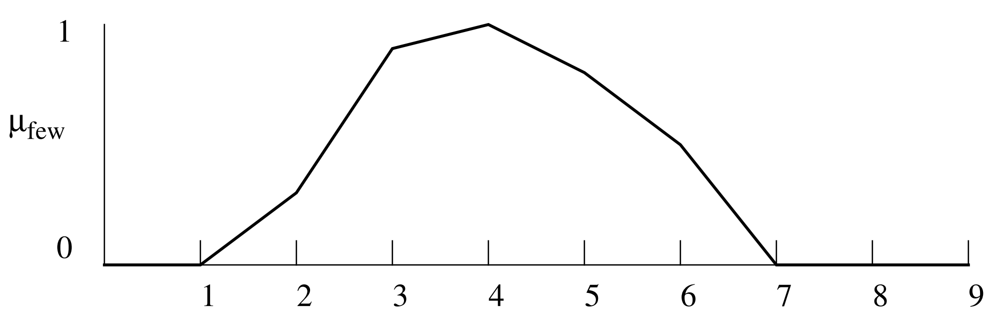


Figure 14: Fuzzy set of *group few*

In FuzzyCLIPS one can define a linguistic variable *group* which 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
    ()                  ; no modifiers declared (may be omitted)
)                       ; end of fuzzy deftemplate
```

5.1.1.2 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 15.

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

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

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

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

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

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

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

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

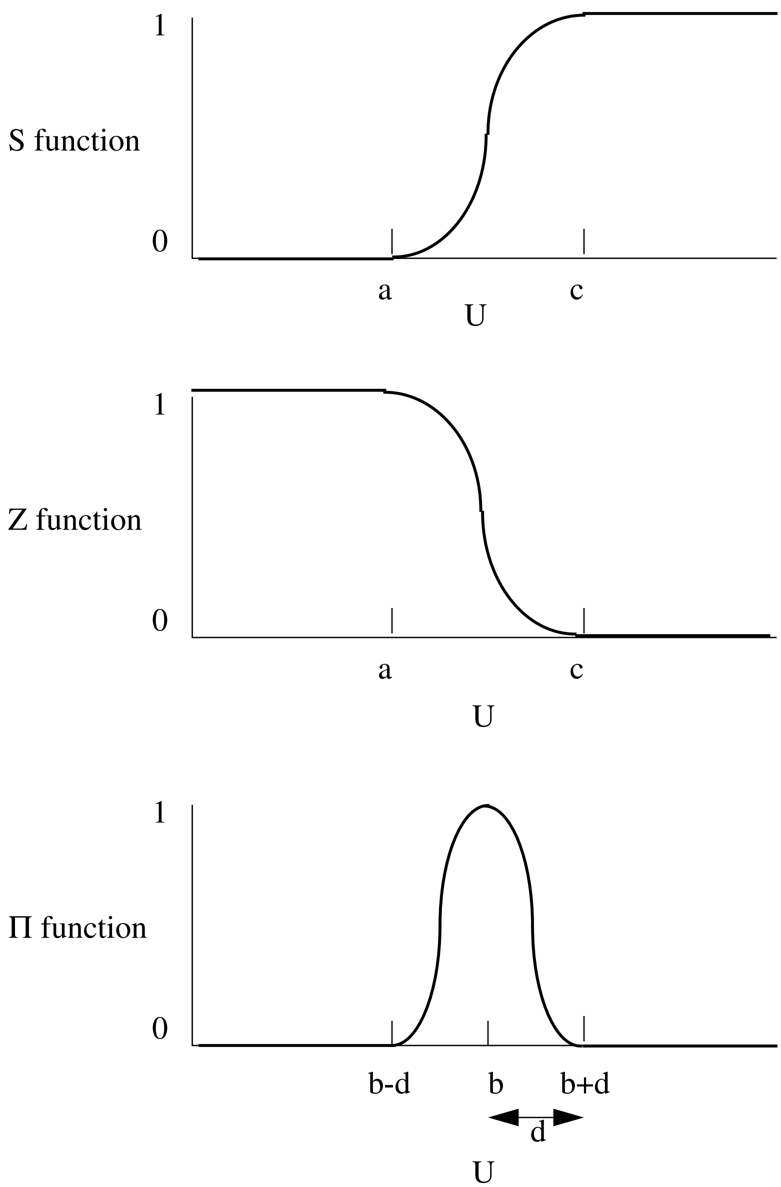


Figure 15: Standard fuzzy set functions

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 which represent the parameters of the respective functions.

Example 16

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

FuzzyCLIPS immediately converts all standard notation to singleton representation. Nine points, equally spaced along the x axis are selected to represent the functions (see Figure 16). The number of points (9) can be changed by modifiying the value of ArraySIZE which is defined in the file fuzzypsr.h and 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.

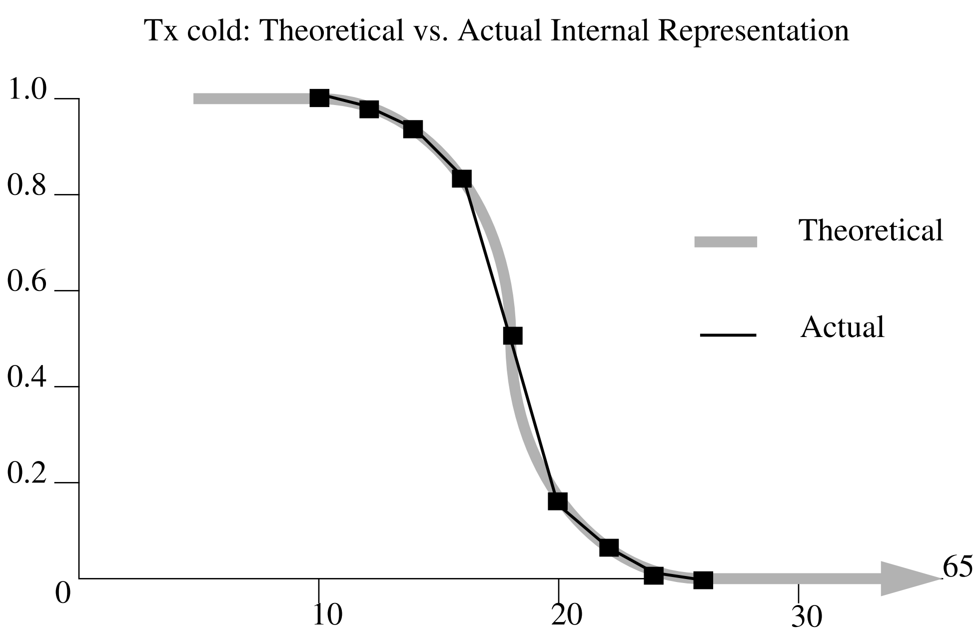


Figure 16: Approximation of standard functions

5.1.2 Modifiers

As mentioned in section 4.1, a *modifier* may be used to apply a mathematical function such as sqr() or sqrt() to the membership values of a fuzzy set. In order to be able to alter a fuzzy set with a modifier, the modifier must first be declared in the *deftemplate* statement of the corresponding fuzzy variable.

The declaration of a modifier m_j (j=1,...,k) has the form

(<name> <function name>)

where <name> represents the name of the modifier and <function name> represents the name of a pre-defined CLIPS function, a deffunction or a user-defined function that accepts a single numeric value and returns a numeric value. See [2] for information on how to add a user-defined function.

Example 17

```
(deftemplate speed      ;declaration of linguistic variable speed
     0 160 km/h         ;universe of discourse
     (                  ;start of primary term declarations
      (slow (30 1) (60 0));slow defined using singleton notation
      (fast (50 0) (80 1));fast defined using singleton notation
     )                  ;end of primary term declarations
     (                  ;start of modifier declarations
     (very sqr)         ;modifier very and function sqr
     (quite my_function) ;modifier quite, user-defined my_function
     )                  ;end of modifier declarations
)                       ; end of fuzzy deftemplate
```

5.2 Using Fuzzy Variables in LHS Patterns

A fuzzy LHS pattern is of the form:

(<fuzzy variable name> <value of fuzzy variable>)

The <*value of fuzzy variable*> is a fuzzy set specified by a combination of primary terms, modifiers, and the logical operators NOT and OR.

The simplest <*value of fuzzy variable*> is a single primary term.

Example 18

```
(deftemplate group      ;declaration of fuzzy variable group
     (
     (few (3 1) (6 0))   ;primary term few
     (many (4 0) (6 1))  ;primary term many
     )
     (
     (very sqr)          ;modifier very is sqr function
     )
)

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

The primary term may be preceded by one or more modifiers.

Example 19

```
(group very few)        ;a single modifier applied to few
(group very very few)   ;the modifier function is applied twice
```

Before a modifier function is applied to a fuzzy set, interpolation is performed to determine extra points so that the maximum height (rise) of any line segment is about 0.1. The extra points maintain the accuracy of the curves that result from taking the *sqr* or *sqrt* of the y values of a straight line (see Figure 17). Note that if we took the square of the membership values of the 2 points that define the set few we would just get the exact same 2 points again (since square of 0 and 1 are just 0 and 1).

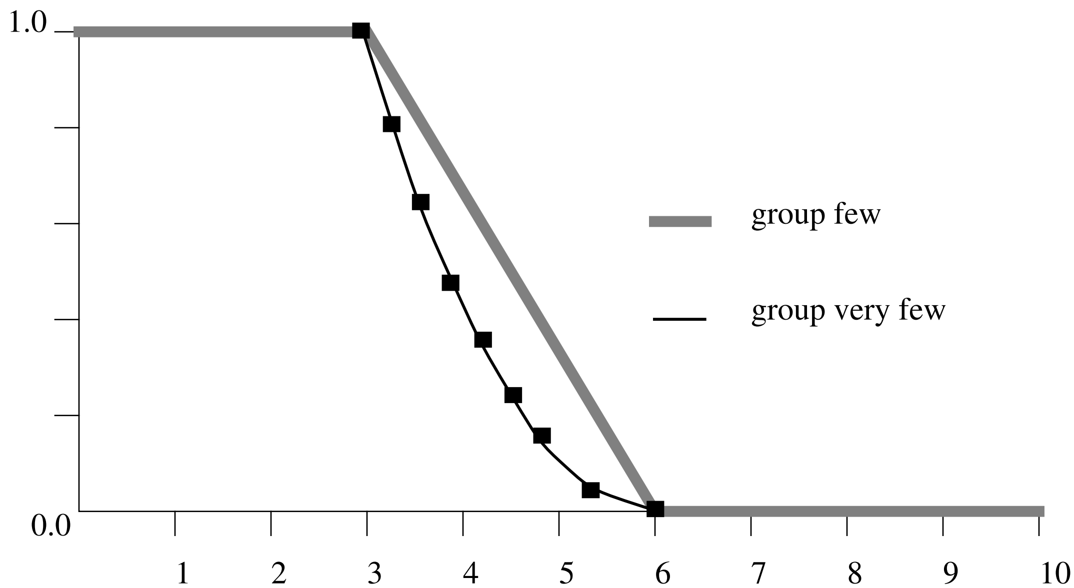


Figure 17: Modifier interpolation method

The logical operator OR may be used between terms to form the union of two fuzzy sets. The syntax of the OR operator is:

[<*modifier*>...<*modifier*>] <*primary term*> OR [<*modifier*>...<*modifier*>] <*primary term*>

where [] denotes optional modifiers.

Example 20

```
(group few OR many)          ;OR between two primary terms
(group very few OR many)     ;OR, modifiers and primary terms
(group very few OR few OR many) ;multiple ORs
```

The NOT operator takes the complement of a fuzzy set (i.e. $\bar{f}(x) = 1 - f(x)$). The NOT operator may only occur at the beginning of a <*value of fuzzy variable*> expression, with the rest of the expression enclosed by square brackets **[ ]**. Note that the brackets **[** and **]** must be separated from the other symbols in the expression by delimiters - in this case spaces, tabs, newlines, other whitespace, or symbols such as (and).

Example 21

```
(group NOT [ few ])       ;the spaces are important!
(group NOT [ very few OR many ])
```

A formal development of the grammar of linguistic variables used in fuzzy deftemplates, asserts and patterns may be found in Appendix A.

5.3 Using Fuzzy Variables in Deffacts Constructs

The syntax of the deffacts construct has been extended to allow fuzzy facts to 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 would 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>*]

where the certainty factor is optional. If not specified a CF of 1.0 is assumed.

The *<description of fuzzy set>* may be either a singleton or standard representation as used in the *deftemplate* statement (see section 5.1) with the added flexibility that any numerical argument may be replaced by a global variable or function as per deffact fact definitions of non-fuzzy facts.

Example 22

```
(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))
)
```

```
(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)
)
```

NOTE that no variables (single or multifield) may be used in fuzzy pattern specifications. Only the single field wildcard specifier (?) may be used when the value of the fuzzy fact to be matched is of no consequence. See Example 28.

5.4 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 fact> | <fuzzy fact expression>)
        [CF <certainty factor> | <certainty factor expression>]
)
```

where the certainty factor is optional. If not specified a CF of 1.0 is assumed.

5.4.1 Assertion of fuzzy facts

The *<fuzzy fact>* may be of any form allowed in section 5.1.

Example 23

```
(assert (group few)) ;assuming deftemplate group of section 5.2
(assert (group NOT [ very few OR many ] ))
```

5.4.2 Assertion of fuzzy fact expressions

A *<fuzzy fact expression>* has the syntax:

(*<fuzzy variable name>* *<description of fuzzy set>*)

The *<description of fuzzy set>* may be either singleton or standard representation as used in the *deftemplate* statement (see section 5.1) with the added flexibility that any numerical argument may be replaced by a variable or function as for standard fact asserts.

Example 24

```
assuming deftemplate group of  Example 18
(assert (group (1 0) (5 1) (7 0)) )    ;singleton description
(assert (group (z 4 8)))               ;standard description

(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 is as described in the CLIPS reference manual.

Example 25

```
(assert-string "(group (z 4 8))")
(assert-string "(group (s 4 (+ 5 3)))")
```

5.4.3 Assertion of certainty factors

The <*certainty factor*> is a constant numeric value between 0.0 and 1.0.

Example 26

```
(assert (somefact) CF 0.8)             ;with CF of 0.8
```

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

5.4.4 Assertion of certainty factor expressions

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

Example 27

```
( 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  5.7
)
```

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

The fact variable (?<*fact-var*>) should be bound on the LHS as described in the CLIPS Reference Manual, or it may be the integer value of the fact number (i.e. the index of the fact on the fact list).

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

Example 28

Suppose that fact 1 (a fuzzy 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 the following rule is executed it will print 'Temperature is 32.5'.

```
(defrule defuzzification
    ?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
)
```

5.6 Certainty Factors of Rules

This section will discuss the syntax for declaring the certainty factor of rules (certainty factors for facts are discussed as appropriate in sections 5.3 and 5.4).
The certainty factor of a rule may be declared in a manner similar to declaring the rule salience:

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

The <*certainty factor*> is a number between 0 and 1.

Example 29

```
(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!")
)
```

5.7 Accessing Parameters

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 add capability to access the matched fuzzy facts and control thresholds for rule firings.

5.7.1 Accessing the universe of discourse

Command: get-u

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

where ?<*fact-var*> is a fact variable bound on the LHS of a rule, or the number of the fact on the fact list.

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 just returns "<*from*> - <*to*>".

Command: get-u-from

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

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

Command: get-u-to

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

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

Command: get-u-units

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

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

5.7.2 Accessing the fuzzy set

Command: get-fs

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

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

Command: get-fs-length

Syntax: (get-fs-length ?<*fact-var*>)

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*>)

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

Purpose: Returns the x-coordinate of the i^{th} 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*>)

Purpose: Returns the y-coordinate of the i^{th} pair in the fuzzy set as a floating point value.

Example 30

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 3 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)
)
```

5.7.3 Accessing the certainty factor

Command: get-cf

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

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

Example 31

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".

5.7.4 Accessing the threshold certainty factor

Command: threshold

Syntax: (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.

5.7.5 Setting the Rule CF Evaluation Behavior

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 rule's certainty factor to be evaluated at time of rule definition (compilation). The value when-activated forces the rule's certainty factor 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. This is similar to the get-salience-evaluation function of CLIPS.

5.8 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-result +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)
FIRE1 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) (0.1495495495495495 0.09909909909909909)  )

==> Activation -1 get-crisp-value-and-print-result: f-2
FIRE2 speed-ok: f-1
~~~ f-2 (speed_change ???) CF 0.63
        ( (0.0 1.0) (0.1 0.1) (0.1333333333333333 0.06666666666666668)
          (0.1495495495495495 0.09909909909909909)  )
                                                [ ~~~ means fuzzy set being modified ]
FIRE3 get-crisp-value-and-print-result: f-2

Change speed by a factor of:  0.3553202565269306

<== Focus MAIN
3 rules fired        Run time is 0.8449999999993452 seconds.
3.550295857990917 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>
```

6.0 Continuous Systems

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

a. 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.

b. 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 3 arguments: a string name of the function to be added; a pointer to a function to be executed; and a priority for the function.

7.0 CLIPS Functionality within FuzzyCLIPS

7.1 Modifying and Duplicating Facts

The CLIPS functions modify and duplicate are not relevant for fuzzy facts. These functions will always return FALSE when used with fuzzy facts.

7.2 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.

7.3 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.

7.4 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.

8.0 Changes from the CLIPS 4.3 Version of FuzzyCLIPS

8.1 Removal of Fuzzy Certainty Factors

Previously one was able to set the certainty factor for a fact or rule to be a crisp number between 0 and 1 or to be a fuzzy number using expressions such as around 0.7 or close_to 0.3. Now it is only possible to use crisp numbers for certainty factors for facts and rules. It was felt that the overhead in computation and the internal complexity of carrying fuzzy numbers for certainty factors was not justified.

8.2 New Way to Specify Certainty Factors for Facts

Previously certainty factors for facts and rules were identified differently. For rules the token CF was used to identify a certainty factor.

```
(defrule rule-with-CF
    (declare (CF 0.7))
    .
    .
    .
)
```

For facts the token # was used to identify a certainty factor.

```
(assert (fact1) # 0.5)
```

Now the token CF is used in both cases for consistency and better readability.

```
(assert (fact1) CF 0.5)
```

8.3 Defining Fuzzy Variables (Deftemplate vs. Defuzzy)

In the previous versions of FuzzyCLIPS a **deffuzzy** construct was used define a fuzzy variable. Now the **deftemplate** construct is used for defining fuzzy templates and non-fuzzy templates. The format of the deftemplate for fuzzy variables is identical to the previous deffuzzy construct (only the keyword deffuzzy is changed to deftemplate). Also note that in the previous version it was necessary to include a list of modifiers for a fuzzy variable even if the list was empty. Now the list can be omitted if empty.

```
e.g.   (deffuzzy temp                 (deftemplate temp
          0 100 degrees         vs.      0 100 degrees
          ((low (z (0 5)))               ((low (z (0 5)))
          ()                          )
       )
```

8.4 Removal of Accessing Function list-lv

The function list-lv is no longer required. The CLIPS functions list-deftemplates, ppdeftemplate, and get-deftemplate-list will address fuzzy deftemplates as well as standard CLIPS deftemplates.

8.5 Addition of get-CF-evaluation and set-CF-evaluation Functions

The certainty factor for a rule can be dynamically evaluated in a fashion similar to the salience for rules. Two functions similar to get-salience-evaluation and set-salience-evaluation have been added. See 5.7.5.

8.6 Operation of moment-defuzzify and maximum-defuzzify

The functions moment-defuzzify and maximum-defuzzify previously calculated the defuzzified value of a fact and asserted a fact of the form:

```
(crisp fuzzyvar 45.5)
```

Now these two functions simply return a number which is the defuzzified value of the fuzzy fact. So for example one might do the following:

```
(bind ?val (moment-defuzzify ?fuzzyfact))
                or
(assert (crisp fuzzyvar (moment-defuzzify ?fuzzyfact)))
```

The latter option produces the same behaviour as the previous version of FuzzyCLIPS.

9.0 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 which is used to defuzzify fuzzy facts. As pointed out in [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

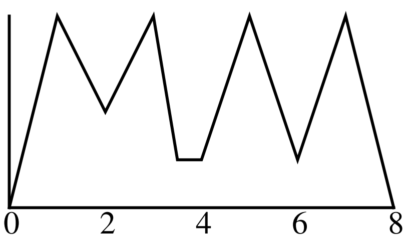
a) if weights are all zero

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

b) if not all weights are zero

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

Consider the following two examples:

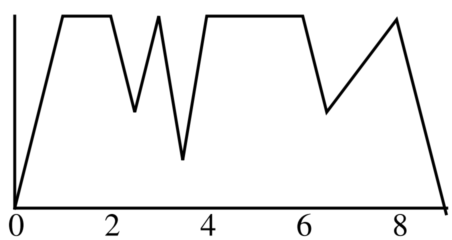


max values at 1, 3, 5 and 7
(all weights 0)

Calc defuzzified values as

(1+3+5+7)/4 = 4.0

same as current method



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

Calc 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:

```
(defrule r1
  (logical (x))
=>
  (assert (temp low))
)
```

```
(defrule r2
  (logical (y))
=>
  (assert (temp medium)
)
```

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 2 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 may need improvement is related to the requirement that all x values in a fuzzy set specification must be strictly increasing. This means that the shape of a fuzzy set may not have any vertical lines. As a special case of this we interpret a fuzzy set description with a single point to be a horizontal line and not to be a crisp value in fuzzy set format. Without going into any details, we might just say that the problem is not without some difficulties in interpretation and implementation.

Bug reports and any suggestions for modifications and further extensions are welcome.

10.0 References

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

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

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

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

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

6 K.P. Adlassing, G. Kolarz, Representation and Semiautomatic Acquisition of Medical Knowledge in Cadiag-1 and Cadiag-2, Computers and Biomedical Research, vol.19, 1988, p.63-79.

7 T. Whalen, B. Schott, and F. Ganoe, Fault Diagnosis in Fuzzy Network, Proc. 1982 Int. Conf. Cybernetics and Society, IEEE Press, New York, 1982.

8 J. Buckley and W. Siler, Fuzzy Operators for Possibility Interval Sets, Fuzzy Sets and Systems, vol.22, 1987, p.215-27.

9 J.F. Baldwin, Evidential Support Logic Programming, Fuzzy Sets and Systems, vol.24, 1987, p.1-26.

10 K.S. Leung, W.S.F. Wong, and W. Lam, Application of a Novel Fuzzy Expert System Shell, Expert Systems, 1989, vol.6, no.1, p.2-10.

11 Z.A. Sosnowski, FLISP - A Language for Processing Fuzzy Data, Fuzzy Sets and Systems, vol.37, 1990, p.23-32.

12 L.A. Zadeh, Fuzzy Sets, Information and Control, 8, 1965, 338-383.

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

14 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”, April 6-8, 1988,Vienna, p.71-4.

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

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

17 M. Cayrol, H. Farency and H. Prade, Fuzzy Pattern Matching, Kybernetes, 11, 1982, pp.103-6.

18 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, p.117-136.

19 Tzi-cker Chiueh, Optimization of Fuzzy Logic Inference Architecture, Computer, May 1992, 67-71.

20 T. Whalen, B. Schott, Issues in fuzzy production systems, Int. J. Man-Machine Studies, 19, 1983, 57-7.

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

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

Appendix A: Syntax of Linguistic Variables

A linguistic variable has been defined by L.A. Zadeh as a system:

<L, T(L), X, G, M>

where L, T(L) (or for short T), X, G and M denote respectively:

- L - name of the variable;
- T - set of labels of a fuzzy subset of the universe of discourse;
- X - universe of discourse;
- G - syntactic rules, defined as generative grammar, which define the well-formed sentences in T;
- M - semantics, which consists of rules by which the meaning of the sentences in T can be determined.

Modifiers such as *very*, *more or less* etc. are common in approximate reasoning. They may be defined as follows. For any value of linguistic variable $t \in T$ defined by the membership function

$$\mu : U \rightarrow [0,1]$$

modifier is described by the formula

$$\textit{modifier } t = \mu^{m}{}_{t}$$

where m is the number connected with the given modifier. For example:

$$\textit{very } t = \mu^{2}{}_{t}$$
$$\textit{more or less } t = \mu^{0.5}{}_{t}$$

In FuzzyCLIPS the syntactic rules G and semantics M of a linguistic variable are defined as follows.

G is a grammar:

$$G = \langle V, \Sigma, P, \delta \rangle$$

The set of terminal symbols V consists of primary terms t_i , modifiers m_j , logical operators NOT and OR, and two special symbols [and]. Thus we can write

$$V = \{t_1 ,t_2 ,..., t_n , m_1 , m_2 ,..., m_k , NOT, OR, [,] \}$$

The set of non-terminal symbols Σ is defined in FuzzyCLIPS as follows.

Σ = { δ, <value of linguistic variable>, <modifier>, <primary term>,
<simple expression>, <expression> }

The set of productions P consists the list of productions:

P :

δ ::= <value of linguistic variable>

<value of linguistic variable> ::=
<expression> | NOT [<expression>] | NOT <simple expression> |
<description of fuzzy set>

<expression> ::=
<simple expression> OR <expression> |
<simple expression>

<simple expression> ::= <modifier><simple expression> | <primary term>

<primary term> ::= t_1 | t_2 | . . . | t_n

<modifier> ::= m_1 | m_2 | . . . | m_k

The semantics M is defined as a function, which associates with each sentence of the language L(G), generated by the grammar G, its meaning as a fuzzy set in a universe of discourse U. So we can write

M = L(G) --> F(U)

This function is defined as follows.

M(<primary term>) = M(t_i) = μ_{ti} $\quad$ i=1,2,...,n

The semantic value of the primary term is a fuzzy set defined by the membership function μ_{ti} .

M(<simple expression>) = M (<primary term>)
if <simple expression> = <primary term>

M(<simple expression>) = f_j (<simple expression>)
if <simple expression> = <modifier><simple expression>

$f_j : F(U) \rightarrow F(U)$, is a function connected with modifier m_j.

M(<expression>) = M(<simple expression>)
if <expression> = <simple expression>

M(<expression>) = M(<simple expression>) $\cup$ M(<expression>)
if <expression> = <simple expression> OR <expression>

M(<value of linguistic variable>) = M(<expression>)
if <value of linguistic variable> = <expression>

M(<value of linguistic variable>) = $\overline{M}$(<expression>)
if <value of linguistic variable> = NOT [<expression>]

M(<value of linguistic variable>) = $\overline{M}$(<simple expression>)
if <value of linguistic variable> = NOT <simple expression>

M(<value of linguistic variable>) = μ
if <value of linguistic variable> = <description of fuzzy set>

Where for $\eta \in \overline{\Sigma}$, $M(<\eta>)$ denotes the compliment of the fuzzy set $M(<\eta>)$ and $\cup$ denotes a union of fuzzy sets.

Appendix B: 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.

Shower model

1. The volumes of the pipes are zero.
2. Water mixes perfectly at point X (see Figure B1).
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.

Shower model equations

$Tc \in [0, 35]$ °C
$Th > Tc \qquad Th \in [0, 100]$ °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.

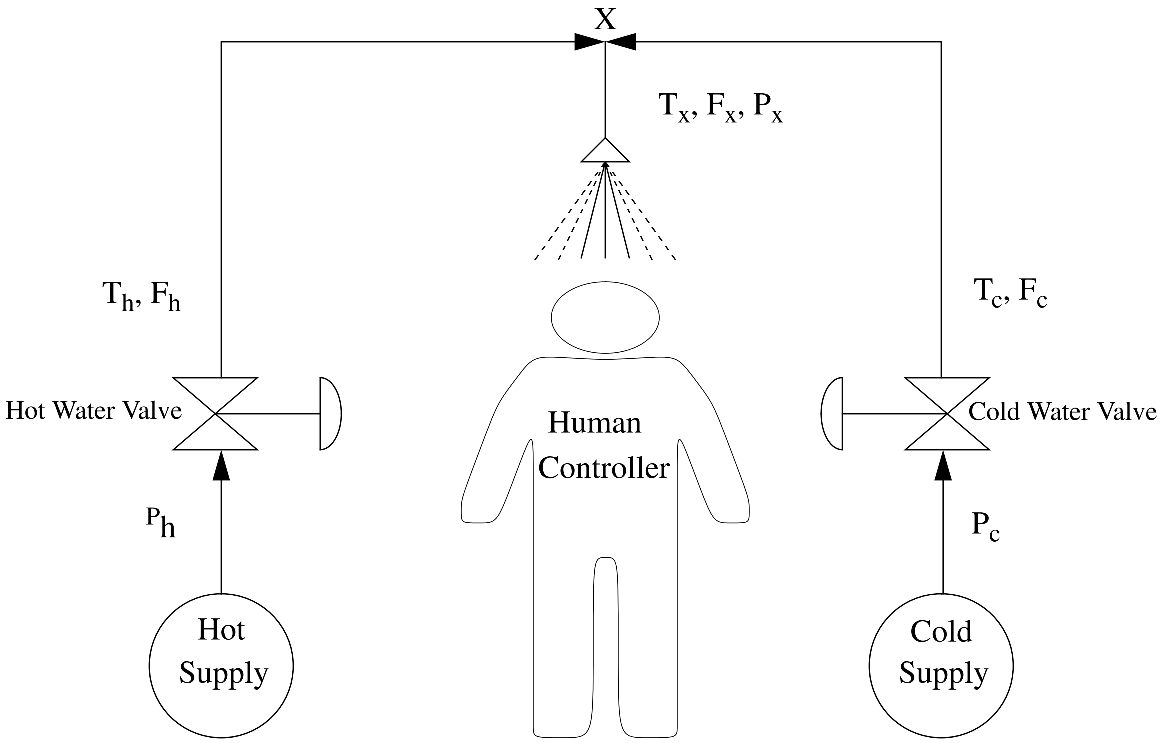
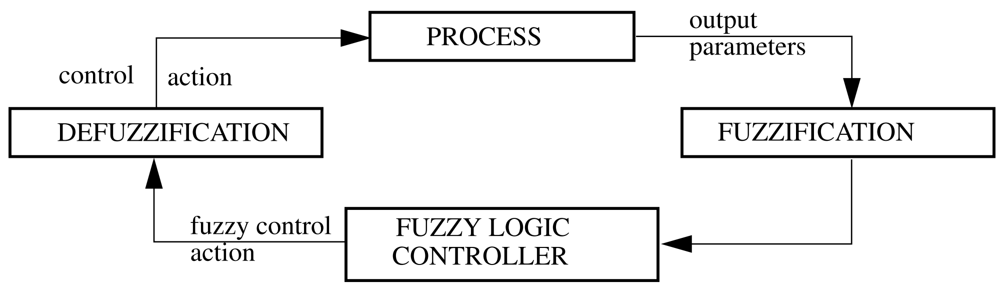


Figure. B1.

<u>Shower control objectives</u>

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 liters/minute 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).

Fuzzy control loop



Text based version (No graphical interface - shwrNOUI.clp)

Steps to follow to run fuzzy shower example (Unix version)

1. Start a version of FZ_CLIPS.
2. Load shower example (load “shwrNOUI.clp” from 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 liters/minute, and water temperature between 34 - 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.

Graphical Interface Version of Shower Problem

The graphical interface version of the shower controller will run only on a Sun Sparc machine with Openwin version 3.0 or later (with NeWS/X11). This is because we used the GUIDE (Devguide) software with GNT (NeWS Toolkit Code Generator). Devguide is used to interactively create most of the user interface. GNT translates the output from Devguide (Shower.G) into ‘C’, cps, and postscript files that form the code for the interface (Shower.c, tmp/Shower.h, srcShower.ps, and tmp/Shower_c.cps, tmp/Shower_m.c, etc.).

In Shower.c and Shower_m.c a number of changes are made:

1. The main routine is renamed to UI_init so that it can be embedded into CLIPS.
2. The ‘while’ loop at the end of the main routine (just before the ‘return 0’ statement is commented out (control of events is taken care of by other code found in the simulation part of the extensions).
3. In the routine runShower the statement ‘setShowerPipingCanvasPaint()’ is added before ‘Allocate_Shower_Tags()’ to draw a graphic in the interface window.

4. In the various locations commented with /* Custom code goes here */ calls to appropriate Callback routines have been added as needed. The code for these Callbacks is found in ShowerCallbacks.c.

src/Shower_c.cps is custom code to add special ‘cdef’ routines needed to complete the interface functionality.

The files Shower_audio.c, audio_device.h, audio_errno.h, audio_hdr.h, libaudio.a, libaudio.h, and flush.au support a very limited sound capability in the program.

Simulate.h and Simulate.c provide a simple event based simulation capability. This is customized by adding code specific to the simulation being developed. Files Events.h, EventNames.h, ShowerSimulate.c and ShowerSimulate.h provide the specific code for the shower simulation.

If logging is turned on during the simulation (described later) a file is created (extension of .log) with data that records system state when changes take place (time, temperature of hot water, temperature of cold water, pressure of hot water, pressure of cold water, position of cold water valve, position of hot water valve, flow of output, and temperature of output). A second file (extension is .stat) is created to show some statistics of the run, such as the number of moves of the hot and cold valves required to get the temperature and the flow back into the comfortable range. An example file is shown below:

	StartTime	Temp	Flow	ElapsedTime	Moves
0	10.730	0.000	0.000	0.660	10
1	17.290	38.333	10.968	0.100	1
2	21.580	42.228	9.820	0.340	5
3	25.660	46.106	8.890	0.400	3
4	29.990	27.039	16.582	0.520	7
5	32.260	32.212	14.945	0.220	3
6	36.580	30.394	14.573	0.270	2

Avg. Time to Recover: 0.358571
Avg. Number Moves: 4.428571

In order to use a plotting program called ‘xvgr’ (available via anonymous ftp from ese3.ese.ogi.edu [129.95.20.62] in pub/grtool) we supply the program formatData (formatData.c) to convert the time information into a suitable form (julian) and to split the data into separate sets. The formatData program takes the name of the log file (without the extension (.log assumed) and creates a file with the extension .dat. The data can be plotted with:

```
xvgr -p ShowerGraph.par xxxx.dat
```

where xxxx.dat is the converted file and ShowerGraph.par is a parameter file used by xvgr to format the plotting window correctly.

The Graphical Interface

The diagram in Figure 2 shows the GUI interface for this problem. The Shower temperature and flow rate are shown in gauges along the top. The hot water and cold water valve positions are shown on opposite sides in sliders. These sliders can be moved only when in manual mode. The temperature of the hot and cold supplies are shown on opposite sides as well in sliders. These can be adjusted at any time (manual or auto mode). The pressures of the hot and cold supplies are shown in gauges along the bottom. The mode selection allows one to operate in auto or manual mode (by default the system will start in manual mode). In manual mode the temperature and flow of the output will respond to your changes in the sliders and selections under the 'change pressure' menu. In auto mode the fuzzy logic controller is in control of the hot and cold water valves and tries to maintain the correct operating ranges for the system. The RealTime Scale menu is not operable. The Data Logging menu allows you to start recording data in a file (extension .log is added) and to stop the logging. The Change Pressure menu provides a number of selections that make events occur which have an effect on the water pressure. E.g. flush toilets (hence the audio stuff!), turn on/off the garden hose, etc.

The shower temp gauge will show green when the temp is in range, red when too high and blue when too cold. Similar reactions hold for the shower flow gauge.

How to Run the Shower Example with the User Interface

1. Start FZ_XCLIPS_shower (a version supplied with the Sun distribution) or use a version compiled with the flag ShowerProblemUI in xmain.c or main.c set to 1.
2. Load the example clips code -- (load "showerUI25.clp")

   a. showerUI25.clp is a 25 rule controller, showerUI8.clp is an 8 rule controller -- use either one.

   b. the file flush.au should be in your current directory
3. Enter -- (reset)
4. Enter -- (run -2)   ; keeps CLIPS looping even when no rules
5. Interact with the window that contains the user interface (normally will turn on auto mode and try various things).
6. A control-C (from within the clips window) will stop the clips loop and allow you to exit clips.

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

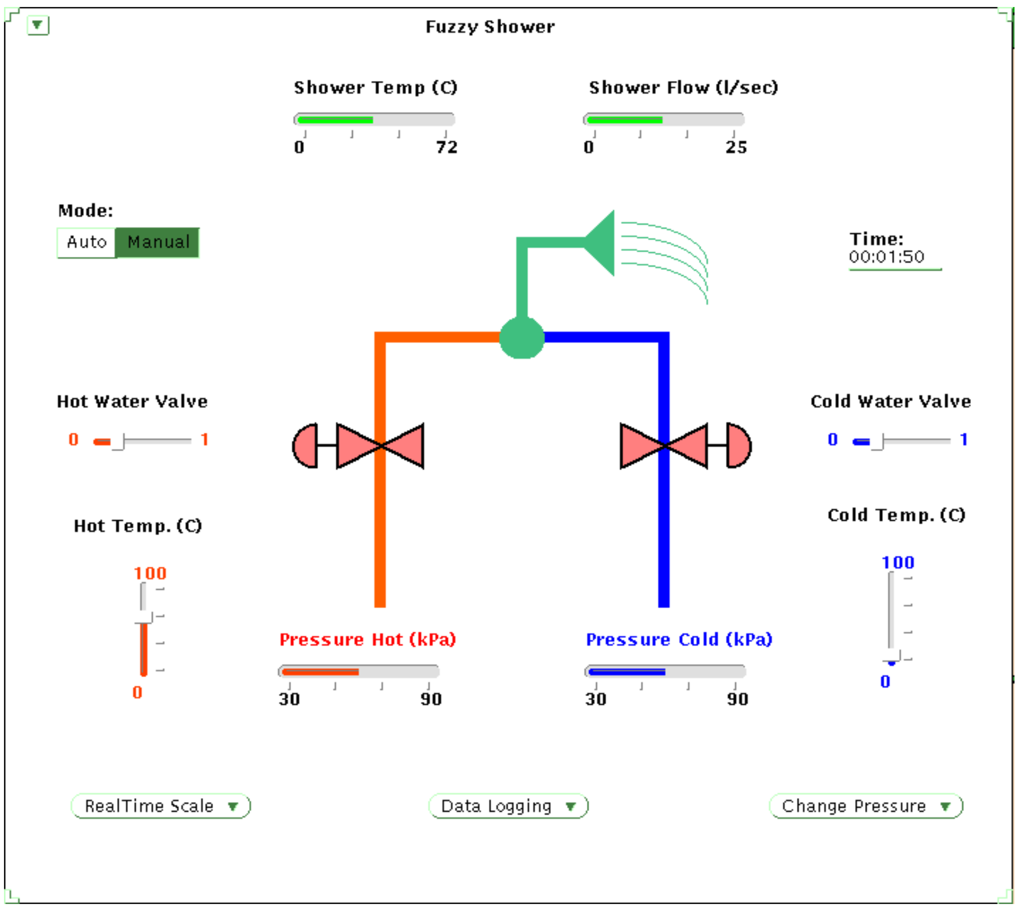

Fuzzy Shower
Shower Temp (C)
0
72
Shower Flow (l/sec)
0
25
Mode:
Auto
Manual
Time:
00:01:50
Hot Water Valve
0
1
Cold Water Valve
0
1
Hot Temp. (C)
100
0
Cold Temp. (C)
100
0
Pressure Hot (kPa)
30
90
Pressure Cold (kPa)
30
90
RealTime Scale
Data Logging
Change Pressure

Figure B2