F.G.H.
2/28/85

LispCourse #11: Error Processing & DWIM

Completions and Corrections

1. KEYACTION and the "A"

   I very stupidly used the "A" key in many of my examples of KEYACTION last time. Don’t try out my examples on the "A" key. Use the "B" key. The reason is that if you alter the "A" to produce, say, a "C", then you can never type in "KEYACTION" to change the "A" key back since "KEYACTION" will come out "KEYCCTION". Beware!

2. The variable *IT* in the P.A.

   The P.A. maintains a variable called *IT* that is always bound to the value of the last expression. You can use IT in the current expression to refer to the value returned by the last expression.
   For example:

```
1_ (CAR ’((A B C)(D E F))
(A B C)
2_ IT
(A B C)
3_ (CDR IT)
(B C)
4_ (CDR IT)
(C)
```

   A more realistic example:

```
5_ (COPYFILE ’{DSK}ABC  ’{DSK}NEW)
{DSK}NEW;1
6_  (TEDIT IT)
{PROCESS}#1,1232
```

3. The % quote in the Lisp READER

   The Lisp READER uses special characters to delimit atoms and lists. In particular, the "(" and ")" characters delimit lists and spaces delimit atoms. Ordinarily *(ABC)* would be considered an list containing one atom. *(ABC DEF)* would be a list containing two atoms.

A "%" preceding any special character can be used to surpress the special interpretation of that character. When preceded by a "%", special characters are considered ordinary characters.

For example, *%(ABC%)* would be considered a single atom and not a list because the special meaning of the ")" and "(" characters are supressed by the preceding "%".

Similarly, *(ABC% DEF)* is a list containing a SINGLE atom because the "%" surpresses the delimiting function of the space between "ABC" and "DEF".

The "%" should be thought of as a QUOTE that works on characters rather than Lisp S-expressions.

Since "%" is itself a special character, it has to be quoted if you want to use it in an atom. For example, *(A %% B)* is a list containing three atoms, the middle of which is the atom %.

An Overview of Error Processing

The diagram in the Appendix illustrates what happens when an error occurs in the Lisp evaluator.

Errors occur when the Evaluator tries to evaluate an expression containing an unbound atom (i.e., an atom with no value), an undefined function (i.e., a form whose CAR is not the name of a function), an illegal argument to a functions, etc.

The major steps in the processing of errors are the following:

DWIM (and CLISP) ž unbound atom and undefined function errors are passed to DWIM, which attempts to automatically "correct" the error. DWIM assumes that the error is either a CLISP expression or a typo.

CLISP is a special syntax for Lisp that is "easier to use" (e.g., CLISP allows infix notation). If the "error" was actually a CLISP expression, then DWIM translates the expression into standard Lisp and returns the translated expression to the Evaluator. (Note: CLISP is not an error, just a special syntax that is implemented using the error mechanism. A very poor piece of systems design!!!!)

If the error appears to be a typo, DWIM "corrects" it and returns the corrected expression to the Lisp evaluator.

If it appears that the error is neither a typo nor a CLISP expression, then DWIM just passes on to the Error Mechanism.

Error Mechanism ž *u.b.a.* and *u.d.f.* errors not corrected by DWIM and all other errors are passed to the Error mechanism.

Each error is assigned an error number that identifies what kind of error it is. There are about 50 such error numbers.

The Error Mechanism, then determines whether to enter a Break. This determination depends on a number of factors including how many function have already been called and how long the computation has been going on.

If the error mechanism decides not to enter a Break, it prints an error message on the TTY window and then aborts the evaluation.

Breaks ž when appropriate, the Error Mechanism passes control to the Break Handler.

The Break Handler enters a "break", usually by opening a Break Window.

The Break Handler prints an error message and then passes control to the Break Exec.

In the Break Exec, the user can evaluate any expression just as in the Lisp Exec. Special commands for inspecting the state of the evaluation causing the error are also available.

To end the Break, the user can either repair the error and then return to the Lisp evaluator OR abort the evaluation in progress and return to the Lisp Exec.

DWIM (Automatic Error Correction)

Introduction

DWIM stands for "Do What I Mean".

DWIM contains two major components: an automatic typo corrector and a special set of Lisp-like syntax called CLISP. CLISP will be talked about in a later section. Only the automatic error corrector will be described here.

The DWIM error correction is a valiant attempt at making a user interface that is robust to simple user errors. It looks at an u.b.a. or a u.d.f. AND at the current context and tries to figure out what the user actually meant.

DWIM embodies an implicit model of the user and the types of errors he is likely to make. DWIM also embodies lots of assumptions about the way the Interlisp system works. Unfortunately, neither DWIM's model of the user nor DWIM's assumptions about the Interlisp environment are very good.

You can't turn DWIM off. Too much of the system has been made dependent on DWIM.   So you have to learn to deal with DWIM and its idiosyncracies.

Spelling Correction

Basic idea:

The major function of the DWIM error corrector is spelling correction. When given a u.b.a or u.d.f. error, DWIM compares the incorrect atom (i.e., the unbound atom or the CAR of the u.d.f. form) against a list of atoms that it knows about. If one of the atoms on the list is "close" to the error atom, DWIM replaces the error atom by the "correct" atom.

DWIM uses a number of heuritics to determine the "closeness" of two atoms Basically, closeness decreases with the number of letters that are different between the two atoms. It increases as a function of the number of letters in the longer atom.

Examples:

1. "CONX" is closer to "COND" than to "CORE"
2. "PRETTYPRNT " is closer to "PRETTYPRINT"
   than "EQX" is to EQP"

DWIM ignores single transpositions (e.g., SD for DS) and doubled letters (e.g., SS for S). DWIM considers "CONS" and "CONNS" to be maximally close (i.e., identical) because they differ only by a doubled letter.   Also "CONS" and CNOS" are maximally close since they differ by a transposition.

DWIM's spelling corrector works as follows:

> It compares the error atom to each atom on its lists of atoms and if it can it chooses one.
>
> The choice rules are:
>
> > If one matches with maximum closeness, then it chooses that atom. For example, if the error atom is CONSS and CONS is on the known atoms list.
> >
> > At the end of the list, if DWIM will choose the **one** atom that had the maximum closeness measure, providing its closeness measure is over some threshhold.
> >
> > If there is no **unique** atom with a maximum closeness measure, DWIM won't choose any atom on the list. This can happen if there are no close atoms are if the maximum closeness is shared by two or more atoms on the list.

Lists of known atoms:

DWIM uses different lists of known atoms for different kinds of errors. Basically, it keeps a list of known function names for undefined function errors and it keeps a list of known variables for unbound atom errors.

These lists are automatically maintained by DWIM and by the Programmer's Assistant.

Some atoms are permenantly on these spelling lists. Others are temporary.

The temporary atoms are placed on the list when they are used in an expression in which uses one of DEFINEQ, SETQ, COND, DF, ...

If a temporary atom gets used to correct an error, then it becomes permanent. Otherwise, it falls off the list after 30 or so new temporary atoms are added to the list.

Basically, the list of known atoms corresponds to the last 30 or so atoms or function names that you referred to.

Note that these rules hold basically for user type-in.

DWIM also operates on errors occurring within a function. However, if an error occurs within a function, DWIM acts slightly differently. In particular, it uses the other atoms within the function definition to determine how to spelling correct an unbound atom.

Other DWIM corrections

DWIM makes a few other corrections besides simple spelling corrections. These corrections include the following:

’ followed by a space and an S-expression ž This would reuslt in an u.b.a. error on the ’. DWIM will "correct" this expression to eliminate the space after the ’. For example: (CONS ’A ’ (A B C)) would be corrected to (CONS ’A ’(A B C)) by DWIM. Similarly, (LIST ’ A ’ B) would be corrected to (LIST ’A ’B).

Misplaced T clause in COND expression ž DWIM will attempt to correct various misplacements of a T clause in a COND expression. It uses various heuristics to do so. For example:

```
(COND ((NUMBERP ’A) ’X)((T ’Y))) would get corrected to
        (COND ((NUMBERP ’A) ’X)(T ’Y))
(COND ((NUMBERP ’A) ’X))(T ’Y) would get corrected to
        (COND ((NUMBERP ’A) ’X)(T ’Y))
```

And other fairly esoteric stuff .... ž See the DWIM documentation for all the Bells and whistles.

The DWIM user interface

DWIM is called automatically when an appropriate error occurs in the Lisp evaluator. However, DWIM sometimes interacts with the user.

In particular, DWIM has two modes: ***CAUTIOUS*** and ***TRUSTING***.

In **TRUSTING** mode, DWIM makes most corrections without asking the user’s permission. It simply prints a record of the corrections it is making in the TTY window.

Example:

```
5_ (COSN 4 (LIST 5))
```

```
=CONS
(4 5)
```

The "=CONS’ line is a message from DWIM saying that it did a spelling correction that resulted in an atom being corrected to CONS.

```
6_ (DEFINEQ (JUNK (LAMBDA NIL (ITIMSE 4 3))))
(JUNK)
7_ (JUNK)
ITIMSE {in JUNK} -> ITIMES
12
```

The "ITIMSE {in JUNK} -> ITIMES" line is a message from DWIM saying that while executing JUNK it changed "ITIMSE" to "ITIMES"

In **CAUTIOUS** mode, DWIM makes asks the user’s permission before making most corrections.

DWIM will print out the correction message followed by a "?".

If the users types "Y", then DWIM will make the correction.

If the user types "N", then DWIM will not mnake the correction and return to the Error Mechanism.

If the user types a space or <RETURN>, then DWIM will just wait until the user types a "Y" or "N".

If the user types nothing, DWIM will wait 10 seconds and then assume a default answer. The default anser may be "Y" or "N" depending on the type of correction being done.

Example:

```
6_ (DEFINEQ (JUNK (LAMBDA NIL (ITIMSE 4 3))))
(JUNK)
7_ (JUNK)
ITIMSE {in JUNK} ->
```

At this point DWIM waits for an answer of Y or N or for 10 seconds. In this case, the default answer after 10 seconds would be "Y".

When processing type-in DWIM is always in TRUSTING mode. The assumption is that you can always undo what DWIM has done.

When processing a function, DWIM can be in either CAUTIOUS or TRUSTING mode. The default is TRUSTING.

To change DWIM modes, use the function **DWIM**. **DWIM** takes one argument, *Mode*.

If *Mode* is the atom C, then DWIM is set to CAUTIOUS mode.

If *Mode* is the atom T, then DWIM is set to TRUSTING mode.

If *Mode* is NIL, then DWIM is turned off.

Beware: Lots of stuff will stop working when DWIM is turned off. It is probably a very bad idea to turn DWIM off, since you can never tell what depends on DWIM to work correctly!!!

Some DWIM parameters

DWIMWAIT ž the number of seconds DWIM will wait for a response from the user before assuming the default answer. Initially, 10 seconds.

FIXSPELLDEFAULT ž the default answer thatr DWIM uses after DWIMWAIT seconds is up during a spelling correction in CAUTIOUS mode. Initially, y.

FIXSPELLREL ž the minimum closeness measure needed to accept a known atom as a correction for an error atom. If 100, then only maximum matches will be accepted. Note that maximum matches include atoms that differ by single transpositions and doublings. Initially set to 70.

ADDSPELLFLG ž If NIL, suppresses the addition of items to the known atoms lists. If T, enables the additions. Initially, T.

NOSPELLFLG ž If T suppresses *all* spelling correction in DWIM. If other non-NIL value, suppresses spelling corrections in functions but not in type-in. If NIL, spelling correction is done. Initially, NIL.

RUNONFLG ž If T, DWIM tries to correct run-on typos. E.g., (IPLUS 8A) might be corrected to (IPLUS 8 A). If NIL, no run-on correction is done. Initially, NIL because run-on corrections are not very good.

SPELLINGS1, SPELLINGS2, SPELLINGS3, USERWORDS ž these are the lists of known atoms that DWIM and the P.A. maintain. You probably can’t do much except look at these.

#SPELLINGS1, #SPELLINGS2, #SPELLINGS3, #USERWORDS ž these parameters have integer values that determine the length of the known atoms lists. If you want to increase the "history" of known atoms, increase the values of the parameters. Initially, 20.

DWIM Documentation

DWIM is documented in Chapter 15 of the IRM.   The chapter appears to be significantly out-of-date!. It is very detailed and is probably hard to understand for the average user.

The introduction to Chapter 15 and Section 15.1 are good overall descriptions.

Sections 15.4 and 15.6 are more detailed descriptions that may be of interest to the non-programmer.

Caution is the bottom line with DWIM

DWIM doesn't work very well as a user interface. It is very non-intuitive and causes most users many problems than it solves. Tread with caution when interacting with DWIM.

Examples: [This is a transcript of a short session with DWIM]

```
6_ (CNOS 4 5)
UNDEFINED CAR OF FORM
CNOS
7_ (CONS 4 5)
(4 . 5)
8_ (CNOS 4 5)
=CONS
(4 . 5)
9_ (CONNS 4 5)
=CONN
{PHYLUM}<4>
10_ UNDO
CONN undone.
11_ (COSN 4 5)
=CONS
(4 . 5)
12_ (CON 4 5)
```

```
=CONN
{PHYLUM}<4>
13_ (CONDD (T ’X))
=CONN
ILLEGAL ARG
(T (QUOTE X))
14_ (COND (T ’X))
X
15_ (CONDD (T ’X))
=CONN
ILLEGAL ARG
(T (QUOTE X))
16_ (CODN (T ’X))
=COND
X
17_ (CONX 4 5)
=CONN
{PHYLUM}<4>
```