McESE-FranzLISP: McMASTER EXPERT SYSTEM EXTENSION OF FranzLISP

F. Franek

Technical Report no TR-22/88

Department of Computer Science and Systems

McMaster University 1988

McESE-FranzLISP: McMASTER EXPERT SYSTEM EXTENSION OF FranzLISP

F. Franek 1)

Dept. of Comp. Sci. & Systems
McMaster University
Hamilton, Ontario
L8S 4K1 Canada

TABLE OF CONTENTS:

Acknowledgement:

1) Research supported by SERB 5-26397 and NSERC OGP0025112 research grants.

ABSTRACT

McESE - McMaster Expert System Environment is a software tool designed to help create problem-specific shells with imprecise and incomplete knowledge, and fast and compact expert systems applications in a particular programming language. This is achieved by "extending" the programming language with a set of commands allowing communication with McESE knowledge bases and McESE special software (see [FB], or [FB1]). In this paper implementational details of FranzLISP extension (called McESE-FranzLISP) are presented and discussed.

(1) INTRODUCTION.

As described in [FB] and [FB1], McESE is a programming environment for building of expert systems. Its main features include the ability to specify the way of handling uncertainty for the whole knowledge base, or for each rule separately, to utilize compiled rule-form knowledge bases for fast inferring, and the ability to build the expert system in a particular programming language. The software of McESE is written in the programming language C for two reasons; compact code that leads to fast execution, and the ability to program in C low-level tasks efficiently enough. In this paper the extension of FranzLISP is described and discussed. From now on we may use term lisp instead of FranzLISP for the sake of brevity.

The task of extending FranzLISP to McESE-FranzLISP consisted of two major tasks:

- to make the software of McESE written in C an integral part of the lisp system with sufficient communication between McESE software and lisp system without disturbing the lisp system too much and without a need of extensive re-writing of the existing McESE software. It must be ensured that any of the C functions of McESE required can be invoked from within lisp, and conversely, any lisp function, be it built-in or user defined, interpreted or compiled, can be invoked from within the C functions;

- to make the non-lisp data structure of compiled knowledge base (we shall call it knowledge tree in accordance with [FB] and [FB1]) an integral part of lisp system accessible from within lisp, but protected from lisp's garbage collector as long as needed.

Of course, there were some other problems to be resolved as well, e.g. how to simplify invocation of McESE, how to protect the components of McESE from the user and so on.

(2) HOW TO MAKE McESE SOFTWARE AN INTEGRAL PART OF FranzLISP.

Since the kernel of FranzLISP in UNIX is written in C, the major lisp function eval can be directly invoked from any C program loaded into the lisp system. To speed up the loading of McESE software into the lisp system, the whole McESE system is first compiled as a single source file, and then loaded into the lisp system using lisp built-in function cfasl. cfasl in fact works as a dynamic on-line loader and so only one linking to the lisp system is necessary (the required C functions may be loaded one at a time, but then many linkings are required slowing down the whole procedure). The individual C functions needed are then made available to the lisp system (in fact made into lisp objects) using lisp built-in function getaddress:

```
(cfasl '/u0/rsch/mcese/lisp/mceses.o '_compiler 'f__100 "c-function")
(getaddress '_getsizekbtree 'f__101 "c-function")
(getaddress '_loadkbtree 'f__102 "c-function")
.
.
.
(getaddress '_helpcop 'f__118 "c-function")
```

Any of the specified C function (e.g. compiler, or getsizekbtree) are now integral part of the lisp system and can be invoked from within any lisp function (e.g. compiler is known in the lisp system as f__100, while getsizekbtree is known as f__101). The values they returned are formatted by the lisp system into lisp values.

A more complicated problem was the problem of invocation of arbitrary lisp functions from within C functions.

First, there was the need of invocation of cvpf's (written in lisp) from within the inference engine (written in C). Cvpf's must have flonum arguments, and must return a flonum (see [FB], [FB1]). A special C function invoke1 provides the service. When McESE system is loaded into the lisp system, a global symbol f__1 is bound to the list (eval (fake addr0)) where addr0 is the address of lisp function f__12. The C function invoke1 is passed f__1 as an argument, invoke1 evaluates f__1 using eval, and that invokes f__12 which creates and returns a list
(eval (list (fake addr1) flonum ... flonum)) back to invoke1. invoke1 pops from a special stack address of the desired lips function and inserts it into the list as addr1, then it pops from the special stack one be one all (flonum, and hence double in C) arguments and inserts them into the list instead of flonum's. Finally, invoke1 terminates the list where needed. The last step involves evaluating the list using eval, which in fact invokes the

required lisp function with the specified arguments. The result of the invocation (again a flonum, and so double in C) is returned to invoke1. When a lisp function is to be invoked from a C function, the address of that function and its arguments are pushed on the special stack, and invoke1(f__1) is called, invoke1 in turns invokes the required lisp function (as described above) and passes the value returned by the lisp function into the C function.

Second, there was the need of invocation of predicate service procedures (written in lisp) from within the inference engine (written in C). Predicate service procedures may have any arguments, but must return a flonum (see [FB], [FB1]). A special C function invoke3 provides the service. When McESE system is loaded into the lisp system, a global symbol f__3 is bound to the list (eval (fake addr0)) where addr0 is the address of lisp function f__32. The C function invoke3 is passed f__3 as an argument, invoke3 evaluates f__3 using eval, and that invokes f__32 which creates and returns a list
(eval (list (fake addr1) (fake addr) ... (fake addr))) back to invoke3. invoke3 pops from a special stack address of the desired lips function and inserts it into the list as addr1, then it pops from the special stack one be one addresses of all arguments and inserts them into the list instead of addr's. Finally, invoke3 terminates the list where needed. The last step involves evaluating the list using eval, which in fact invokes the required lisp function with the specified arguments. The result of the invocation (a flonum, and so double in C) is returned to invoke3. When a lisp function is to be invoked from a C function, the address of that function and its arguments are pushed on the special stack, and invoke3(f__3) is called, invoke3 in turns invokes the required lisp function (as described above) and passes the value returned by the lisp function into the C function.

Third, the explanation component of McESE inference engine required to obtain in the form of a C string the lisp print form of objects used for the particular inference (see [FB], [FB1]). invoke7 provides the service. When McESE system is loaded into the lisp system a global symbol f__7 is bound to the list
(get_pname (maknam (explode (fake addr)))))), invoke7(f__7) pops the address of the object from a special stack, then using eval evaluates f__7 which returns the desired string.

Any other communication between lisp and McESE programs, e.g. as in mcese:open function where the lisp code provides to the C code addresses of required cvpf's and predicate service procedures, is facilitate be list created and interned in the lisp system, passed to the C coded, and either used or modified by the C code and then passed back to the lisp system. This kind of communication allows the full range necessary and does not intrude upon lisp at all. Given the fact that FranzLISP's kernel is written in C, had more technical and implementational details of FranzLISP been available, we could have modified our C code to extend the original kernel directly. This approach did not seem appropriate for the actual different implementations on different machines may differ, and for to much of intrusion to the system. The means we opted for instead do slow the execution a bit (which is not of such a crucial importance given the slow execution of lisp in general) but allow for extension of lisp independent from the implementational details of FranzLISP's kernel.

(3) HOW TO MAKE McESE COMPILED KNOWLEDGE BASES PART OF FranzLISP.

When McESE compiler parses and compiles a RSET (see [FB], [FB1]), it all takes place within a single call of the C function compiler. Thus all space required is "malloced" in the C program and the whole knowledge tree built there. When finish, compiler records the knowledge tree in a disk file. The space used is "freed" and control returned back to the lisp system. This poses no problems as far as memory management. A more complicated situation arises when this compiled knowledge tree is to be loaded to main memory from disk. It must stay in the lisp system as long as the user desires while some other C and lisp functions are executed. Since the garbage collector of FranzLISP is invoked automatically when the system has not enough memory available, the knowledge tree must be protected against it. It must also be protected against intrusion by any other function of the system. On top of it, when the user does not need the knowledge tree any longer, the memory it takes must be made available to the system again. These requirements were satisfied by "storing" the knowledge tree within a lisp data structure, so-called immediate vector. The C code for loader had to be altered slightly and partitioned into two separate functions. The first one, getsizekbtree, returns to the lisp system the size of the knowledge tree to be loaded (this information is stored in the disk file containing the knowledge tree). Then the lisp system creates an immediate vector of the required size and binds it to rsetid (rset id). The address of the immediate vector is then passed to the C program which loads the knowledge tree there absolutizing all address links in the tree (see [FB], [FB1]). Since immediate vectors carry binary data, there is no problem with the "contents" being the knowledge tree. As long as this immediate vector is bound to the rsetid, the systems considers it active and hence it is protected from the garbage collector as well as all lisp functions. Since the rsetid symbolic name is of the form f__RSETID# (# stands for a number), and symbols beginning with f__ are not available to the user, the likelihood of the user intruding upon the knowledge tree is rather slim, as he has no knowledge of the address of the immediate vector. On the other hand, when the knowledge tree is unloaded, the rsetid is unbound (and removed from oblist), and so the immediate vector storing the knowledge tree is not considered active by the system and hence it is for grabs by garbage collector. From the point of view of the inference engine and other components of McESE accessing the knowledge tree, all they need to know is the beginning of the tree and correct address links within the tree. Thus the fact that the knowledge tree is inside a lisp object matters not to them.

(4) McESE-FranzLISP SYSTEM BEHAVIOUR.

To invoke mcese system, first invoke FranzLISP from UNIX level by typing lisp'. When in the FranzLISP system, load in the mcese system by typing (load 'mcese). In a few seconds the system is loaded and mcese prompt mcese>> will occur:

```
maccs 1 > lisp
Franz Lisp, Opus 38.92
-> (load 'mcese)
[load /usr/lib/lisp/mcese]
[load /u0/rsch/mcese/lisp/mcese.l]
[fasl /u0/rsch/mcese/lisp/mcese1.o]
[fasl /u0/rsch/mcese/lisp/mcese2.o]
/usr/lib/lisp/nld -N -x -A /usr/ucb/lisp -T b3a00 /u0/rsch/mcese/
lisp/mceses.o -e _compiler -o /tmp/Li27513.0  -lc
[McESE-FranzLisp version 1.3 - 1988 loaded]

*** mcese: please, do not use symbols containing f__
           (such names are reserved for mcese system)
```

```
*** mcese: please, do not put files with names mcese, mcese.l
           and mcese.o in your directory (these are load files
           for mcese system)

mcese>>
```

To make the loading of McESE as simple as possible, the default directory /usr/lib/lisp is used: the file mcese consists of a single load instruction specifying what should be loaded and where it is. Thus the file /u0/rsch/mcese/lisp/mcese.l is loaded. It contains definition of the new top level and so is not compiled (when compiled it caused some erratic troubles). It also contains load instructions for /u0/rsch/mcese/lisp/mcese1.o file, which is file with all lisp functions of the McESE system (they mainly consists of lisp code surrounding C functions checking correctness of arguments and interpreting the results). For faster execution these are compiled. It is followed by load instructions for /u0/rsch/mcese/lisp/mcese2.o file, which contains all instruction for loading McESE C software into the system.

At this moment both, FranzLISP and McESE functions are fully accessible at all levels of the system. Thus:

```
mcese>> (plus 2 3)
5
mcese>>
```

Since all data structures and functions of McESE have names starting with f__, symbols (or strings) containing f__ are not available to the user:

```
mcese>> f__1

*** mcese error: using forbidden symbol f__
f__1
mcese>> (print "abc f__wrt")

*** mcese error: using forbidden symbol f__
(print "abcf__wrt")
```

There are, of course, ways how to fool the system and get an access to f__ objects, but it is hard to do so accidentally. This all is achieved by redefining the top level of the lisp system. The new top level includes the checking for f__ symbols. The system returns automatically back to this top level even from lower levels when (reset) is used.

```
mcese>> (mcese:comp 'medrset 'medrset.comp)
t
```

The above function mcese:comp successfully compiled (t was returned) rset called medrset and the compiled knowledge tree was stored in a disk file called medrset.comp . An input tracing as well as supression of error recovery may be specified.

```
mcese>> (mcese:load 'medrset.comp)
f__RSETID0
```

The above function mcese:load successfully loaded the knowledge tree from 'medrset.comp file to the lisp system. The rset id f__RSETID0 was returned.

```
mcese>> (setq x (mcese:last-load))
```

```
f__RSETID0
```

The above function mcese:last-load returns rset id of the rset most recently loaded into the lisp system.

```
mcese>> (mcese:rsetidp x)
t
```

The above function mcese:rsetidp tests if its arguments is a rset id of an active knowledge tree.

The following function mcese:discomp discompiles the loaded knowledge tree, displays it on screen, plus some vital statistics about the knowledge tree:

```
mcese>> (mcese:discomp x)

*** discomp: DISCOMPILE of COMPILED RSET

rule  med1:
      pain_in_throat  &
      .9 * hardship_to_swallow  &
      .4 * noisy_and_mouth_breathing  &
      .6 * headache  &
      .3 * fever  &
      .3 * cough
==>
      sore_throat

rule  med2:
      .8 * fever  &
      .6 * headache  &
      pain_in_throat  &
      .7 * hardship_to_swallow  &
      .6 * spotted_throat  &
      .2 * vomiting  &
      rash
-- more --
==>
      strep_throat_wrash

rule  med3:
      .8 * fever  &
      .6 * headache  &
      pain_in_throat  &
      .7 * hardship_to_swallow  &
      .6 * spotted_throat  &
      .2 * vomiting  &
      .3 * abdominal_pain  &
      .8 * ~rash
==>
      strep_throat_worash

rule  med4:
      noisy_and_mouth_breathing  &
      pain_in_throat  &
      fever  &
      .8 * hardship_to_swallow  &
      ~sore_throat  &
      ~strep_throat_wrash  &
      ~strep_throat_worash
-- more --
==>
      tonsillitis
```

```
rule  med5:
      strep_throat_wrash  &
      strep_throat_worash
== max ==>
      strep_throat

rule  med6:
      tonsillitis  &
      sore_throat  &
      strep_throat
== max ==>
      throat_trouble [>= .6]

rule  med7:
      ~throat_trouble
==>
      other_diseases

rule  med8:
      vomiting  &
-- more --
      abdominal_pain  &
      headache
== ff1 ==>
      other_diseases

*** discomp: RSET statistics follows:

level 0 chain:
next pred on level 0: pain_in_throat (address not know yet)
next pred on level 0: hardship_to_swallow (address not know yet)
next pred on level 0: noisy_and_mouth_breathing (address not know yet)
next pred on level 0: headache (address not know yet)
next pred on level 0: fever (address not know yet)
next pred on level 0: cough (address not know yet)
next pred on level 0: spotted_throat (address not know yet)
next pred on level 0: vomiting (address not know yet)
next pred on level 0: rash (address not know yet)
next pred on level 0: abdominal_pain (address not know yet)

level 1 chain:
next pred on level 1: sore_throat
next pred on level 1: strep_throat_wrash
next pred on level 1: strep_throat_worash
-- more --

level 2 chain:
next pred on level 2: tonsillitis
next pred on level 2: strep_throat

level 3 chain:
next pred on level 3: throat_trouble

level 4 chain:
next pred on level 4: other_diseases


pred abdominal_pain:
    occurs in LHS of rule med3
    occurs in LHS of rule med8

pred cough:
    occurs in LHS of rule med1
```

```
pred fever:
    occurs in LHS of rule med1
    occurs in LHS of rule med2
    occurs in LHS of rule med3
-- more --
    occurs in LHS of rule med4

pred hardship_to_swallow:
    occurs in LHS of rule med1
    occurs in LHS of rule med2
    occurs in LHS of rule med3
    occurs in LHS of rule med4

pred headache:
    occurs in LHS of rule med1
    occurs in LHS of rule med2
    occurs in LHS of rule med3
    occurs in LHS of rule med8

pred noisy_and_mouth_breathing:
    occurs in LHS of rule med1
    occurs in LHS of rule med4

pred other_diseases:
    occurs in RHS of rule med7
    occurs in RHS of rule med8

pred pain_in_throat:
-- more --
    occurs in LHS of rule med1
    occurs in LHS of rule med2
    occurs in LHS of rule med3
    occurs in LHS of rule med4

pred rash:
    occurs in LHS of rule med2
    occurs in LHS of rule med3

pred sore_throat:
    occurs in LHS of rule med4
    occurs in LHS of rule med6
    occurs in RHS of rule med1

pred spotted_throat:
    occurs in LHS of rule med2
    occurs in LHS of rule med3

pred strep_throat:
    occurs in LHS of rule med6
    occurs in RHS of rule med5

pred strep_throat_worash:
-- more --
    occurs in LHS of rule med4
    occurs in LHS of rule med5
    occurs in RHS of rule med3

pred strep_throat_wrash:
    occurs in LHS of rule med4
    occurs in LHS of rule med5
    occurs in RHS of rule med2

pred throat_trouble:
    occurs in LHS of rule med7
    occurs in RHS of rule med6

pred tonsillitis:
    occurs in LHS of rule med6
    occurs in RHS of rule med4

pred vomiting:
    occurs in LHS of rule med2
    occurs in LHS of rule med3
    occurs in LHS of rule med8

rule med1 uses built-in CVPF weighted cumulative evidence
-- more --
rule med2 uses built-in CVPF weighted cumulative evidence
rule med3 uses built-in CVPF weighted cumulative evidence
rule med4 uses built-in CVPF weighted cumulative evidence
rule med5 uses built-in CVPF max
rule med6 uses built-in CVPF max
rule med7 uses built-in CVPF weighted cumulative evidence
rule med8 uses CVPF ff1

function ff1 has 3 arguments (address not known yet)
t
```

Since the above "discompiled" code can be stored in a disk file instead of being displayed on the screen, mcese:discomp provides a convenient means to reconstruct a RSET from the compiled version of it if need be.

The following function mcese:unload releases the space taken by the compiled RSET, note that after unloading mcese:last-load indicate that there are no more loaded compiled RSET's in the lisp system:

```
mcese>> (mcese:unload x)
t
mcese>> (mcese:last-load)
nil
```

The following function mcese:open loads a compiled RSET from the specified disk file to the lisp system (or if an rsetid is used instead it knows not to load it), then it loads the (user specified) corresponding FSET, and provides all addresses necessary for the complete knowledge tree. If all proceeded correctly, a kbid (knowledge base id) of the form f__KBID# is returned:

```
mcese>> (setq x (mcese:open 'medrset.comp 'medfset))
[load medfset]
f__KBID0
```

The following function mcese:last-open returns kbid of the knowledge tree opened most recently:

```
mcese>> (mcese:last-open)
f__KBID0
```

The following function mcese:kbidp checks if its argument is a kbid of an active knowledge tree:

```
mcese>> (mcese:kbidp x)
t
```

The function mcese:display is like mcese:discomp, only it shows the addresses in the vital statistics part:

```
mcese>> (mcese:display x)

*** discomp: DISCOMPILE of COMPILED RSET

rule  med1:
      pain_in_throat  &
   .
   .
   .

*** discomp: RSET statistics follows:

level 0 chain:
next pred on level 0: pain_in_throat (address: 667356)
next pred on level 0: hardship_to_swallow (address: 667416)
  .
  .
  .

rule med8 uses CVPF ff1

function ff1 has 3 arguments (address: 666944)
t
```

The following functions mcese:show-inc (mcese:set-inc respectively) returns the inconsistency level tolerance (sets the inconsistency level tolerance to the given value respectively) of the specified knowledge base:

```
mcese>> (mcese:show-inc x)
1.0
mcese>> (mcese:set-inc x .7)
t
mcese>> (mcese:show-inc x)
0.7
```

The following functions mcese:show-alarm (mcese:set-alarm respectively) returns the symbolic name of the alarm function (sets the symbolic name of the alarm function to the given name respectively) of the specified knowledge base:

```
mcese>> (mcese:show-alarm x)
nil
mcese>> (mcese:set-alarm x 'foo)

*** mcese:set-alarm: symbol >>foo<< not bound to a function
nil
mcese>> (defun foo ())
foo
mcese>> (mcese:set-alarm x 'foo)
t
mcese>> (mcese:show-alarm x)
foo
mcese>> (mcese:set-alarm x)
t
mcese>> (mcese:show-alarm x)
nil
```

The following function mcese:maxinfer-trace evaluates in backward chaining mode the predicate other_diseases. The inference is executed in tracing mode and hence displayed on the screen step by step:

```
mcese>> (mcese:maxinfer-trace x 'other_diseases)

is the child's breathing noisy, or does he breath through mouth? .3
*** infer: level 0 predicate >>noisy_and_mouth_breathing<< evaluated to .3

does the child have sore throat? 1
*** infer: level 0 predicate >>pain_in_throat<< evaluated to 1

does the child have a fever? 1
*** infer: level 0 predicate >>fever<< evaluated to 1

does the child has troubles to swallow? .7
*** infer: level 0 predicate >>hardship_to_swallow<< evaluated to .7
*** infer: value of predicate >>pain_in_throat<< used as has been
*** infer: value of predicate >>hardship_to_swallow<< used as has been
*** infer: value of predicate >>noisy_and_mouth_breathing<< used as has been

does the child have a headache? 1
*** infer: level 0 predicate >>headache<< evaluated to 1
*** infer: value of predicate >>fever<< used as has been

does the child have a cough? .7
*** infer: level 0 predicate >>cough<< evaluated to .7
*** infer: predicate >>sore_throat<< evaluated via rule >>med1<< to
           min = .81714, max = .81714
*** infer: predicate >>sore_throat<< evaluation:
           min = .81714 via rule >>med1<<
           max = .81714 via rule >>med1<<
*** infer: value of predicate >>fever<< used as has been
*** infer: value of predicate >>headache<< used as has been
*** infer: value of predicate >>pain_in_throat<< used as has been
*** infer: value of predicate >>hardship_to_swallow<< used as has been

does the child have spots in his throat? .5
*** infer: level 0 predicate >>spotted_throat<< evaluated to .5

has the child vomited? 0
*** infer: level 0 predicate >>vomiting<< evaluated to 0

does the child have a rash? 0
*** infer: level 0 predicate >>rash<< evaluated to 0
*** infer: predicate >>strep_throat_wrash<< evaluated via rule >>med2<< to
-- more --
           min = .65102, max = .65102
*** infer: predicate >>strep_throat_wrash<< evaluation:
           min = .65102 via rule >>med2<<
           max = .65102 via rule >>med2<<
*** infer: value of predicate >>fever<< used as has been
*** infer: value of predicate >>headache<< used as has been
*** infer: value of predicate >>pain_in_throat<< used as has been
*** infer: value of predicate >>hardship_to_swallow<< used as has been
*** infer: value of predicate >>spotted_throat<< used as has been
*** infer: value of predicate >>vomiting<< used as has been

does the child have an abdominal pain? .2
*** infer: level 0 predicate >>abdominal_pain<< evaluated to .2
*** infer: value of predicate >>rash<< used as has been
*** infer: predicate >>strep_throat_worash<< evaluated via rule >>med3<< to
           min = .81, max = .81
*** infer: predicate >>strep_throat_worash<< evaluation:
           min = .81 via rule >>med3<<
           max = .81 via rule >>med3<<
```

```
*** infer: predicate >>tonsillitis<< evaluated via rule >>med4<< to
           min = .52674, max = .52674
*** infer: predicate >>tonsillitis<< evaluation:
           min = .52674 via rule >>med4<<
           max = .52674 via rule >>med4<<
*** infer: value of predicate >>sore_throat<< used as has been
-- more --
*** infer: value of predicate >>strep_throat_wrash<< used as has been
*** infer: value of predicate >>strep_throat_worash<< used as has been
*** infer: predicate >>strep_throat<< evaluated via rule >>med5<< to
           min = .81, max = .81
*** infer: predicate >>strep_throat<< evaluation:
           min = .81 via rule >>med5<<
           max = .81 via rule >>med5<<
*** infer: predicate >>throat_trouble<< evaluated via rule >>med6<< to
           min = 1, max = 1
*** infer: predicate >>throat_trouble<< evaluation:
           min = 1 via rule >>med6<<
           max = 1 via rule >>med6<<
*** infer: predicate >>other_diseases<< evaluated via rule >>med7<< to
           min = 0, max = 0
*** infer: value of predicate >>vomiting<< used as has been
*** infer: value of predicate >>abdominal_pain<< used as has been
*** infer: value of predicate >>headache<< used as has been
*** infer: predicate >>other_diseases<< evaluated via rule >>med8<< to
           min = .4, max = .4
*** infer: predicate >>other_diseases<< evaluation:
           min = 0 via rule >>med7<<
           max = .4 via rule >>med8<<
0.4
```

The following function mcese:explain displays the information how the max (or min) evaluation of the given predicate was obtained during the last inference cycle.

```
mcese>> (mcese:explain 'sore_throat 'max)

max value .81714 for predicate >>sore_throat<< was obtained via rule >>med1<<


rule  med1:
      pain_in_throat  &
      .9 * hardship_to_swallow  &
      .4 * noisy_and_mouth_breathing  &
      .6 * headache  &
      .3 * fever  &
      .3 * cough
==>
      sore_throat

   max value 1 of LHS predicate >>pain_in_throat<< used
   1 is the value of the corresponding LHS term
   max value .7 of LHS predicate >>hardship_to_swallow<< used
   .63 is the value of the corresponding LHS term
   max value .3 of LHS predicate >>noisy_and_mouth_breathing<< used
   .12 is the value of the corresponding LHS term
   max value 1 of LHS predicate >>headache<< used
   .6 is the value of the corresponding LHS term
   max value 1 of LHS predicate >>fever<< used
-- more --
   .3 is the value of the corresponding LHS term
   max value .7 of LHS predicate >>cough<< used
   .21 is the value of the corresponding LHS term
certainty value of LHS
```

```
(as the weighted cumulative evidence of the LHS values) is .81714
RHS predicate >>sore_throat<< was evaluated to .81714

nil
```

The following function mcese:eval returns (sorted) list of certainty values obtained in the last inference cycle for the specified list of predicates:

```
mcese>> (mcese:eval '(sore_throat strep_throat tonsillitis) 'max)


((sore_throat 0.81714) (strep_throat 0.81) (tonsillitis 0.52674))
```

The on-line help is facilitated by the function mcese:help, which allows to brows through the help file and/or copy the whole help file to the users home directory:

```
mcese>> (mcese:help)

*** mcese:help  (1) enter 'v' to view the help file
                (2) enter 'c' to copy the help file to your directory
                (3) enter 'b' to view and copy
                (4) enter 'q' to quit
q
t
```

The following functions mcese:close closes the specified knowledge base, unloading its compiled RSET from the lisp system. The functions from FSET remains in the system. They could be removed from oblist one by one, but it is a rather slow process, so we opted for leaving them there. The future use of McESE-FranzLISP will decide whether we have to add the removal of FSET to the capabilities of mcese:close:

```
mcese>> (mcese:close x)
t
mcese>> x
f__KBID0
mcese>> (mcese:kbidp x)
nil
mcese>> (mcese:last-open)
nil
```

To leave the McESE system, just use lisp function (exit):

```
mcese>> (exit)
maccs 2 >
```

(5) CONCLUSION.

The McESE-FranzLISP is the FranzLISP extension by the McESE system. It provides almost full range of McESE capabilities as described in [FB], [FB1] (the forward chaining and the editor for simultaneous editing of a rule and its cvpf are not implemented yet, but will be very soon). Moreover, McESE-FranzLISP provides three built-in cvptf; the weighted cumulative evidence (which returns the sum of all certainty values of the LHS divided by the sum of all weights (see [FB], [FB1]), the max (which returns the maximal of all certainty values of the LHS terms), and the min (which returns the minimum of all certainty values of the LHS terms). The system is being currently used in the fourth year CS course at McMaster University Architecture of Expert Systems, and its graduate version. The system satisfies its goals well and can be a valuable addition to the repertoire of knowledge engineers, mainly for the ability to make fast prototypes in McESE-FranzLISP with a low overhead costs before the expert system is built in a faster and more compact language (as in McESE-C) utilizing the same McESE knowledge bases as the prototype in McESE-FranzLISP.

APPENDIX

Index of McESE-FanzLISP functions

```
(mcese:close '_kbid)
(mcese:comp '_rset ['_comp] ['_intrace] ['_errorec])
(mcese:discomp '_rset ['_out])
(mcese:display '_kb ['_out])
(mcese:eval ['_kbid] '_predlist '_type)
(mcese:explain ['_kbid] '_pred '_type)
(mcese:help)
(mcese:kbidp '_kbid)
(mcese:last-load)
(mcese:last-open)
(mcese:load '_rset)
(mcese:maxinfer ['_kbid] '_pred '_object1 .... '_objectn)
(mcese:maxinfer-trace ['_kbid] '_pred '_object1 .... '_objectn)
(mcese:mininfer ['_kbid] '_pred '_object1 .... '_objectn)
(mcese:mininfer-trace ['_kbid] '_pred '_object1 .... '_objectn)
(mcese:open '_rset ['_fset])
(mcese:rsetidp '_rsetid)
(mcese:show-alarm '_kbid)
(mcese:show-inc '_kbid)
(mcese:show-inconsistency '_kbid)
(mcese:set-alarm '_kbid '_func)
(mcese:set-inc'_kbid '_inc)
(mcese:set-inconsistency '_kbid '_inc)
(mcese:sufmaxinfer ['_kbid] '_cutoff '_pred '_object1 .... '_objectn)
(mcese:sufmaxinfer-trace ['_kbid] '_cutoff '_pred '_object1 .... '_objectn)
(mcese:sufmininfer ['_kbid] '_cutoff '_pred '_object1 .... '_objectn)
(mcese:sufmininfer-trace ['_kbid] '_cutoff '_pred '_object1 .... '_objectn)
(readf)
(readi)
(reads)
(mcese:unload '_rsetid)
```

Description of individual functions

(mcese:close '_kbid)

returns: nil and displays error messages if errors have occurred, otherwise t is returned.

side effects: none.

action: the argument must evaluate to a valid kbid. The knowledge base _kbid is closed (i.e. _kbid is not a valid kbid any more), its rule set is "unloaded" from memory. The knowledge base cannot be used again unless opened anew.

(mcese:comp '_rset ['_comp] ['_intrace] ['_errorec])

returns: nil and displays error messages if errors have occurred. Otherwise t is returned.

side effects: none.

action: the first argument (supplied for _rset) must evaluate to a name. The
second argument must evaluate to a name (and it is supplied for _comp), or the
second argument must evaluate to either 0 or 1 (and then it is supplied for
_intrace). The third argument must evaluate to either 0 or 1; if the second
argument was _comp, the third argument is supplied for _intrace; if the second
argument was _intrace, the third argument is supplied for _errorec. The fourth
argument must evaluate to either 0 or 1 and is supplied for _errorec.
_rset is the name of a source rule set file.
If specified, _comp is the name of a file the compiled _rset should be put in.
If specified, _intrace is the input trace indicator. If 1, mcese:comp
will display the input from _rset on the screen as the parsing and compiling
of _rset progresses, if 0, the display of the input from _rset is suppressed.
Default value is 0.
If specified, _errerec is the error recovery indicator. If 1, mcese:somp
will continue compiling even after an error was found, if 0, after the first
error mcese:comp stops. Default value is 1.

(mcese:discomp '_rset ['_out])

returns: returns nil and displays error messages if errors have occurred;
if successful mcese:discomp returns t.

side effects: none.

action: the value of the first argument must be a valid rsetid . The
value of the second (optional) argument must be a name of a file. If only
one argument is specified, mcese:discomp prints on the screen the
contents and statistics of _rset. In case the second argument is specified,
the contents and the statistics of _rset is printed into the output file
specified by the value of the second argument.

(mcese:display '_kb ['_out])

returns: returns nil and displays error messages if errors have occurred;
if successful mcese:display returns t.

side effects: none.

action: the value of the first argument must be a valid kbid. The value
of the second (optional) argument must be a name of a file. If only one
argument is specified, mcese:display prints on the screen the contents
and statistics of the knowledge base _kbid. In case the second argument
is specified, the contents and the statistics of the knowledge base _kbid
is printed into the output file specified by the value of the second
argument.

(mcese:eval ['_kbid] '_predlist '_type)

returns: nil and displays error messages if errors have occurred. A sorted
list of predicates and their values if successful.

side effects: none.

action: the first argument must evaluate to a valid kbid (and then it is
supplied for _kbid), or to a list (and then the last opened knowledge base

is supplied for _kbid and the first argument is supplied for _predlist). If the first argument was _kbid, the second argument must evaluate to a list and is supplied for _predlist. The last argument must evaluate either to max or min , and is supplied for _type. The _predlist must be a list of predicate names. The list of lists (_predicate _value) {where _value is the max (or min respectively) value of _predicate as inferred during the last inference with the knowledge base _kbid} is then returned, sorted according the values (in a descending order).

(mcese:explain ['_kbid] '_pred '_type)

returns: nil and displays error messages if errors have occurred, otherwise t is returned.

side effects: none.

action: if only two arguments are specified, it is assumed that the last opened knowledge base is used (and supplied for _kbid); the first argument (supplied for _pred) must then evaluate to a name (symbol), the second argument (supplied for _type) must evaluate to max or min. If three arguments are specified, the first one (supplied for _kbid) must evaluate to a valid kbid, the second (supplied for _pred) to a symbol, the third (supplied for _type) to max or min. The explanation how the max (or min respectively) value of the predicate _pred was obtained during the last inference with the knowledge base _kbid is then displayed on the screen.

(mcese:help)

returns: nil and displays error messages if errors have occurred, otherwise t is returned.

side effects: none.

action: a menu of options is displayed, and the help file is either viewed, copied, or both.

(mcese:kbidp '_kbid)

returns: t if the value of the argument is a valid kbid, otherwise nil is returned.

side effects: none.

action: none.

(mcese:last-load)

returns: rsetid of the most recent rule set loaded, or nil (if none loaded).

side effects: none.

action: none.

(mcese:last-open)

returns: kbid of the most recent knowledge base opened, or nil (if none opened).

side effects: none.

action: none.

(mcese:load '_rset)

returns: returns nil and displays error messages if errors have occurred; if successful mcese:load returns an rsetid.

side effects: none.

action: mcese:load opens a file whose name is the value of the argument, it is assumed to contain a compiled mcese rule set (as produced by mcese:comp). This compiled rule set is then loaded into memory and a valid rsetid identifying it is returned.

(mcese:maxinfer ['_kbid] '_pred '_object1 .... '_objectn)

like mcese:maxinfer-trace without tracing.

(mcese:maxinfer-trace ['_kbid] '_pred '_object1 .... '_objectn)

returns: nil and displays error messages if errors have occurred. Otherwise a flonum between 0 and 1 (inclusive) or -1.0 is returned.

side effects: none.

action: the first argument must evaluate either to a valid kbid (and then it is supplied for _kbid), or to a name (and the it is supplied for _pred). In case the first argument is _pred, the last opened knowledge base is supplied for _kbid. Arguments following _pred are assumed to be lisp objects to be bound to variables of the predicate _pred. Backward chaining evaluation of the predicate _pred with given bindings is performed in maximum mode. The inference is traced on the screen. The result of the inference is returned. If ALARM is detected, the corresponding alarm function is invoked. If -1.0 is returned, the predicate _pred could not be evaluated.

(mcese:mininfer ['_kbid] '_pred '_object1 .... '_objectn)

like mcese:maxinfer but in minimum mode.

(mcese:mininfer-trace ['_kbid] '_pred '_object1 .... '_objectn)

like mcese:maxinfer-trace but in minimum mode.

(mcese:open '_rset ['_fset])

returns: nil and displays error messages if errors have occurred; returns a valid rsetid and displays error messages if _rset was loaded successfully but some other errors occurred; returns a valid kbid if both, _rset and _fset were loaded successfully and the knowledge base was open successfully.

side effects: none.

action: the value of the first argument must be either a valid rsetid, or a name of a file containing a compiled rule set. In the latter case, the compiled rule set is loaded into memory and a valid rsetid is issued for it. If the second argument is specified, the file _fset is loaded into memory (it should contain lisp functions - CVPF's for _rset). To complete open successfully, all addresses of level 0 predicate functions, and all CVPF's must be found. If yes, a valid kbid is issued and returned and the knowledge base is ready for use. If not all level 0 predicate functions or CVPF's are found, and error message is displayed and only the valid rsetid is returned.

(mcese:rsetidp '_rsetid)

returns: t if the value of the argument is a valid rsetid, otherwise it returns nil.

side effects: none.

action: none.

(mcese:show-alarm '_kbid)

returns: nil and displays error messages if errors have occurred, otherwise the name of the alarm function for _kbid is returned.

side effects: none.

action: the argument must evaluate to a valid kbid.

(mcese:show-inc '_kbid) (mcese:show-inconsistency '_kbid)

returns: nil and displays error messages if errors have occurred, otherwise the inconsistency level tolerance for _kbid is returned.

side effects: none.

action: the argument must evaluate to a valid kbid.

(mcese:set-alarm '_kbid '_func)

returns: returns nil and displays error messages if errors have occurred, otherwise t is returned.

side effects: none.

action: the argument must evaluate to a valid kbid. Name of alarm function for the specified knowledge base _kbid is set set to the value of the second argument (must evaluate to a name of an existing lisp function).

(mcese:set-inc '_kbid '_inc) (mcese:set-inconsistency '_kbid '_inc)

returns: returns nil and displays error messages if errors have occurred, otherwise t is returned.

side effects: none.

action: the argument must evaluate to a valid kbid. The inconsistency level tolerance for the specified knowledge base _kbid is set to the value of the second argument (must evaluate to a number between 0 and 1 inclusive).

(mcese:sufmaxinfer ['_kbid] '_cutoff '_pred '_object1 .... '_objectn)

like mcese:sufmaxinfer-trace but without tracing.

(mcese:sufmaxinfer-trace ['_kbid] '_cutoff '_pred '_object1 .... '_objectn)

returns: nil and displays error messages if errors have occurred. Otherwise a flonum between 0 and 1 (inclusive) or -1.0 is returned.

side effects: none.

action: the first argument must evaluate either to a valid kbid (and then it is supplied for _kbid), or to a number between 0 and 1 inclusive (and then it is supplied for _cutoff). In case the first argument is _cutoff, the last opened knowledge base is supplied for _kbid. The argument following _cutoff must evaluate to a name and is supplied for _pred. Arguments following _pred are assumed to be lisp objects to be bound to variables of the predicate _pred. Backward chaining evaluation of the predicate _pred with given bindings is performed in sufficient maximum mode with the cutoff value _cutoff. The inference is traced on the screen. The result of the inference is returned. If ALARM is detected, the corresponding alarm function is invoked. If -1.0 is returned, the predicate _pred could not be evaluated.

(mcese:sufmininfer ['_kbid] '_cutoff '_pred '_object1 .... '_objectn)

like mcese:sufmaxinfer but in sufficient minimum mode.

(mcese:sufmininfer-trace ['_kbid] '_cutoff '_pred '_object1 .... '_objectn)

like mcese:sufmaxinfer-trace but in sufficient minimum mode.

(readf)

returns: nil or a flonum.

side effects: none.

action: a flonum is read from the keyboard and returned. If error, nil is returned.

(readi)

returns: nil or a fixnum.

side effects: none.

action: a fixnum is read from the keyboard and returned. If error, nil is returned.

(reads)

returns: nil or a string.

side effects: none.

action: a string is read from the keyboard and returned. If error, nil is returned.

(mcese:unload '_rsetid)

returns: nil and displays error messages if errors have occurred, otherwise t is returned.

side effects: none.

action: the argument must evaluate to a valid rsetid. The compiled rule set _rsetid is "unloaded" from memory, _rsetid is not a valid rsetid any more. The opened knowledge base (if any) containing this rule set is closed.

REFERENCES

[FB] F. Franek, I. Bruha, McESE - NcMaster Expert System Environment, submitted for publication.

[FB1] F. Franek, I. Bruha, The McESE project, Tech. Rep., Dept. of Comp. Sci. & Systems, McMaster University, Hamilton, Ont., Canada, 1988.