UTIL now works. There are still problems. They are gradually being fixed.

Command: util

?- halt.

?- em <file>

?- em.

?- redo <file>.

?- redo.

NB To interrupt type: \[Esc\]

Int: a

prompt

This will do an abort.

Problems
1) READIN stuff not currently loaded. (used in NL files?)

2) Listing not available. (Use clause (Header,Body) in emergencies)

3) == isn't working in UTIL.

This means that subst and occ don't work. (You use these don't you?)

4) If you type control characters at it through editor, you will blow it up.

5) When Prolog blows up it prints 2 pages of horrific diagnostics!

These problems will be fixed. (Wait for messages to this effect).
ECMIQ7 UTIL
- UTILOPS
- WRITEE
extract from ERCLIB.LIBCONTENTS

2.33 LISTPDMEMS (W.Watson, Mol. Bi

Command: LISTPDMEMS(pdfile, control)

Lists selected members, specified by 'control', of an existing partitioned file 'pdfile' to the console or a line-printer.

'control' can be a file containing the members concerned, one per line, ending with .END. If the 'control' parameter is omitted this input is taken from the console. If 'control' is itself a member of pdfile' its name may be abbreviated to _membername. In each of these cases a prompt will be issued for the output device on which the member listings are to appear.

Otherwise 'control' should specify the output device required and the list of member names will be requested from the console.
Command: PACKPD(pdfile, control, option)

Copies selected disk files, specified by ‘control’, to members (with the same names) of a partitioned file ‘pdfile’, which will be created if it does not already exist, optionally destroying the disk files afterwards.

‘control’ is a file containing the members concerned, one per line, ending with .END. If the ‘control’ parameter is omitted this input is taken from the console. If ‘control’ is itself a member of ‘pdfile’, which must already exist in this case, the parameter may be abbreviated to _membername.

‘option’ can be one of three letters (upper or lower case)
K - keep disk files after copying in
D - delete disk files after copying in
T - delete members without copying, i.e. tidy ‘pdfile’
Command: UNPACKPD(pdfile, control, option)

Copies selected members, specified by 'control', of an existing partitioned file 'pdfile' to disk files of the same names as the members, optionally destroying the members afterwards.

'control' is a file containing the members concerned, one per line, ending with .END. If the 'control' parameter is omitted this input is taken from the console. If 'control' is itself a member of 'pdfile' the parameter may be abbreviated to _membername.

'option' can be one of three letters (upper or lower case)

K - keep members after copying out
D - delete members after copying out
T - delete members without copying, i.e. tidy 'pdfile'
<table>
<thead>
<tr>
<th>Group</th>
<th>Command</th>
<th>Purpose</th>
<th>Output</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>General File Utilities</td>
<td>ACCEPT</td>
<td>Transfer file OFFERed by another user</td>
<td></td>
<td>5-4</td>
</tr>
<tr>
<td></td>
<td>ARCHIVE</td>
<td>Mark file(s) for transfer to archive store</td>
<td></td>
<td>5-6</td>
</tr>
<tr>
<td></td>
<td>CHERISH</td>
<td>Mark file(s) for backing up</td>
<td></td>
<td>5-6</td>
</tr>
<tr>
<td></td>
<td>DESTROY</td>
<td>Destroy file(s) in disc store</td>
<td></td>
<td>5-2</td>
</tr>
<tr>
<td></td>
<td>DISCARD</td>
<td>Destroy file(s) in archive store</td>
<td></td>
<td>5-7</td>
</tr>
<tr>
<td></td>
<td>DISCONNECT</td>
<td>Remove file from virtual memory</td>
<td></td>
<td>5-3</td>
</tr>
<tr>
<td></td>
<td>FILES</td>
<td>Obtain complete or partial list of files in disc and archive stores</td>
<td></td>
<td>5-1</td>
</tr>
<tr>
<td></td>
<td>HAZARD</td>
<td>Remove CHERISH marker(s) for file(s)</td>
<td></td>
<td>5-6</td>
</tr>
<tr>
<td></td>
<td>OFFER</td>
<td>Mark file for transfer to another user</td>
<td></td>
<td>5-4</td>
</tr>
<tr>
<td></td>
<td>PERMIT</td>
<td>Allow other users access to a file</td>
<td></td>
<td>5-4</td>
</tr>
<tr>
<td></td>
<td>RENAME</td>
<td>Change the name of a file</td>
<td></td>
<td>5-2</td>
</tr>
<tr>
<td></td>
<td>RESTORE</td>
<td>Copy a file from archive to disc store</td>
<td></td>
<td>5-7</td>
</tr>
<tr>
<td>Type Specific</td>
<td>ANALYSE</td>
<td>Obtain details of type, contents, access permission, etc. of a file</td>
<td></td>
<td>6-1</td>
</tr>
<tr>
<td>File Utilities</td>
<td>CONCAT</td>
<td>Join two or more character files</td>
<td></td>
<td>6-3</td>
</tr>
<tr>
<td></td>
<td>CONVERT</td>
<td>Convert a data file to a character file</td>
<td></td>
<td>6-3</td>
</tr>
<tr>
<td></td>
<td>COPY</td>
<td>Copy a file</td>
<td></td>
<td>6-2</td>
</tr>
<tr>
<td></td>
<td>LIST</td>
<td>List file on output device</td>
<td></td>
<td>6-4</td>
</tr>
<tr>
<td></td>
<td>NEWSPDFILE</td>
<td>Create new, empty, partitioned file</td>
<td></td>
<td>4-4</td>
</tr>
<tr>
<td></td>
<td>SEND</td>
<td>List file on output device and destroy it</td>
<td></td>
<td>6-5</td>
</tr>
<tr>
<td>Manipulating Data</td>
<td>CLEAR</td>
<td>Break link set up by DEFINE</td>
<td></td>
<td>7-6</td>
</tr>
<tr>
<td></td>
<td>DEFINE</td>
<td>Set up link between logical channel and particular file or output device, or get list of current links</td>
<td></td>
<td>7-3</td>
</tr>
<tr>
<td></td>
<td>DEFINEMT</td>
<td>Set up link between logical channel and particular magnetic tape file</td>
<td></td>
<td>10-2</td>
</tr>
<tr>
<td></td>
<td>NEWSMFILE</td>
<td>Create new file to be accessed via store mapping facilities</td>
<td></td>
<td>9-1</td>
</tr>
<tr>
<td>File Editing</td>
<td>ECCE</td>
<td>Edit character file</td>
<td></td>
<td>8-10</td>
</tr>
<tr>
<td></td>
<td>EDIT</td>
<td>Edit character file</td>
<td></td>
<td>8-1</td>
</tr>
</tbody>
</table>

Table 4.5: Edinburgh Subsystem Command Summary
(continued on next page)
<table>
<thead>
<tr>
<th>Group</th>
<th>Command</th>
<th>Purpose</th>
<th>Output</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LOOK</td>
<td>Examine contents of character file</td>
<td>*</td>
<td>8-9</td>
</tr>
<tr>
<td></td>
<td>RECALL</td>
<td>Examine file containing record of interactive terminal I/O</td>
<td>*</td>
<td>8-10</td>
</tr>
<tr>
<td></td>
<td>RECAP</td>
<td>Examine file containing record of interactive terminal I/O</td>
<td>*</td>
<td>8-21</td>
</tr>
<tr>
<td></td>
<td>SHOW</td>
<td>Examine contents of character file</td>
<td>*</td>
<td>8-21</td>
</tr>
<tr>
<td>File Editing (continued)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compilers and associated</td>
<td>ALGOL</td>
<td>Compile ALGOL 60 source file</td>
<td>*</td>
<td>11-1</td>
</tr>
<tr>
<td>commands</td>
<td>FORTE</td>
<td>Compile FORTRAN IV source file</td>
<td>*</td>
<td>11-1</td>
</tr>
<tr>
<td></td>
<td>IMP</td>
<td>Compile IMP source file</td>
<td>*</td>
<td>11-1</td>
</tr>
<tr>
<td></td>
<td>LINK</td>
<td>Join two or more object files</td>
<td>*</td>
<td>11-5</td>
</tr>
<tr>
<td></td>
<td>PARM</td>
<td>Set compiler options, or get list of current options</td>
<td>*</td>
<td>11-2</td>
</tr>
<tr>
<td></td>
<td>RUN</td>
<td>Execute program</td>
<td></td>
<td>11-5</td>
</tr>
<tr>
<td>Commands associated</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>with Directories</td>
<td>ALIAS</td>
<td>Give alias name to a specified command, or remove all aliases</td>
<td></td>
<td>11-10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>associated with the command</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>INSERT</td>
<td>Insert details of object file in current active directory</td>
<td></td>
<td>11-9</td>
</tr>
<tr>
<td></td>
<td>INSERTMACRO</td>
<td>Insert details of character file containing a macro in current active directory</td>
<td></td>
<td>16-13</td>
</tr>
<tr>
<td></td>
<td>NEWDIRECTORY</td>
<td>Create a new directory file if default size not adequate</td>
<td>*</td>
<td>11-8</td>
</tr>
<tr>
<td></td>
<td>REMOVE</td>
<td>Remove reference to object file from current active directory</td>
<td></td>
<td>11-10</td>
</tr>
<tr>
<td></td>
<td>REMOVEMACRO</td>
<td>Remove details of character file containing a macro from current active directory</td>
<td></td>
<td>16-3</td>
</tr>
<tr>
<td></td>
<td>TIDYDIR</td>
<td>Tidy directory file</td>
<td></td>
<td>11-9</td>
</tr>
<tr>
<td>Background Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DELETEDOC</td>
<td>Remove job from background job queue</td>
<td>*</td>
<td>16-2</td>
</tr>
<tr>
<td></td>
<td>DETACH</td>
<td>Put job into background job queue</td>
<td>*</td>
<td>16-2</td>
</tr>
<tr>
<td></td>
<td>DETACHJOB</td>
<td>Put job into background job queue</td>
<td>*</td>
<td>16-3</td>
</tr>
<tr>
<td>Commands associated</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>with accounting</td>
<td>METER</td>
<td>Print usage information for current session</td>
<td>*</td>
<td>17-2</td>
</tr>
<tr>
<td></td>
<td>PASSWORD</td>
<td>Change foreground and/or background password</td>
<td>17-1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>USERS</td>
<td>Print number of currently active users</td>
<td>*</td>
<td>17-2</td>
</tr>
</tbody>
</table>

Table 4.5: Edinburgh Subsystem Command Summary
(continued on next page)
<table>
<thead>
<tr>
<th>Group</th>
<th>Command</th>
<th>Purpose</th>
<th>Output</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information and other commands</td>
<td>ALERT</td>
<td>Obtain information on recent changes in the service</td>
<td>*</td>
<td>4-7</td>
</tr>
<tr>
<td></td>
<td>CPULIMIT</td>
<td>Set time limit for each command</td>
<td>*</td>
<td>17-3</td>
</tr>
<tr>
<td></td>
<td>DELIVER</td>
<td>Set text for heading of line printer output, etc., or get current text</td>
<td>*</td>
<td>17-4</td>
</tr>
<tr>
<td></td>
<td>DOCUMENTS</td>
<td>Print information about documents in System queues</td>
<td>*</td>
<td>17-4</td>
</tr>
<tr>
<td></td>
<td>HELP</td>
<td>Get advice on using Subsystem</td>
<td>*</td>
<td>4-7</td>
</tr>
<tr>
<td></td>
<td>MESSAGES</td>
<td>Inhibit or permit messages to the interactive terminal</td>
<td></td>
<td>17-5</td>
</tr>
<tr>
<td></td>
<td>OBEY</td>
<td>Execute a sequence of commands</td>
<td>*</td>
<td>17-5</td>
</tr>
<tr>
<td></td>
<td>OBEYJOB</td>
<td>Execute a sequence of commands</td>
<td>*</td>
<td>16-4</td>
</tr>
<tr>
<td></td>
<td>OPTION</td>
<td>Set Subsystem options, or get list of options in effect</td>
<td>*</td>
<td>17-5</td>
</tr>
<tr>
<td></td>
<td>QUIT</td>
<td>Terminate session</td>
<td>*</td>
<td>17-8</td>
</tr>
<tr>
<td></td>
<td>SETMODE</td>
<td>Set characteristics of interactive terminal</td>
<td></td>
<td>17-9</td>
</tr>
<tr>
<td></td>
<td>STOP</td>
<td>Terminate session</td>
<td>*</td>
<td>17-10</td>
</tr>
<tr>
<td></td>
<td>SUGGESTION</td>
<td>Send suggestion to System Manager</td>
<td></td>
<td>17-11</td>
</tr>
<tr>
<td></td>
<td>TELL</td>
<td>Send message to specified user, immediately or at his next log-on</td>
<td></td>
<td>17-11</td>
</tr>
</tbody>
</table>

Table 4.5: Edinburgh Subsystem Command Summary
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPT</td>
<td>5-4</td>
</tr>
<tr>
<td>ALERT</td>
<td>4-7</td>
</tr>
<tr>
<td>ALGOL</td>
<td>11-1</td>
</tr>
<tr>
<td>ALIAS</td>
<td>11-10</td>
</tr>
<tr>
<td>ANALYSE</td>
<td>6-1</td>
</tr>
<tr>
<td>ARCHIVE</td>
<td>5-6</td>
</tr>
<tr>
<td>CHERISH</td>
<td>5-6</td>
</tr>
<tr>
<td>CLEAR</td>
<td>7-6</td>
</tr>
<tr>
<td>CONCAT</td>
<td>6-3</td>
</tr>
<tr>
<td>CONVERT</td>
<td>6-3</td>
</tr>
<tr>
<td>COPY</td>
<td>6-2</td>
</tr>
<tr>
<td>CPULIMIT</td>
<td>17-3</td>
</tr>
<tr>
<td>DEFINE</td>
<td>7-3</td>
</tr>
<tr>
<td>DEFINEMT</td>
<td>10-2</td>
</tr>
<tr>
<td>DELETEDOC</td>
<td>16-2</td>
</tr>
<tr>
<td>DELIVER</td>
<td>17-4</td>
</tr>
<tr>
<td>DESTROY</td>
<td>5-2</td>
</tr>
<tr>
<td>DETACH</td>
<td>16-2</td>
</tr>
<tr>
<td>DETACHJOB</td>
<td>16-3</td>
</tr>
<tr>
<td>DISCARD</td>
<td>5-7</td>
</tr>
<tr>
<td>DISCONNECT</td>
<td>5-3</td>
</tr>
<tr>
<td>DOCUMENTS</td>
<td>17-4</td>
</tr>
<tr>
<td>ECCE</td>
<td>8-10</td>
</tr>
<tr>
<td>EDIT</td>
<td>8-1</td>
</tr>
<tr>
<td>FILES</td>
<td>5-1</td>
</tr>
<tr>
<td>FORTE</td>
<td>11-1</td>
</tr>
<tr>
<td>HAZARD</td>
<td>5-6</td>
</tr>
<tr>
<td>HELP</td>
<td>4-7</td>
</tr>
<tr>
<td>IMP</td>
<td>11-1</td>
</tr>
<tr>
<td>INSERT</td>
<td>11-9</td>
</tr>
<tr>
<td>INSERTMACRO</td>
<td>16-13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINK</td>
<td>11-5</td>
</tr>
<tr>
<td>LIST</td>
<td>6-4</td>
</tr>
<tr>
<td>LOOK</td>
<td>8-9</td>
</tr>
<tr>
<td>MESSAGES</td>
<td>17-5</td>
</tr>
<tr>
<td>METER</td>
<td>17-2</td>
</tr>
<tr>
<td>NEWDIRECTORY</td>
<td>11-8</td>
</tr>
<tr>
<td>NEWPDFILE</td>
<td>4-4</td>
</tr>
<tr>
<td>NEWSMFILE</td>
<td>9-1</td>
</tr>
<tr>
<td>OBEY</td>
<td>17-5</td>
</tr>
<tr>
<td>OBEYJOB</td>
<td>16-4</td>
</tr>
<tr>
<td>OFFER</td>
<td>5-4</td>
</tr>
<tr>
<td>OPTION</td>
<td>17-5</td>
</tr>
<tr>
<td>PARM</td>
<td>11-2</td>
</tr>
<tr>
<td>PASSWORD</td>
<td>17-1</td>
</tr>
<tr>
<td>PERMIT</td>
<td>5-4</td>
</tr>
<tr>
<td>QUIT</td>
<td>17-8</td>
</tr>
<tr>
<td>RECALL</td>
<td>8-10</td>
</tr>
<tr>
<td>RECAP</td>
<td>8-21</td>
</tr>
<tr>
<td>REMOVE</td>
<td>11-10</td>
</tr>
<tr>
<td>REMOVEMACRO</td>
<td>16-3</td>
</tr>
<tr>
<td>RENAME</td>
<td>5-2</td>
</tr>
<tr>
<td>RESTORE</td>
<td>5-7</td>
</tr>
<tr>
<td>RUN</td>
<td>11-5</td>
</tr>
<tr>
<td>SEND</td>
<td>6-5</td>
</tr>
<tr>
<td>SETMODE</td>
<td>17-9</td>
</tr>
<tr>
<td>SHOW</td>
<td>8-21</td>
</tr>
<tr>
<td>STOP</td>
<td>17-10</td>
</tr>
<tr>
<td>SUGGESTION</td>
<td>17-11</td>
</tr>
<tr>
<td>TELL</td>
<td>17-11</td>
</tr>
<tr>
<td>TIDYDIR</td>
<td>11-9</td>
</tr>
<tr>
<td>USERS</td>
<td>17-2</td>
</tr>
</tbody>
</table>

Table 4.6: Edinburgh Subsystem Commands (Alphabetical Order)
Extract from SUBST.VIEW

2  How to use VIEW

To look at section 2.1, press the 'return' key.

Please send any problems/suggestions to
Tony Gibbons, ERCC04

Contents

2.1  VIEWing a File
2.2  Preparing your own material

2.1  VIEWing a File

VIEW displays a file in "pagefuls", like a book. To get to the next page, press the 'return' key.

The little message "...more" at the bottom means that there is more in this section!

You can always get to the next page, even if it's not in the same section, by pressing the 'return' key.

(You always have to press the 'return' key to make VIEW do something)

When VIEW displays a continuation page, like this one, it puts the continuation number (in this case 2) in the top right hand corner.

You can go to continuation page n in a section by typing /n

/ by itself takes you to the last page of the section

As an example, type /3 and then press the 'return' key to get to the next page in this section
If you just want to get back to the previous page, type - (a 'minus') and then press the 'return' key.

If you want, you can go straight to continuation page n in a section by typing /n after the section number.

For example, type 2.1/4 to get to the next page.

Sometimes it is more convenient to refer to a section by its name rather than its number. You can do this by typing its name or, even, just part of its name.

The note below says how this is done more precisely.

For example, type VIEWING/5 and press the 'return' key to get to the next page.

EW scans the full contents looking first for an exact match. If there is more than one, it replies 'non-unique'. If there is only one exact match, it displays that section. If there are no exact matches, it scans the full contents again looking for a matching substring. If there is more than one, it displays a list of all those found. If there is only one, it displays that section. Otherwise, it replies 'no match found'.

When VIEW displays a section, it first displays the preface and then gives the names of the subsections at the next level.

You can get a list of these names, and of the names of all the sub-subsections etc by typing C.

If you do this now, you will have to type 2.1/5 to get back here.
If you wish to look at subsection n of the current section, it is sufficient to type in .n

Thus if you are looking at 2.1 and you type in .1, you will go to 2.1.1

For example, type .1 to look at subsection 2.1.1.

To exit from VIEW, type
Q (or QUIT, E, ENQ or STOP)

When you have found your way around section 2.1, do have a look at 2.1.2 and 2.1.3.
2.1.3  Subsidiary Files

When you enter VIEW, you are VIEWing the 'basefile'. One section of the basefile contains a list of references to other files that you can VIEW. You VIEW one of these by typing its name or its number. They are distinguished from ordinary sections in the contents by an asterisk.

When you 'get into' the subsidiary file, it has its own section numbers starting from 1 just like the basefile.

You get out of a subsidiary file and return to the basefile by typing R.

You can also VIEW a subsidiary file by typing <filename>

This is like looking up a reference in a book.

You can go down several levels in this way, each subsidiary file serving as the basefile for the next level.

2.2  Preparing your own material

In its simplest form, VIEW can be used on an ordinary character file (or a character file member of a PD file) in much the same way as LOOK. You give the command

VIEW (filename)
Unless the file has been structured as described below, VIEW will simply split it into 'continuation' pages, addressable by typing /n.

The two ways of structuring a file are:
(i) by using directives, see 2.2.1
(ii) by using partitioned files, see 2.2.2

Section 2.2.3 describes certain special functions which can be used to 'personalise' the output if you wish.

Contents

2.2.1 Structuring with Directives
2.2.2 Structuring using PD files
2.2.3 Special Functions
2.2.4 Efficiency considerations
2.2.5 Pointing at a subsidiary file
2.2.6 Aesthetic considerations

2.2.1 Structuring with Directives

You can use directives to structure a character file into a 'preface' followed, optionally, by one or more 'sections'. The preface is all the lines of text from the start of the file to the first section.

Division into sections is accomplished by using brackets as follows:

```
preface introducing sections 1, 2, ...
!< name of section 1
  content of section 1
>!
!< name of section 2
  content of section 2
>!
```

etc

Any section can be subdivided by using more pairs of brackets within the outer brackets eg

```
preface introducing sections 1, 2,...
!< name of section 1
  preface introducing 1.1, 1.2,...
    of section 1
!< name of section 1.1
      content of section 1.1
>!
!< name of section 1.2
      content of section 1.2
>!
!< name of section 2
```

etc

Such subdivisions can be extended to any depth, the 'content' of each section being a preface optionally followed by one or more subsections.
If the later parts of a file are left unstructured, the earlier structured part is the only part which can be VIEWed. An extreme case of this is when a file contains as the only directive the terminator '!' at the start of a line. The text up to that terminator is then treated as a 'preface' (split up into pages by VIEW if it is long enough) and the text after the terminator is not VIEWable.

2.2.2 Structuring using PDL files

If VIEW is used on a PDL file, it lists the (viewable) members in alphabetical order and numbers them 1, 2, 3...
Then, to view a specific member, you just type the corresponding number or name.

If the PDL file has a member called 'PREFACE', this is listed in full before the names of the other members.

It is appropriate to observe here that EMAS allows a member of a PDL file to be another PDL file, to any depth and VIEW operates similarly at each level.

The members are given in alphabetical order of member name. However, you can give an 'alias' to a member by writing

!TITLE title to be used

as the first line of the member.

Note! Read section 2.2.4

2.2.3 Special Functions

The VIEW program has a number of built-in functions which can be used to insert items like today's date in the material being viewed.

A function is written as \[f_n\] where \(n\) is an integer. The following functions are available

1 Date \(28/06/81\)
2 Time \(20.04.12\)
3 User no. ECN123
4 Delivery LAURENCE_BYRD_HOPE_PARK_SQUARE
5 Surname F.Ross
6 DCP \(2972\)

2.2.4 Efficiency considerations

When VIEW is entered, it builds a directory to the file so that subsequent requests to VIEW a particular section can be dealt with rapidly.

Building the directory itself takes a significant amount of time so, in the case of a PDL file, VIEW attempts to insert the directory as a new member called VIEWDIR. If this is successful, subsequent calls of VIEW for the same file are much quicker.

If the PDL file is subsequently changed, the next time it is VIEWed the VIEWDIR is recalculated and replaced.
Consequently, the name of the file which is being used by other people, should remember to keep the VIEWDIR up to date by VIEWing the PD file after making changes.

PD files should always be used for large files.

2.2.5 Pointing at a subsidiary file

You do this by including a query section of the form

```
!<filename>
```

When you return from VIEWing the file, by typing R, you go 'up'.

2.2.6 Aesthetic considerations

It is easy to set up some kind of file to be viewed but you dont get the best effect quite so easily!

Remember that your material will be displayed in 20 line chunks. Try and make this a logical division.

Also remember that typically a terminal's line length is 72 characters. Lines longer than this will be displayed but will wrap round and upset the line count.

Lines longer than 100 characters are truncated.

To make the names facility most effective, choose your names to be

short,
meaningful
and
unique.

```
0682 THEVIEWOUT IIK LISTED T4D LF4D

***EMAS 2772 EMAS*** ECM125 P.Ross
***EMAS 2772 EMAS*** ECM125 P.Ross
***EMAS 2772 EMAS*** ECM125 P.Ross
***EMAS 2772 EMAS*** ECM125 P.Ross
***EMAS 2772 EMAS*** ECM125 P.Ross
***EMAS 2772 EMAS*** ECM125 P.Ross
***EMAS 2772 EMAS*** ECM125 P.Ross
***EMAS 2772 EMAS*** ECM125 P.Ross
```
**EMAS 2972**

**EMAS 2972**

**EMAS 2972**

**EMAS 2972**

**EMAS 2972**

**EMAS 2972**

**EMAS 2972**

**EMAS 2972**

**EMAS 2972**

90995 F00 2K LISTED T15 LP15

command: dir

*EMAS4#MUNG

*EVANS

MATH

NL

PLAN

SS#DIR

SS#OPT

TECH

JST

command: dir evans

file: *EVANS Type: PARTITIONED Length: 4720 Bytes

last altered: 12/09/81 at 00.24.16

access Permissions: Self: All Others: None

current users: 1

members:

VANS FIGURE XEVANS

command: dir math

file: *MATH Type: PARTITIONED Length: 33632 Bytes

last altered: 12/09/81 at 00.27.35

access Permissions: Self: All Others: None

current users: 1

members:

BREADT DIVIDE EQUAL FORMUL HEURIS

MATH

command: dir nl

file: *NL Type: PARTITIONED Length: 11248 Bytes

last altered: 12/09/81 at 00.24.57

access Permissions: Self: All Others: None

current users: 1

members:

ATNOLD ELIZA ELIZANEW PARSE QA

NL

command: dir plan

file: *PLAN Type: PARTITIONED Length: 5056 Bytes

last altered: 12/09/81 at 00.26.02

access Permissions: Self: All Others: None

current users: 1

members:

VANS NEW AMO
dir teach
file: *TEACH  Type: PARTITIONED  Length: 4984 Bytes
last altered: 12/09/81 at 00.23.50
access Permissions:  Self: All  Others: None
members: NFER  MANDC  RANDOM  RANDOMOLD  READ  READIN  TEACH


dir winst
file: *WINST  Type: PARTITIONED  Length: 17840 Bytes
last altered: 12/09/81 at 00.25.39
access Permissions:  Self: All  Others: None
members: RCH1PRB  ARCHPRB  BLOCKPRB  ISOLPRB  PAIRPRB  WINST  WINST

90995 F00 2K LISTED T15 LP15

**EMAS 2972 EMAS*** ECMIO2 A. Bundy  ALAN_BUNDY_HOPE_PARK_SQUARE
**EMAS 2972 EMAS*** ECMIO2 A. Bundy  ALAN_BUNDY_HOPE_PARK_SQUARE
**EMAS 2972 EMAS*** ECMIO2 A. Bundy  ALAN_BUNDY_HOPE_PARK_SQUARE
**EMAS 2972 EMAS*** ECMIO2 A. Bundy  ALAN_BUNDY_HOPE_PARK_SQUARE
**EMAS 2972 EMAS*** ECMIO2 A. Bundy  ALAN_BUNDY_HOPE_PARK_SQUARE
**EMAS 2972 EMAS*** ECMIO2 A. Bundy  ALAN_BUNDY_HOPE_PARK_SQUARE
**EMAS 2972 EMAS*** ECMIO2 A. Bundy  ALAN_BUNDY_HOPE_PARK_SQUARE
**EMAS 2972 EMAS*** ECMIO2 A. Bundy  ALAN_BUNDY_HOPE_PARK_SQUARE
**EMAS 2972 EMAS*** ECMIO2 A. Bundy  ALAN_BUNDY_HOPE_PARK_SQUARE
**EMAS 2972 EMAS*** ECMIO2 A. Bundy  ALAN_BUNDY_HOPE_PARK_SQUARE
*Evans’ Geometric Analogy Program - Rational Reconstruction*/
*Alan Bundy 26.10.79*/

*top level program*/
van(FigA, FigB, FigC, AnsList, Ans) :-
    find_rule(FigA, FigB, Rule), rule_is(Rule),
    apply_rule(Rule, FigC, AnsObJs, AnsRel, Sims),
    ans_desc(AnsObJs, AnsRels, Sims),
    select_result(FigC, AnsList, AnsObJs, AnsRels, Sims, Ans),
    ans_is(Ans).

*find rule given figures*/
ind_rule(FigA, FigB, Rule) :-
    relations(FigA, Source), relations(FigB, Target),
    objects(FigA, Alist), objects(FigB, Blist),
    similarities(FigA, FigB, Triples),
    select_set(Triples, Matches),
    takeaway1(Alist, Matches, Removals),
    takeaway2(Blist, Matches, Adds),
    make_rule(Removals, Adds, Matches, Source, Target, Rule).

* Apply rule to figc to produce answer*/
apply_rule(rule(Removals, Adds, Matches, Source, Target),
    FigC, AnsObJs, Target, Matches) :-
    relations(FigC, FigDesc), objects(FigC, ObList),
    seteq(FigDesc, Source),
    maplist(second, Matches, NewList),
    append(NewList, Adds, AnsObJs).

selectResult from those provided */
select_result(FigC, [FigN|Rest], AnsObJs, AnsRel, AnsSims, FigN) :-
    relations(FigN, NRel), seteq(NRel, AnsRel),
    similarities(FigC, FigN, NSims), seteq(NSims, AnsSims),
    objects(FigN, NObJs), seteq(NObJs, AnsObJs).

select_result(FigC, [FigN|Rest], AnsObJs, AnsRel, AnsSims, Ans) :-
    select_result(FigC, Rest, AnsObJs, AnsRel, AnsSims, Ans).

*select legal subset of similarity triples for matches*/
select_set(Tripe, Match) :-
    select_set(\[\], Match).

select_set(Aused, Bused, \[\] ,\]) :-
    not(member(Aobj, Bused)), not(member(Bobj, Aused)),
    select_set([Aobj|Aused], [Bobj|Bused], Rest, Rest).

select_set(Aused, Bused, [[Aobj, Bobj, Trans]|Rest], Rest) :-
    select_set(Aused, Bused, Rest, Rest).

*take away the triples from the list*/
maplist(first, Triples, Firsts), subtract(List, Firsts, Ans).

akeaway2(List, Triples, Ans) :-
    maplist(second, Triples, Seconds), subtract(List, Seconds, Ans).

* First and second elements of a list */
first([A, B, C], A).
second([A, B, C], B).

* Make rule from descriptions inherited from figs a & b*/
ake_rule(Removals, Adds, Matches, Source, Target, Rule) :-
    maplist(first, Matches, Spairs), maplist(second, Matches, Tpairs),
    append(Removals, Spairs, L1), append(L1, Tpairs, L2),
    append(L2, Adds, Consts),
    unbind(Consts, Substs),
    subst(Substs, rule(Removals, Adds, Matches, Source, Target), Rule).

*find corresponding variable for each constant and produce substitution*/
ubind([], true).

*nd([Const,Rest], Const=X & Rest1) :-
  unbind(Rest, Rest1).

* Messages */
---------*/
le_is(rule(Removals, Adds, Matches, Source, Target)) :-
    writeln('Rule is:'),
    writeln('remove: %t'),
    writeln('add: %t'),
    writeln('match: %t'),
    writeln('source: %t'),
    writeln('target: %t \n\n', [Removals, Adds, Matches, Source, Target]).

s_desc(Objs, Rel, Sims) :-
    writeln('Answer description is:'),
    writeln('objects: %t'),
    writeln('relations: %t'),
    writeln('similarities: %t \n\n', [Objs, Rel, Sims]).

s_is(Ans) :-
    writeln('Answer is %t\n\n', [Ans]).
/*figures*/
/*test descriptions for evans program*/
/*Alan Bundy 26.10.79*/

problem1(Ans) :- evans(figa, figb, figc, [fig1, fig2, fig3, fig4, fig5], Ans).
problem2(Ans) :- evans(figa, figb, figc, [fig1, fig2, fig3, fig4a, fig5], Ans).
problem3(Ans) :- evans(figa, figb, figc, [fig1, fig2, fig3, fig5], Ans).

objects(figa, [tri1, tri2]).
relations(figa, [[inside, tri1, tri2]]).

objects(figb, [tri3]).
relations(figb, []).

similarities(figa, figb, [[tri2, tri3, direct], [tri1, tri3, [scale, 2]]]).

objects(figc, [square, circle]).
relations(figc, [[inside, square, circle]]).

objects(fig1, [circle2, circle3]).
relations(fig1, [[inside, circle2, circle3]]).
similarities(figc, fig1, [[circle, circle3, direct], [circle, circle2, [scale, half]]]).

objects(fig2, [square2]).
relations(fig2, []).

similarities(figc, fig2, [[square, square2, direct]]).

objects(fig3, [tri4, circle4]).
relations(fig3, [[inside, tri4, circle4]]).

similarities(figc, fig3, [[circle, circle4, direct]]).

objects(fig4, [circle5]).
relations(fig4, []).

similarities(figc, fig4, [[circle, circle5, direct]]).

objects(fig4a, [square3]).
relations(fig4a, []).

similarities(figc, fig4a, [[square, square3, [scale, 2]]]).

objects(fig5, [tri5]).
relations(fig5, []).

similarities(figc, fig5, []).
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy

90979 MATH_XMATH 1K LISTED T15 LP15

math
cover
read
divide
qual
ormul
euris
odel
erit
emant
imple
item
ify
END

90979 MATH_XMATH 1K LISTED T15 LP15

**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy

ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE

RECEI
Using PROLOG as a theorem prover:

Try goal `equal(y, x)`.

lan Bundy 16.6.81 */

qual(x, y). % The Hypothesis
r -l(X, X). % The Reflexive Axiom
qual(U, W) :- equal(U, V), equal(W, V). % The Twisted Transitivity Axiom

* MINI-PROJECTS

Try goal `- equal(y, x)`.

Experiment by switching the order of the above axioms and trying the same goal. What sort of bad behaviour emerges? How could it be voided?

Experiment with different axioms, e.g. the group theory example from p92 of 'Artificial Mathematicians'.

/*
simple Boyer-Moore theorem prover as in 
Chap. 11 The Productive Use of Failure 
in 'Artificial Mathematicians'.

The code written here is a simplified version of the algorithm 
written in a paper of J. Moore 
"Computational Logic: Structure Sharing and Proof of Program Properties", 
report CSL 75-2, which appeared also as Dept. of Computational Logic 

This is a simplified version of the theorem-prover described in the book 
A Computational Logic' by Boyer and Moore.

Variable and procedure names have been chosen to be the same 
as much as possible, and the mini-projects will be aided by the description 
in the paper.

% PROOF STRATEGY %

To prove a theorem use the following algorithm 
(On p. of 'Artificial Mathematicians'):
1. Try symbolic evaluation, recording the 
   reasons for failure in the list Analysis.
2. If unsuccessful, try proof by induction 
   using the previously generated failure list 
   to suggest the induction scheme.
3. Finally try generalising the theorem if the 
   induction was unsuccessful.

prove(A) :-
    writef(
        'Trying to prove \n        symbol_eval(A,B,Analysis),
    prove(B,Analysis).

prove(tt,_) :- % Symbolic evaluation was successful
    writef('Expression evaluated to \n            prove(A,Analysis)
    pickindvars(A,Analysis,Var),
    prove_by_induction(A,Var).

prove(A,Analysis) :-
    generalise(A,New),
    prove(New=tt).
Symbolic evaluation is performed by the procedure\[\text{symbol_eval}(A, \text{New}, \text{Analysis})\] which evaluates expression $A$ into expression $\text{New}$ producing failure bag $\text{Analysis}$.

Primitive pure LISP functions, i.e. $\text{car}$, $\text{cdr}$, $\text{cond}$ and equal are handled by an evaluator if they can be simplified. Otherwise function arguments are symbolically evaluated bottom-up, bottoming out on atomic expressions. Function definitions are expanded according to the criteria described in the section of code.

% EXPANSION OF FUNCTION DEFINITIONS%

```lisp
(symbol_eval(tt, tt, []) :- !. % Finished if expression evaluates to tt

(symbol_eval([], [], []) :- !.

(eval(A, A1), !, symbol_eval(A1, B, Analysis)).

(symbol_eval(A, B, Analysis) :- A = .. [Pred|Args], rewrite(Args, Args1, Anal), A1 = .. [Pred|Args1], expand(Pred, A1, A2, Fault), merge(Fault, Anal, Analysis), def_eval(A2, B)).

(rewrite([], [], []) :- !.

(rewrite(X, X, []) :- atomic(X), !.

(-ite([H | T], [H1 | T1], Analysis) :- symbol_eval(H, H1, Fault), rewrite(T, T1, Desc), merge(Fault, Desc, Analysis)).

(val(car([], []), []) :- !.

(val(cdr([], []), []) :- !.

(val(car(cons(H, T)), H) :- !.

(val(cdr(cons(H, T)), T) :- !.

(val(cond([], U, V), V) :- !.

(val(cond(cons(X, Y), U, V), U) :- !.

(val(equal(X, X), tt) :- !.

% EXPANSION OF FUNCTION DEFINITIONS%

% Functions that can be expanded according to their function definition are contained in an open_fn predicate. In rewriting expressions functions are expanded where possible using the predicate open_eval unless
% fault description is returned as described
% in section 3.2 of Moore’s paper.

xpand(Pred, Clause, Clause, Fault) :-
    open_fn(Clause, Newclause),
    ugliness(Pred, Newclause, Bomb),
    Bomb == [],
    !,
    make_fault(Bomb, Fault).

xpand(Pred, Clause, Newclause, []) :-
    open_eval(Clause, Newclause).

xpand(Pred, Clause, Clause, []).% pen_fn(append(X,Y),cond(X,cons(car(X),append(cdr(X),Y)),Y)) :- !.

pen_eval(append(X,Y),Clause) :-
    def_eval(car(X),Cl),
    def_eval(cdr(X),C2),
    def_eval(cond(X,cons(C1,append(C2,Y)),Y),Clause).

ef_eval(X,Y) :- eval(X,Y), !.

% Make up a fault description from the faults returned when trying to expand a recursively defined function.

ake_fault([],[]) :- !.

ake_fault([[fault(B,F)],[Bombs],[fault(B,F)],[Faults]]) :-
    make_fault(Bombs,Faults).

ake_fault([bomb(X)],[fault(bomb(X),fail([]))]).

ake_fault([fail(X)],[fault(bomb([]),fail(X))]).

ake Fault([bomb(X),fail(Y)],[fault(bomb(X),fail(Y))]).

% Is expression ugly?

Gliness(Pred,X,[]) :- atomic(X), !.

gliness(Pred,X,Analysis) :-
    nasty_car_or_cdr(X),
    !,
    analyse(Pred,X,Analysis).

gliness(_,X,Analysis) :-
    X=..[Pred|Args],
    find_ugly(0, Args, Analysis).

ind_ugly(_,[],[]) :- !.

ind_ugly(Pred,[H|T],Analysis) :-
    ugliness(Pred, H, H1),
    find_ugly(Pred, T, T1),
    append(H1,T1,Analysis),
    !.
\texttt{dd\_bomb(X, [], X) :- !.}

\texttt{dd\_bomb(Y, X, Z) :- append(Y, [X], Z).}

\texttt{nalyse(Pred, X, [bomb(X)]) :- loop\_threat(Pred, X), !.
 nalyse(Pred, X, [fail(X)]).
 oop\_threat(append, car(X)).
 oop\_threat(append, cdr(X)).

\texttt{ontrivial(X) :- memberchk(X, [cons(U, V), []]), !, fail.
ontrivial(X).}

\% PROOF BY INDUCTION \%
\% Find induction candidate
\% A simple majority vote is used to decide
\% which list to induct on.
\% This is calculated by max.

\texttt{ickindvars(_, Bag, Var) :-
 max(Bag, [[], Term, O, N],
 Term = fault(bomb(cdr(Var)), fail(_)).}

\texttt{ax([], A, A, N, N) :- !.}

\texttt{ax([pair(Pred, M) | Rest], Current, Ans, N, Num\_ans) :-
 M > N,
 !,
 max(Rest, Pred, Ans, M, Num\_ans).
 ax([_| Rest], Current, Ans, N, Num\_ans) :-
 max(Rest, Current, Ans, N, Num\_ans).}

\% Successively prove the base and step cases

\texttt{rove\_by\_induction(A, Literal) :-
 prove\_base(A, Literal),
 prove\_step(A, Literal).}

\texttt{rove\_base(A, Literal) :-
 writef("\n Base case"),
 subst(Literal = [], A, New),
 prove(New),
 !.

 % To prove the step case, substitute the appropriate
 % cons expression into the clause, symbolically
 % evaluate as much as possible, then fertilise
 % to prove the expression

\texttt{rove\_step(A, Literal) :-
 writef("\n Step case"),
 subst(Literal = cons(a1, Literal), A, New),
 symbol\_eval(New, Clause, ).

\texttt{rove\_by\_induction(A, Literal) :-}}
prove(Newclause).

ertilise(equal(X, Y), Clause, New) :- subst(X=Y, Clause, New).

% PROOF BY GENERALISATION %

% Code to be written

test cases
:- prove(equal(append([], x), x)).
:- prove(equal(append(a, append(b, c)), append(append(a, b), c))).

* Mini-projects

• Add definitions of functions like reverse and copy to enable the theorem-prover to work on other examples. Explain how the theorem-prover might be modified to overcome any small problems that might arise.

• See more sophisticated criteria to choose the induction variable, or more generally the induction schema to be used. This will probably involve changing the way the failures are returned in the variable Analysis.

• Write a more powerful version of fertilise.

• Write code to perform generalise.

%

% merge merges two bags

merge([], Bag, Bag) :- !.

merge(Bag, Var,Hack) :- var(Var), !, merge(Bag, [variable], Hack).

merge([pair(Ugly, K) | T], Bag, Newbag) :-
    select(pair(Ugly, N), Bag, Rest), !,
    M is N + K,
    merge(T, [pair(Ugly, M) | Rest], Newbag).

merge([pair(Ugly, N) | T], Bag, Newbag) :-
    !,
    merge(T, [pair(Ugly, N) | Bag], Newbag).

merge([H | T], Bag, Newbag) :-
    select(pair(H, N), Bag, Rest), !,
    M is N + 1,
    merge(T, [pair(H, M) | Rest], Newbag).

merge([H | T], Bag, Newbag) :-
    merge(T, [pair(H, 1) | Bag], Newbag).
<table>
<thead>
<tr>
<th>EMAS 2972 EMAS*** ECMO2 A. Bundy</th>
<th>ALAN_BUNDY_HOPE_PARK_SQUARE</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMAS 2972 EMAS*** ECMO2 A. Bundy</td>
<td>ALAN_BUNDY_HOPE_PARK_SQUARE</td>
</tr>
<tr>
<td>EMAS 2972 EMAS*** ECMO2 A. Bundy</td>
<td>ALAN_BUNDY_HOPE_PARK_SQUARE</td>
</tr>
<tr>
<td>EMAS 2972 EMAS*** ECMO2 A. Bundy</td>
<td>ALAN_BUNDY_HOPE_PARK_SQUARE</td>
</tr>
<tr>
<td>EMAS 2972 EMAS*** ECMO2 A. Bundy</td>
<td>ALAN_BUNDY_HOPE_PARK_SQUARE</td>
</tr>
<tr>
<td>EMAS 2972 EMAS*** ECMO2 A. Bundy</td>
<td>ALAN_BUNDY_HOPE_PARK_SQUARE</td>
</tr>
<tr>
<td>EMAS 2972 EMAS*** ECMO2 A. Bundy</td>
<td>ALAN_BUNDY_HOPE_PARK_SQUARE</td>
</tr>
</tbody>
</table>
Breadth First Search Theorem Prover.
JUAL contains test example.

lan Bundy 16.6.81

* Top Goal */

d :- breadth(0).

* Breadth First Search Strategy */

breadth(N) :-
N1 is N+1,
repeat(resolve(input,N,N1)),
repeat(resolve(N,input,N1)),
breadth(N1).

* Repeat as many times as possible */

repeat(Goal) :- Goal, fail. % Keep trying and failing
repeat(Goal). % and succeed only when you run out of things to do

* Resolution Step */

level(N1, N2, N) :-
find_clause(Parent1, Consequent1, Antecedent1, N1), % Find clauses at
find_clause(Parent2, Consequent2, Antecedent2, N2), % appropriate depth
select(Literal, Consequent1, RestConsequent), % find a common literal
select(Literal, Antecedent2, RestAntecedent), % return leftovers
append(RestConsequent, Consequent2, Consequent), % cobble leftovers together
append(Antecedent1, RestAntecedent2, Antecedent),
record_clause(Consequent, Antecedent, N). % record new clause

* Record Existence of New Clause */
record_clause([J, [J], N]) :-
writef('Success! Empty Clause Found\n\n'), !, % test for empty clause
% and stop
record_clause(Consequent, Antecedent, N) :- % test for loop
find_clause(Name, Consequent, Antecedent, M), !. % i.e. clause with same inn
record_clause(Consequent, Antecedent, N) :- !, % record new clause
gensym(clause, Name), % make up new name
assert(clause(Name, Consequent, Antecedent, N)), % assert clause
writef('%t is name of new resolvant %l <= %l at depth %t\n\n',
[Name, Consequent, Antecedent, N]). % tell user
ind_clause(Name, Consequent, Antecedent, 0) :-
    clause(Name, Consequent, Antecedent, topclause), !.
ind_clause(Name, Consequent, Antecedent, N) :-
    clause(Name, Consequent, Antecedent, N).

* MINI-PROJECTS

Try this theorem prover with the clauses of file EQUAL.

Experiment by making up some clauses of your own and trying them out.

Modify the theorem prover to print out the solution when it has found it.

Modify the theorem prover to remove the input restriction.

Modify the theorem prover to incorporate the literal selection restriction.

Build a depth first theorem prover along the same lines.

90981 MATH_BREADT 3K LISTED T15 LP15

**EMAS 2972 EMAS*** ECMIO2 A. Bundy ALAN_BUNDY HOPE_PARK_SQUARE
**EMAS 2972 EMAS*** ECMIO2 A. Bundy ALAN_BUNDY HOPE_PARK_SQUARE
**EMAS 2972 EMAS*** ECMIO2 A. Bundy ALAN_BUNDY HOPE_PARK_SQUARE
**EMAS 2972 EMAS*** ECMIO2 A. Bundy ALAN_BUNDY HOPE_PARK_SQUARE
**EMAS 2972 EMAS*** ECMIO2 A. Bundy ALAN_BUNDY HOPE_PARK_SQUARE
**EMAS 2972 EMAS*** ECMIO2 A. Bundy ALAN_BUNDY HOPE_PARK_SQUARE
**EMAS 2972 EMAS*** ECMIO2 A. Bundy ALAN_BUNDY HOPE_PARK_SQUARE
**EMAS 2972 EMAS*** ECMIO2 A. Bundy ALAN_BUNDY HOPE_PARK_SQUARE
**EMAS 2972 EMAS*** ECMIO2 A. Bundy ALAN_BUNDY HOPE_PARK_SQUARE
**EMAS 2972 EMAS*** ECMIO2 A. Bundy ALAN_BUNDY HOPE_PARK_SQUARE
* Heuristics Search Theorem Prover.
3UAL contains test example.

Ian Bundy 19.6.81 */

* Top Goal */

% Pick the clause with best score
\[ \text{pick\_best}(\text{Fringe}, \text{Current}, \text{Rest}), \quad \% \text{Pick an input clause} \]
\[ \text{findall}(\text{Clause}, \text{successor}(\text{Current}, \text{Clause}), \text{New}\text{Clauses}), \quad \% \text{Put them on fringe} \]
\[ \text{heuristic}(\text{New}\text{Fringe}). \quad \% \text{and recurse} \]

% Record the clause
\[ \text{record\_clause}(\text{Consequent}, \text{Antecedent}, \text{Name}). \quad \% \text{Record Existence of New Clause} \]
\[ \text{record\_clause}([], [], \text{Name}) :- \quad !, \quad \text{fail}. \]
\[ \text{record\_clause}(\text{Consequent}, \text{Antecedent}, \text{Name}) :- !, \quad \% \text{make up a name} \]
\[ \text{gensym}(\text{clause}, \text{Name}), \quad \% \text{get some of clause} \]
\[ \text{record\_clause}(\text{Consequent}, \text{Antecedent}, \text{Name}). \]

* Resolution Step */

% Get the two parents
\[ \text{resolve}(\text{Parent1}, \text{Parent2}, \text{Resolvant}) :- \]
\[ \text{clause}(\text{Parent1}, \text{Consequent1}, \text{Antecedent1}, \text{N1}), \quad \% \text{Get the two parents} \]
\[ \text{clause}(\text{Parent2}, \text{Consequent2}, \text{Antecedent2}, \text{N2}), \]
\[ \text{select}(\text{Literal1}, \text{Consequent1}, \text{RestConseq1}), \quad \% \text{Select a common literal} \]
\[ \text{select}(\text{Literal2}, \text{Antecedent2}, \text{RestAnteced2}), \]
\[ \text{append}(\text{RestConseq1}, \text{Consequent2}, \text{Consequent}), \quad \% \text{Join the odd bits together} \]
\[ \text{append}(\text{Antecedent1}, \text{RestAnteced2}, \text{Antecedent}), \]
\[ \text{record\_clause}(\text{Consequent}, \text{Antecedent}, \text{Resolvant}). \quad \% \text{Record the clause} \]

* Record Existence of New Clause */

% test for empty clause
\[ \text{record\_clause}([], [], \text{empty}) :- \quad !, \quad \% \text{test for empty clause} \]
\[ \text{writef}(\text{'Success! Empty Clause Found\n\n'}, !), \quad \% \text{tell the user} \]
\[ \text{aborted}. \quad \% \text{and stop} \]

% test for loop
\[ \text{record\_clause}(\text{Name}, \text{Consequent}, \text{Antecedent}, \text{M}), !, \quad \% \text{test for loop} \]
\[ \text{fail}. \quad \% \text{fail.} \]
\[ \text{record\_clause}(\text{Consequent}, \text{Antecedent}, \text{Name}) :- !, \quad \% \text{record new clause} \]
\[ \text{gensym}(\text{clause}, \text{Name}), \quad \% \text{make up a name} \]
\[ \text{evaluate}(\text{Consequent}, \text{Antecedent}, \text{M}). \quad \% \text{set some of clause} \]
writef(‘It is name of new resolvent %1 <- %1 with score %t

* Evaluation Function on Clauses (length of clause) */

valuate(Consequent, Antecedent, Score) :-
    length(Consequent, C), % add length of rhs
    length(Antecedent, A), % to length of lhs
    Score is C+A. % to get clause length

* Pick clause with best score (i.e. lowest) */

pick_best([Hd|T1], Choice, Rest) :-
    clause(Hd, _, _, N), % Get score of first clause
    pick_best(T1, Hd, N, Choice, Rest). % and run down list remembering best so far

pick_best([], Hd, N, Hd, []). % When you get to the end return running score

pick_best([Hd1|T1], Hd, N, Choice, [Hd3|Rest]) :-
    clause(Hd1, _, _, N1), % Get score of first clause
    compare(Hd, N, Hd1, N1, Hd2, N2, Hd3, N3), % Compare with running score and or
    pick_best(T1, Hd2, N2, Choice, Rest). % recurse with new best score

compare(Hd, N, Hd1, N1, Hd, N, Hd1, N1) :- % put running score first
    N =< N1, !. % unless new score is best

compare(Hd, N, Hd1, N1, Hd1, N1, Hd, N). % Otherwise put new score first

* Find a clause with score N */

ind_clause(Name, Consequent, Antecedent, N) :-
    clause(Name, Consequent, Antecedent, topclause), !,
    evaluate(Consequent, Antecedent, N).

ind_clause(Name, Consequent, Antecedent, N) :-
    clause(Name, Consequent, Antecedent, N).

INI-PROJECTS

- Try this theorem prover with the equality clauses of EQUAL.
- Experiment with clauses of your own devising.
- Modify the theorem prover to print out the solution when it has found it.
- Modify the theorem prover to remove the input restriction.
- Experiment with different versions of the evaluation function by editing 'evaluate' (see section 6.5.3 of 'Artificial mathematicians').
test example for BREADT and HEURIS.
Symmetry can be inferred from reflexivity and twisted transitivity
notes p62).

 Alan Bundy 22.6.81 */

clause(hypothesis, [equal(x,y)], []). % Input clauses
clause(reflexive, [equal(X,X)], []). % Input clauses
clause(twisted, [equal(U,W)], [equal(U,V), equal(W,V)], []). % Input clauses

use(goal, []). % Top clause

Alan Bundy 22.6.81 */

clause(hypothesis, [equal(x,y)], []). % Input clauses
clause(reflexive, [equal(X,X)], []). % Input clauses
clause(twisted, [equal(U,W)], [equal(U,V), equal(W,V)], []). % Input clauses

use(goal, []). % Top clause
Depth First Theorem Prover

with vetting by use of interpretations and incorporating input restriction.

with MODEL

[VIDE contains test example]

Ian Bundy 22.6.81 */

% Top Goal */

clause(Goal,_,_,topclause), % Find the top clause
semantic(Goal). % and away you go

* Depth first theorem prover */

semantic(Old) :-
  sucessor(Old,New), % Find a successor to the current clause
  vet(New), % check that it is unsatisfiable
  semantic(New). % and recurse

* Clause is a resolvant of Current with an input clause */

successor(Current,Clause) :-
  clause(Input,_,_,input), % Pick an input clause
  ( resolve(Current,Input,Clause) ; % resolve it with the
    resolve(Input,Current,Clause) ; % current clause
    paramodulate(Current,Input,Clause) ; % or paramodulate it
    paramodulate(Input,Current,Clause) ).

* Resolution Step */

resolve( Parent1, Parent2, Resolvant) :-
  clause(Parent1, Consequent1, Antecedent1, _), % Get the two parents
  clause(Parent2, Consequent2, Antecedent2, _),
  select(Literal1, Consequent1, RestConseq1), % Select a common literal
  select(Literal2, Antecedent2, RestAntecedent2), % and return the rest
  append(RestConseq1, Consequent2, Consequent), % Join the odd bits together
  append(Antecedent1, RestAntecedent2, Antecedent),
  record_clause(Consequent,Antecedent,Resolvant). % Record the clause

* Paramodulation Step */

paramodulate( Parent1, Parent2, Paramodulant ) :-
  clause(Parent1, Consequent1, Antecedent1, _), % Get the two parents
  clause(Parent2, Consequent2, Antecedent2, _),
  select(equall(T,S), Consequent1, RestConseq1), % select an equation
  replace(S,Consequent2,Antecedent2,NewConseq2,NewAntecedent2), % put it in the ot!
append(Antecedent1, NewAnte2, Antecedent),
record_clause(Consequent, Antecedent, Paramodulant). %record the clause

* Replace T by S (or S by T) in clause */

place(T, S, OldConse, OldAnte, NewConse, OldAnte) :-
  replace1(T, S, OldConse, NewConse).
place(T, S, OldConse, OldAnte, OldConse, NewAnte) :-
  replace1(T, S, OldConse, NewConse).
place(S, T, OldConse, OldAnte, NewConse, OldAnte) :-
  replace1(S, T, OldConse, NewConse).
place(S, T, OldConse, OldAnte, OldConse, NewAnte) :-
  replace1(S, T, OldConse, NewConse).

* Replace T by S in Old to get New */

place1(T, S, Old, New) :-
  some(replace2(T, S), Old, New).

place2(T, S, T, S) :-
  % replace this occurrence

place2(T, S, Old, New) :-
  Old =.. [Sym : OldArgs],
  replace1(T, S, OldArgs, NewArgs),
  New =.. [Sym : NewArgs].

* Record Existence of New Clause */

record_clause([], [], empty) :-
  % test for empty clause
  writef('Success! Empty Clause Found\n\n'), !, % tell user
  abort. % and stop
record_clause(Consequent, Antecedent, Name) :-
  % test for loop
  clause(Name, Consequent, Antecedent, []), !. % i.e. clause with same inn
record_clause(Consequent, Antecedent, Name) :- !, % record new clause
  gensym(clause, Name), % make up new name
  assert(clause(Name, Consequent, Antecedent, new)), % assert clause
  writef('%t is name of new resolvant %1 <- %1 \n\n',
  [Name, Consequent, Antecedent]). % tell user

* Apply Pred to just one element of list */

some(Pred, [Hd1 | T1], [Hd2 | T1]) :-
  apply(Pred, [Hd1, Hd2]). % apply it to this one
some(Pred, [Hd1 | T11], [Hd1 | T12]) :-
  some(Pred, T11, T12). % or one of the others

* MINI-PROJECTS

Try out theorem prover with the arithmetic clauses and models of file DIVIDE.
Experiment with some clauses and models of your own devising (See section 10 for some ideas).
Modify the theorem prover so that it works by breadth first search
Compare with file BREADT).

Modify the theorem prover so that it works by heuristic search
Compare with file HEURIS).
now to evaluate a clause in an interpretation 
provided it is variable free!!)
se with SEMANT
IVIDE contains test examples
lan Bundy 22.6.81 */

* Vet the clause in any interpretations */

\( \text{not satisfiable(Clause).} \) % Clause has no model

* Clause is satisfiable in some Interpretation */

\( \text{atifiable(Clause)} : - \)
\( \text{interpretation(Interp),} \) % If there is an interpretation
\( \text{model(Interp, Clause),} \) % in which Clause is true
\( \text{writef(’} %t \text{ rejected by } %t
\n\n\text{’, [Clause, Interp]).} \) % tell user

* Interpretation is a model of a Clause */

\( \text{model(Interp, Clause)} : - \)
\( \text{clause(Clause, Consequent, Antecedent, _),} \) % Get Clause definition
\( \text{checklist(is_true(Interp), Consequent),} \) % Check all lhs literals are true
\( \text{checklist(is_false(Interp), Antecedent).} \) % and all rhs ones are fals

* Evaluate expression in Interpretation */

\( \text{valuate( Interp, Integer, Integer ) : - integer(Integer), !.} \) % integers represent
\( \text{valuate( Interp, Constant, Value ) : -} \) % other constants have values assign
\( \text{atom(Constant), !, interpret(Interp, Constant, Value).} \)

\( \text{valuate( Interp, Complex, Value ) : -} \)
\( \text{Complex = .. [Sym | Args], !,} \) % recurse on arguments of
\( \text{maplist(valuate(Interp, Args, Vals),} \) % complex terms
\( \text{Complex1 = .. [Sym | Vals],} \) % then interpret topmost
\( \text{interpret(Interp, Complex1, Value).} \) % symbol

* Evaluation of clause (hacks for checklist) */

\( \text{s_true(Interp, Literal) : - evaluate(Interp, Literal, true).} \)
\( \text{s_false(Interp, Literal) : - evaluate(Interp, Literal, false).} \)
<table>
<thead>
<tr>
<th><strong>EMAS 2972</strong></th>
<th><strong>EMAS</strong>*</th>
<th><strong>ECMI02 A. Bundy</strong></th>
<th><strong>ALAN_BUNDY_HOPE_PARK_SQUARE</strong></th>
</tr>
</thead>
</table>

**EMAS 2972** | **EMAS*** | **ECMI02 A. Bundy** | **ALAN_BUNDY_HOPE_PARK_SQUARE** |

**EMAS 2972** | **EMAS*** | **ECMI02 A. Bundy** | **ALAN_BUNDY_HOPE_PARK_SQUARE** |

**EMAS 2972** | **EMAS*** | **ECMI02 A. Bundy** | **ALAN_BUNDY_HOPE_PARK_SQUARE** |

**EMAS 2972** | **EMAS*** | **ECMI02 A. Bundy** | **ALAN_BUNDY_HOPE_PARK_SQUARE** |

**EMAS 2972** | **EMAS*** | **ECMI02 A. Bundy** | **ALAN_BUNDY_HOPE_PARK_SQUARE** |

**EMAS 2972** | **EMAS*** | **ECMI02 A. Bundy** | **ALAN_BUNDY_HOPE_PARK_SQUARE** |

**EMAS 2972** | **EMAS*** | **ECMI02 A. Bundy** | **ALAN_BUNDY_HOPE_PARK_SQUARE** |

**EMAS 2972** | **EMAS*** | **ECMI02 A. Bundy** | **ALAN_BUNDY_HOPE_PARK_SQUARE** |

**EMAS 2972** | **EMAS*** | **ECMI02 A. Bundy** | **ALAN_BUNDY_HOPE_PARK_SQUARE** |
* DIVIDE.*

example for SEMANT and MODEL

```prolog
lan Bundy  22.6.81 */

* Clauses */

la (right, [not_div(X*Z, Y)], [not_div(X, Y)], input). % Input clauses

la (left, [not_div(Z*X, Y)], [not_div(X, Y)], input).

la (thirty, [equal(30, 2*3*5)], [], input).

la (hypothesis, [not_div(5, a)], [], input).

la (conclusion, [], [not_div(30, a)], topclause).  % Top clause

* Models */

* arith2 */

interpret(arith2).  % arith2 is an interpretation

interpret(arith2, a, 2).  % meaning of a

interpret(arith2, not_div(X, Y), false) :- % meaning of not_div
  O is Y mod X, !.

interpret(arith2, not_div(X, Y), true).

interpret(arith2, equal(X, Y), true) :- % meaning of equal
  X =:= Y, !.

interpret(arith2, equal(X, Y), false).

interpret(arith2, X*Y, Z) :- Z is X*Y. % meaning of *

* arith3 */

interpret(arith3).  % arith3 is an interpretation

interpret(arith3, a, 3).  % meaning of a

interpret(arith3, not_div(X, Y), false) :- % meaning of not_div
  O is Y mod X, !.

interpret(arith3, not_div(X, Y), true).

interpret(arith3, equal(X, Y), true) :- % meaning of equal
  Y =:= Y , !.
```

ARITH2 is an interpretation for the not divides example (see notes p124).
interpret(arith3, X*Y, Z) :- Z is X*Y. % meaning of *
implement depth first search rewrite rule system
see Artificial Mathematicians p 104.
see with UTIL

Ian Bundy 10.7.81 */

* Find Normal Form of Expression */

limize(Expression, NormalForm) :- % To put an expression in normal form:
  rewrite(Expression, Rewriting), % Rewrite it once
  normalize(Rewriting, NormalForm). % and recurse

ormalize(Expression, Expression) :- % The expression is in normal form
  not rewrite(Expression, _). % if it cannot be rewritten

* Rewrite Rule of Inference */

write(Expression, Rewriting) :- % To rewrite Expression
  rule(Name, LHS, RHS), % get a rule LHS => RHS
  replace(LHS, RHS, Expression, Rewriting), % replace LHS by RHS
  writef('%t rewritten to %t by rule %t\n', [Expression, Rewriting, Name]).

* Replace single occurrence of T by S in Old to get New */

place(T, S, S). % replace this occurrence

lace(T, S, Old, New) :- % replace one of the arguments
  Old =.. [Sym | OldArgs], % get the arguments
  some(replace(T, S, OldArgs, NewArgs), % replace one
  New =.. [Sym | NewArgs]. % put it all together again

* Apply Pred to just one element of list */

ome(Pred, [Hd1 | T1], [Hd2 | T1]) :-
  apply(Pred, [Hd1, Hd2]). % apply it to this one

ome(Pred, [Hd | T1], [Hd | T12]) :-
  some( Pred, T1, T12). % or one of the others

* Some rules */

ule(1, X*X, 0). % Algebraic Simplification rules
ule(2, 1*X, X).
ule(3, X, X).
ule(4, X+0, X).

* A Typical Problem */
MINI-PROJECTS

Try out the system with the arithmetic rules given above.

Experiment with some rules of your own devising (Suggestions can be found in chapter 9 and section 5.2 of 'Artificial Mathematicians').

Modify the system so that it works by: (a) call by value, (b) call by name (See p107 of 'Artificial Mathematicians').

/
* skolem.

skolem Normal Form Procedure
*FORMUL contains test examples)

lan Bundy 17.3.79 */

*operator declarations*/
  - op(400, xfy, ;). % Disjunction
  _p(300, xfy, <->). % Double Implication
  -op(400, xfy, ->). % Implication

*skolem normal form*/

skolem(Sentence, NormalForm) :- !,
  skolem(Sentence, NormalForm, [J, O]). % Normal calling pattern
  % assume no free vars and being ass

skolem(P <-> Q, (P1->Q1)&(Q1->P1), Vars, Par) :- !, % Double implication
  skolem(P, P1, Vars, Par), skolem(Q, Q1, Vars, Par).

skolem(all(X,P),P1,Vars,0) :- !,
  skolem(P,P1,[X|Vars],0). % Universal quantification

skolem(all(X,P),P2,Vars,1) :- !,
  gensym(f,Fs), univ(Fs,Vars,F),
  subst(X=F,P,P1),
  skolem(P1,P2,Vars,1). % dual case

skolem(some(X,P),P2,Vars,0) :- !,
  gensym(f,Fs), univ(Fs,Vars,F),
  subst(X=F,P,P1),
  skolem(P1,P2,Vars,0). % Existential quantification

skolem(some(X,P),P1,Vars,1) :- !,
  skolem(P,P1,[X|Vars],1). % Existential quantification
dual case

skolem(P->Q,P1->Q1,Vars,Par) :- !,
  opposite(Par, Par1), skolem(P,P1,Vars,Par1),
  skolem(Q, Q1, Vars, Par). % Implication

skolem(P;Q,P1;Q1,Vars,Par) :- !,
  skolem(P,P1,Vars,Par), skolem(Q,Q1,Vars,Par). % Disjunction

skolem(P&Q,P1&Q1,Vars,Par) :- !,
  skolem(P,P1,Vars,Par), skolem(Q,Q1,Vars,Par). % Conjunction

skolem(not P, not P1,Vars,Par) :- !,
  opposite(Par, Par1), skolem(P,P1). % Negation
*opposite parities*/
\[\text{opposite}(0,1).
\text{opposite}(1,0).
\]

*univ - apply args to symbol*/
\[\text{univ}(F, V, F) :- !,
F =.. [F, V].\]

* MINI-PROJECTS

Try out on various formulae.

Modify the program to deal with bounded quantification, e.g.
\[\text{all}_\text{in}(X, \text{Set}, P) - \text{meaning, for all } X \text{ in } \text{Set, } P \text{ is true.}\]

Build programs for putting formulae in: (a) Prenex Normal Form, (b) Conjunctive Normal Form, as per section 5.2 of ‘Artificial Ematicians’. You may wish to use file REWRIT.
est Formulae for SKOLEM program

lan Bundy 23.6.81 /*

est1(Ans): -
    skolem( all(a, all(b, all(c, some(x, a*x^2 + b*x + c = 0))), Ans).

est2(Ans): -
    skolem( all(m, some(delta, all(x, (abs(x)<delta) -> (1/x)>m))), Ans).

90984 MATH_FORMUL 1K LISTED T15 LP15

* FORMUL.

90984 MATH_FORMUL 1K LISTED T15 LP15
unification procedure for first order logic (with occurs check).
see p80 of Artificial Mathematicians.

Ian Bundy 10.7.81

* Top Level */

unify(Exp1, Exp2, Subst) :- % To unify two expressions
    unify(Exp1, Exp2, true, Subst). % Start with empty substitution

* Unify with output and input substitutions */

unify(Exp, Exp, Subst, Subst). % If expressions
    unify(Exp1, Exp2, OldSubst, AnsSubst) :- % otherwise
        disagree(Exp1, Exp2, T1, T2), % find first disagreement pair
        make_pair(T1, T2, Pair), % make a substitution out of them, if pos
        combine(Pair, OldSubst, NewSubst), % find first disagreement pair
        subst(Pair, Exp1, NewExp1), % apply it to expressions
        subst(Pair, Exp2, NewExp2),
        unify(NewExp1, NewExp2, NewSubst, AnsSubst). % and recurse

* Find Disagreement Pair */

disagree(Exp1, Exp2, T1, T2) :- % otherwise
    Exp1 =.. [Sym1|_], Exp2 =.. [Sym2|_], % if exps have different
    Sym1 \== Sym2, !. % function symbol, then succeed

find_one(Args1, Args2, T1, T2) :-
    ind_one([Hd | T11], [Hd | T12], T1, T2).

* Find first disagreement pair in argument list */

ind_one([Hd | T11], [Hd | T12], T1, T2) :- !,
    % if heads are identical then
    find_one(T11, T12, T1, T2). % find disagreement in rest of list

ind_one([Hd1 | T11], [Hd2 | T12], T1, T2) :-
    disagree(Hd1, Hd2, T1, T2), !. % else find it in heads.

* Try to make substitution out of pair of terms */

make_pair(T1, T2, T1=T2) :- % T1=T2 is a suitable substitution
    is_variable(T1), % if T1 is a variable and
    free_of(T1, T2). % T2 is free of T1

make_pair(T1, T2, T2=T1) :- % or if T2 is a variable and
    is_variable(T2), % T2=T1 is a suitable substitution
    free_of(T1, T2). % T1 is free of T2
* By convention: x, y, z, u, v and w are the only variables */

s_variable(u).

s_variable(x).

s_variable(y).

is_variable(v).

is_variable(w).

is_variable(z).

* T is free of X */

ree_of(X,T) :- occ(X,T,0). % if X occurs 0 times in T

* Combine new substitution pair with old substitution */

combine(Pair, OldSubst, NewSubst) :-
    mapand(update(Pair), OldSubst, Subst1), % apply new pair to old subst
    check_overlap(Pair, Subst1, NewSubst). % and delete ambiguous assignments

* Apply new pair to old substitution */

update(Pair, Y=S, Y=S1) :- % apply new pair to rhs of old subst
    subst(Pair, S, S1).

* If X is bound to something already then ignore it */

k_overlap(X=T, Subst, Subst) :- % Ignore X=T
    memberchk(X=S, Subst), !. % if there already is an X=S

check_overlap(Pair, Subst, Pair & Subst). % otherwise don’t

* MINI-PROJECTS

Simplify unify to a one way matcher, as per p76 of ‘Artificial Mathematicians’.

Generalize Unify to a simultaneous unifier of a set of expressions, gen_unify) as per p80 of ‘Artificial Mathematicians’.

Build the associativity axiom into unify (assoc_unify), as per p82 of ‘Artificial Mathematicians’.
nl

90961 NL_XNL 1K LISTED T15 LP15

END
*generation of random sentences from an ATN*

Ian Bundy 19.11.79

**Top Level**

\[
generate :- \text{generate(sentence, start)}, \text{writef}{\n\n', [[]]}, \text{generate.}
\]

\[
rate(\text{ATN, finish}).
\]

\[
generate(\text{ATN, Node}) :- \text{arc(\text{ATN, Node, Choices}), random_pick(Choices, [\text{NewNode, Label, Type}]}, \text{gen(\text{Type, Label, ATN, NewNode}).}
\]

\[
\text{en(\text{atn, SubATN, ATN, NewNode}) :- generate(SubATN, start)}, \text{generate(\text{ATN, NewNode}).}
\]

\[
\text{en(\text{word, Word, ATN, NewNode}) :- \text{writef}{\%t ', [\text{Word}]}, \text{generate(\text{ATN, NewNode}).}
\]

**ATNs**

\[
\text{tence ATN}/**
\]

\[
\text{rc(sentence, start, [[a, nounphrase, atn]]}, \text{rc(sentence, a, [[b, verb, atn]]}, \text{rc(sentence, b, [[c, nounphrase, atn]]}, \text{rc(sentence, c, [[finish, stop_mark, atn]]}.
\]

**nournphrase ATN/**

\[
\text{rc(nounphrase, start, [[finish, proper_noun, atn], [a, determiner, atn]]}, \text{rc(nounphrase, a, [[finish, noun, atn], [a, adjective, atn]]}.
\]

**verb ATN/**

\[
\text{rc(verb, start, [[finish, is, word], [finish, kisses, word], [finish, kills, word]]}.
\]

**determiner ATN/**

\[
\text{rc(determiner, start, [[finish, a, word], [finish, an, word], [finish, the, word]]}.
\]

**noun ATN/**

\[
\text{rc(noun, start, [[finish, fascist, word], [finish, dictator, word]]}.
\]

**adjective ATN/**

\[
\text{rc(adjective, start, [[finish, happy, word], [finish, fascist, word], [finish, italian, word]}.
\]

**Proper Noun ATN/**

\[
\text{rc(proper_noun, start, [[finish, benito, word]]}.
\]
Stop Mark ATN*/
c(stop_mark, start, [[finish, (.), word], [finish, (?), word], [finish, (!), word]]).

70962 NL_ATN 2K LISTED T15 LP15

**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy

ALAN_BUNDY_HOPE_PARK_SQUARE
ALAN_BUNDY_HOPE_PARK_SQUARE
ALAN_BUNDY_HOPE_PARK_SQUARE
ALAN_BUNDY_HOPE_PARK_SQUARE
ALAN_BUNDY_HOPE_PARK_SQUARE
ALAN_BUNDY_HOPE_PARK_SQUARE
ALAN_BUNDY_HOPE_PARK_SQUARE
ALAN_BUNDY_HOPE_PARK_SQUARE
*generation of random sentences from an ATN
Ian Bundy 19.11.79
see with eliza (for print_list)*/

*Top Level*/
generate :- generate(sentence,start,Sentence), print_list(Sentence), fail.
generate(AN, finish, []).

generate(AN, Node, [Word | Rest]) :-
    arc(AN, Node, NewNode, Word, t),
    generate(AN, NewNode, Rest).

generate(AN, Node, Sentence) :-
    arc(AN, Node, NewNode, SubATN, nt),
    generate(SubATN, start, SubBit),
    generate(AN, NewNode, Rest),
    append(SubBit, Rest, Sentence).

/*ATNs*/

/*Sentence ATN*/
g(sentence, start, a, nounphrase, nt).
g(sentence, a, b, verb, nt).
g(sentence, b, c, nounphrase, nt).
g(sentence, c, finish, stop_mark, nt).

/*nounphrase ATN*/
g(nounphrase, start, a, determiner, nt).
g(nounphrase, a, finish, noun, nt).
g(nounphrase, a, a, adjective, nt).
g(nounphrase, start, finish, proper_noun, nt).

/*verb ATN*/
g(verb, start, finish, is, t).
g(verb, start, finish, kills, t).
g(verb, start, finish, kisses, t).

/*determiner ATN*/
g(determiner, start, finish, a, t).
g(determiner, start, finish, an, t).
g(determiner, start, finish, the, t).

/*noun ATN*/
g(noun, start, finish, fascist, t).
g(noun, start, finish, dictator, t).

/*adjective ATN*/
g(adjective, start, finish, happy, t).
g(adjective, start, finish, fascist, t).
g(adjective, start, finish, italian, t).
natural language conversational program
lan Bundy 15.11.79*/

*variable marker*/
- op(500, fx, []).

top level*/
liza :- read_in(Input), eliza(Input).
- a([bye, .]) :- !.
liza(Input) :- sr_pair(S, R),
    match(Input, S, off, Subst),
    replace(R, Subst, Output),
    print_list(Output), eliza.

*associative one way matcher*/
amatch([], [], off, []).
amatch([Word|RestSent], [Word|RestPatt], off, Subst) :-
    match(RestSent, RestPatt, off, Subst).
amatch([Word|RestSent], [?Var|RestPatt], off, [[Word|Phrase]/?Var/Subst]) :-
    match(RestSent, RestPatt, ?Var, [Phrase/?Var/Subst]).
amatch(Sent, Patt, ?Var, [[Phrase]/?Var/Subst]) :-
    match(Sent, Patt, off, Subst).hamatch(RestSent, RestPatt, ?Var, [[Phrase]/?Var/Subst]) :-
    match(RestSent, RestPatt, ?Var, [Phrase/?Var/Subst]).

*substitute text for variables*/
replace([], [], []).
replace([?Var|Rest], Subst, New) :-
    !, member(Phrase/?Var/Subst), replace(Rest, Subst, Old),
    append(Phrase, Old, New).
replace([Word|RestPatt], Subst, [Word|RestSent]) :-
    replace(RestPatt, Subst, RestSent).

*print list of words without brackets or commas*/
print_list([]) :- writef(\n \n, []).
print_list([Hd|Tl]) :- writef(\t , [Hd]), print_list(Tl).

*stimulus response table*/
r_pair([i, am, ?x, .], [how, long, have, you, been, ?x, ?]).r_pair([?x, you, ?y, me, .], [what, makes, you, think, i, ?y, you, ?]).
r_pair([?x, release, no, on, .]).
| **EMAS 2972 EMAS*** ECMIO2 A. Bundy | ALAN_BUNDY_HOPE_PARK_SQUARE |
| **EMAS 2972 EMAS*** ECMIO2 A. Bundy | ALAN_BUNDY_HOPE_PARK_SQUARE |
| **EMAS 2972 EMAS*** ECMIO2 A. Bundy | ALAN_BUNDY_HOPE_PARK_SQUARE |
| **EMAS 2972 EMAS*** ECMIO2 A. Bundy | ALAN_BUNDY_HOPE_PARK_SQUARE |
| **EMAS 2972 EMAS*** ECMIO2 A. Bundy | ALAN_BUNDY_HOPE_PARK_SQUARE |
| **EMAS 2972 EMAS*** ECMIO2 A. Bundy | ALAN_BUNDY_HOPE_PARK_SQUARE |
| **EMAS 2972 EMAS*** ECMIO2 A. Bundy | ALAN_BUNDY_HOPE_PARK_SQUARE |
| **EMAS 2972 EMAS*** ECMIO2 A. Bundy | ALAN_BUNDY_HOPE_PARK_SQUARE |
| **EMAS 2972 EMAS*** ECMIO2 A. Bundy | ALAN_BUNDY_HOPE_PARK_SQUARE |
Simple ELIZA

* a natural language misunderstander *

Alan Bundy 15-Nov-79, Richard O’Keefe 17-Feb-81

- [readin]. % load read_in(-) and its support.

public eliza/0. :- mode eliza.

liza :-

read_in(Input),
mandatory_substitutions(Input, Clean),
Clean \== [bye, ],
rule(Given, Yield),
match(Given, Clean),
reply(Yield), nl, !,
eliza.

liza :-

write(‘I hope I have helped you.’), nl.

- mode mandatory_substitutions(+, -), idiom(+, -).

mandatory_substitutions(Given, Yield) :-

idiom(Phrase, Translation),
append(Phrase, Rest_of_Given, Given),
append(Translation, Rest_of_Yield, Yield), !,
mandatory_substitutions(Rest_of_Given, Rest_of_Yield).

mandatory_substitutions([], []) :- !.

mandatory_substitutions([Word | Rest_of_Given], [Word | Rest_of_Yield]) :- !,
mandatory_substitutions(Rest_of_Given, Rest_of_Yield).

idiom([i], [you]).
idiom([me], [you]).
idiom([we], [you]).
idiom([us], [you]).
idiom([you], [i]).
idiom([my], [your]).
idiom([mine], [yours]).
idiom([our], [your]).
idiom([ours], [yours]).
idiom([your], [my]).
idiom([yours], [mine]).
idiom([am], [are]).
idiom([you, are], [i, am]).
idiom([stop], [bye]).
idiom([quit], [bye]).
idiom([goodbye], [bye]).
idiom([please], []).

- mode match(+, +), append(?, ?, ?).

match([Head | Rest], Fragment) :-

append(Head, Leftover, Fragment),
macht(Rest, Leftover).

match([Head | Rest], [Head | Leftover]) :- !.
match(List, Leftover).
\[\text{\textbf{match}}([]], [[]].\]

\text{\texttt{\textbf{match}}([], List, List).}
\text{\texttt{\textbf{claim}}([Head|Tail|, List, [Head|Rest]]) :- !, append(Tail, List, Rest).}
\text{\texttt{\textbf{- mode reply}(+).}}
\text{\texttt{\textbf{reply}}([Head|Tail]): -
\texttt{\textbf{reply}}(Head), !,
\texttt{\textbf{reply}}(Tail).
\texttt{\textbf{reply}}([]).
\texttt{\textbf{reply}}(Proper\_Word): -
\texttt{\textbf{write}}(', '),
\texttt{\textbf{write}}(Proper\_Word).
\texttt{\texttt{\textbf{rule}}([you, are, X, .]),
\texttt{\textbf{rule}}([X, i, Y, you, .]),
\texttt{\textbf{rule}}([you, like, Y, .]),
\texttt{\textbf{rule}}([you, feel, X, .]),
\texttt{\textbf{-}[X],
\texttt{\texttt{[how, long, have, you, been, X, ?]].
\texttt{[what, makes, you, think, i, Y, you, ?]].
\texttt{[does, anyone, else, in, your, family, like, Y, ?]].
\texttt{[do, you, often, feel, that, way, ?]].
\texttt{[please, go, on, .]].

90965 NL\_ELIZANEW 3K LISTED T15 LP15

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy

**EMAS 2972 EMAS** ECMIO2 A. Bundy
*parse*/
*acceptor and parser for sentences from ATN
lan Bundy 29.11.79
se with readin*/

*acceptor Top Level*/
accept :- read_in(Text), accept(Text).
accept(Text) :- accept_part(Text, sentence, start, []).

accept part of a sentence*/
accept_part([Word!Rest], ATN, start, Rest) :-
  arc(ATN, start, finish, Word, t).
accept_part(Text, ATN, Node, Rest) :-
  arc(ATN, Node, Next, SubATN, nt),
  accept_part(Text, SubATN, start, Text1),
  accept_part(Text1, ATN, Next, Rest).

*Parse top level*/
parse :- read_in(Text), parse(Text, Tree), print_tree(0, Tree).
parse(Text, Tree) :- parse_part(Text, sentence, start, [], Trees),
  Tree =.. [sentence|Trees].

*parse part of text*/
parse_part([Word!Rest], ATN, finish, Text, []) :- !.
parse_part([Word!Rest], ATN, start, Rest, [Word]) :-
  arc(ATN, start, finish, Word, t).
parse_part(Text, ATN, Node, Rest, [SubTree|Trees]) :-
  arc(ATN, Node, Next, SubATN, nt),
  parse_part(Text, SubATN, start, Text1, SubTrees),
  SubTree =.. [SubATN!SubTrees],
  parse_part(Text1, ATN, Next, Rest, Trees).

*print a syntax tree*/
print_tree(N, Tip) :-
  atomic(Tip), !,
  indent_write(N, Tip).
print_tree(N, Tree) :-
  Tree =.. [Node|SubTrees],
  indent_write(N, Node),
  N1 is N+1,
  checklist(print_tree(N1), SubTrees).
ident_write(N, Text) :-
    N5 is 5*N, tab(N5), write(Text), nl.

*ATNs*/

*Sentence ATN*/
rc(sentence, start, a, nounphrase, nt).
rc(sentence, a, b, verb, nt).
rc(sentence, b, c, nounphrase, nt).
rc(sentence, c, finish, stop_mark, nt).

*nounphrase ATN*/
rc(nounphrase, start, a, determiner, nt).
rc(nounphrase, a, finish, noun, nt).
rc(nounphrase, a, a, adjective, nt).
rc(nounphrase, start, finish, proper_noun, nt).

*verb ATN*/
rc(verb, start, finish, is, t).
rc(verb, start, finish, kills, t).
    (verb, start, finish, kisses, t).

*determiner ATN*/
rc(determiner, start, finish, a, t).
rc(determiner, start, finish, an, t).
rc(determiner, start, finish, the, t).

*noun ATN*/
rc(noun, start, finish, fascist, t).
rc(noun, start, finish, dictator, t).
rc(noun, start, finish, italian, t).

*adjective ATN*/
rc(adjective, start, finish, friendly, t).
rc(adjective, start, finish, misunderstood, t).
rc(adjective, start, finish, happy, t).
rc(adjective, start, finish, fascist, t).
rc(adjective, start, finish, italian, t).

oper Noun ATN*/
    .proper_noun, start, finish, benito, t).
rc(proper_noun, start, finish, sophie, t).

*Stop Mark ATN*/
rc(stop_mark, start, finish, (., t).
rc(stop_mark, start, finish, (?), t).
rc(stop_mark, start, finish, (!), t).
# Question Answering system based on Eliza
lan Bundy 16.11.79
se with readin, eliza and infer*/

# top level*/
a :- read_in(Input), qa(Input).

q[a(bye,[]) :- !.

Input) :-
    translate(Input,Desc,Type), answer(Type,Desc), qa.

# translation*/
translate(Input,Desc,Type) :-
    sm_triple(S,D,Type), match(Input,S,off,Subst),
    replace(Subst,D,Desc).

*replace variables by phrases in description*/
place(Subst,Word,Word) :- (atomic(Word); var(Word)), !.

place(Subst,?Var,Phrase) :-
    !, member(Phrase/Var,Subst).

place(Subst,D,Desc) :-
    D = .. [Sym!Args], not(Args = []), !,
    maplist(replace(Subst),Args,NewArgs),
    Desc = .. [SymiNewArgs].

*answer writer on basis of type and description*/
swer(assertion,Desc) :-
    assert_facts(Desc), writef('ok
', []).
swer(rule=assertion, (Ant->Consq)) :-
    assert(backward_rule(Ant,Consq)),
    writef('asserting rule %t implies %t \n', [Ant,Consq]),
    writef('ok
', []).
swer(yes/no-question,Desc) :-
    (infer(Desc) -> writef('yes
', []);
        writef('dont know
', []).
swer(wh-question,Desc) :-
    variables(Desc,Vars),
    (infer(Desc) -> checklist(print_list,Vars);
        writef('dont know
', []).

*stimulus meaning table*/

m_triple([all,?p,are,?q,], ([?p,X] -> [?q,X]),rule=assertion).
n_triple([who, is, a, ?p, ?q, ?], [?p, X] & [?q, X], wh-question).

n_triple([?t, is, ?p, .], [?p, ?t], assertion).

<table>
<thead>
<tr>
<th><strong>EMAS 2972 EMAS</strong>* ECMIO2 A. Bundy</th>
<th>ALAN_BUNDY_HOPE_PARK_SQUARE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EMAS 2972 EMAS</strong>* ECMIO2 A. Bundy</td>
<td>ALAN_BUNDY_HOPE_PARK_SQUARE</td>
</tr>
<tr>
<td><strong>EMAS 2972 EMAS</strong>* ECMIO2 A. Bundy</td>
<td>ALAN_BUNDY_HOPE_PARK_SQUARE</td>
</tr>
<tr>
<td><strong>EMAS 2972 EMAS</strong>* ECMIO2 A. Bundy</td>
<td>ALAN_BUNDY_HOPE_PARK_SQUARE</td>
</tr>
<tr>
<td><strong>EMAS 2972 EMAS</strong>* ECMIO2 A. Bundy</td>
<td>ALAN_BUNDY_HOPE_PARK_SQUARE</td>
</tr>
<tr>
<td><strong>EMAS 2972 EMAS</strong>* ECMIO2 A. Bundy</td>
<td>ALAN_BUNDY_HOPE_PARK_SQUARE</td>
</tr>
<tr>
<td><strong>EMAS 2972 EMAS</strong>* ECMIO2 A. Bundy</td>
<td>ALAN_BUNDY_HOPE_PARK_SQUARE</td>
</tr>
<tr>
<td><strong>EMAS 2972 EMAS</strong>* ECMIO2 A. Bundy</td>
<td>ALAN_BUNDY_HOPE_PARK_SQUARE</td>
</tr>
</tbody>
</table>
Synthesis of an exponentiation routine

-clp(700, x, f, x1, x2).

after test(X, U) :- X after test U.

loop(U, V, W) after loop(U, V, W).

X1 X2 after mult(R, R1, R2, X1, X2).

X1 - X2 after subtc(R, R1, X2, X1) :- const(X2).

*/

-zero(X) after test zero(R, X).

even(X) after test even(R, X).

est(X, U) needs C :- U needs C.

f(_, _, C, _) needs C.

loop(U, V, W) needs V < V0 & u: U & v: V & w: W.

ult(R, R1, R2, X1, X2) needs R1: X1 & R2: X2.

ubtc(R, R1, X2, X1) needs R1: X1.

ivc(R, R1, X2, X1) needs R1: X1.

zero(R, X) needs R: X.

even(R, X) needs R: X.

loop(_, _, _) affects u: _.

loop(_, _, _) affects v: _.

\ affects R: _ :- R: _ after U.

\ever \X & X.
\ever R: X & R: Y & X /= Y.
\always X /= X :! fail.
\always X /= Y.
\always X - 1 \X.
\always X/2 \X.

const(1).

const(2).

always loop(U, V, W, loop(U, V, W)).

loop(U, V, W, W) if zero(V).

loop(U, V, W, X) if \zero(V) & even(V) & loop(U * U, V/2, W, X).

loop(U, V, W, X) if \zero(V) & \even(V) & loop(U, V - 1, U * W, X).
Initially \( u: u_0 \).
Initially \( v: v_0 \).
Initially \( w: w_0 \).

\[
\textbf{ant loop}(u_0,v_0,w_0,x) & \land w:x.
\]

\[
90976 \text{ PLAN\_EXPON 2K LISTED T15 LP15}
\]

**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy

ALAN_BUNDY_HOPE_PARK_SQUARE
ALAN_BUNDY_HOPE_PARK_SQUARE
ALAN_BUNDY_HOPE_PARK_SQUARE
ALAN_BUNDY_HOPE_PARK_SQUARE
ALAN_BUNDY_HOPE_PARK_SQUARE
ALAN_BUNDY_HOPE_PARK_SQUARE
ALAN_BUNDY_HOPE_PARK_SQUARE
ALAN_BUNDY_HOPE_PARK_SQUARE
/* WARPLAN-C CONDITIONAL PLAN GENERATOR */

lans:-plan_from(start, _).

plan_from(TO,A):-
want C,
consistent(C,A),
plan(C, void, TO, P, T, A, []),
other_cases(T, P, A, exit).

(X1&X2, PO, TO, P, T, A, Q):-!,
plan(X1, PO, TO, P1, T1, A, Q),
plan(X2, P1, T1, P, T, A, Q).

plan(X, PO, TO, P, TO, A, Q):-
(always X, P=PO: holds(X, TO), setadd(X, PO, P)).

plan(X, _, _, _, _, A, _):= minimality_violation(X, A), !, fail.

plan(X, _, TO, _, _, _, _):-recorded(loopcheck, Type, _),
(Type=weak, mkground(X, O, _); Type=strong),
list1(X, TO, Q), !, fail.

plan(X, PO, TO, P, T, A, G):-
(X after U, U needs C, consistent(C, A),
 inform(try(X, U, TO)),
 extra_assumptions(U, A),
 achieve(X, U, PO, TO, T, A, [X:TO, .. G]),
 inform(got(X, U, T)),
P=(X&PO);

X if C,
consistent(C, PO&A),
plan(C, PO, TO, P, T, A, [X:TO, .. G]).

eve(X, U, PO, TO, (T; U), A, Q):-
reserved(PO, U),
U needs C,
consistent(C, PO&A),
plan(C, PO, TO, P, T, A, Q),
reserved(PO, U).

acheve(X, U, PO, (TO; V), (T; V), A, Q):-
reserved(X, V),
retrace(PO, V, P),
acheve(X, U, P, TO, T, A, Q),
reserved(X, V).

olds("X, (_; if(X, _, _, _))").
olds(X, (T; V)): -
(X after V, preserved(X, V), holds(X, T), preserved(X, V)).
olds(X, start): - initially X.

reserved(X&Y, V):-!, preserved(X, V), preserved(Y, V).
reserved(X, V).
extra_assumptions(test(X, U), A):-!, mkmember(X, A).
extra_assumptions(_, _).
inimality_violation(X,A):=-!, member(X,A).
inimality_violation(X,A):-member(X,A).

ther_cases((T;test(X,V)),P,A,W):-!
  negate(X,A,A1),
  retrace(P,test(X,V),P1),
  plan_from((T;if(X, V, P1, W)),A1).

ther_cases((T;V),P,A,W):-
  retrace(P,V,P1),
  other_cases(T,P1,A, (V;W)).

ther_cases(start,_,_,W):-
  write('----------'),nl,
  write_plan(W, 0),
  reply, fail.


etrace1(X1&X2, V, C, Y):-!
  etrace1(X1, V, C, Y1),
  etrace1(X2, V, C, Y2),
  union(Y1, Y2, Y).
  
  case(X,V,_, void):-X1 after V, X==X1, !.
  
  case(X1,\if(X1,_,_,_),_, void):-X==X1, !.
  etrace1(X,_,C,void):-member(X1, C),X==X1, !.
  etrace1(X,_,_,X).

consistent(C,P):-
  mkground (C&P, 0, _),
  never S,
  subset(S, C&P), !, fail.

consistent(_,_).

istel(X,[X|_]).

istel(X,[_,L|_]):=listel(X,L).

nion(void,X,X):-!.

nion(X, void,X):-!.

nion(X,Y&Y).

etadd(X,S,S):-member(X1, S), X==X1, !.

\ad(X,S,X&S).

ember(X,S):-\var(S), !, fail.

ember(X,S1&S2):-(!,(ember(X,S1);ember(X,S2))).

ember(X,X).

kmember(X,X&S):-!.

kmember(X,Y&S):-mkmember(X,S).

ubset(S1&S2,S):-!, subset(S1,S), subset(S2,S).

ubset(X,S):-member(X,S).

ubset(X,_,_):=always X.

egate(_,Y,_,_):=\var(Y), !, fail.

egate(X,Y&Z,Y1&Z):=negate(X, Y, Y1), !.

egate(X,Y&Z,Y&Z1):-!, negate(X, Z, Z1), !.

negate(X,X1,\X):-X==X1.

mkground('VAR'(N1),N1,N2):-!, N2 is N1+1.
mkground('VAR'(N),N1,N1):-!.
mkground(X,N1,N2):-X=.. [F.. A], mkgroundlist(A,N1,N2).
mkgroundlist([X,.. A1],N1,N3):-
  mkground(X,N1,N2),
mkgroundlist(A,N2,N3).
form(X):-debugon,!,write(X),nl,reply.

debug :-
    recorded(debug, on, _).

ndbug :-
    ( recorded(debug, _, P), !, erase(P); true),
    recorded(debug, on, _).

ffdebug :-
    ( recorded(debug, _, P), !, erase(P); true),
    recorded(debug, off, _).

eply :-
    repeat,
        ttygetO(C),
        obey(C), !.

bay(10):-!, ttyput(13).

(27):-!, offdebug.
(7):- abort.

rite_plan((if(_,U,_,W);W1),N):-!,
    tab(N),write(U),nl,
    N1 is N+1,
    write_plan(W1,N1),
    write_plan(W,N).

rite_plan((U,W),N):-
    tab(N),write(U),nl,
    write_plan(W,N).

rite_plan(exit,N):-
    tab(N),write(exit),nl.

977 PLAN_WPLANC 4K LISTED T15 LP15
90978 PLAN_WPO 1K LISTED T15 LP15

-op(950, xfx, [after, needs, affects, after_test, if]).
-op(950, fx, [always, never, initially, want]).
-op(900, xfy, &).
-op(700, xfx, :).
-op(600, fx, \).
90952 TEACH_INFER 1K LISTED T15 LP15

*infer*/
*LOGO type inference package
lan Bundy 16.11.79*/

*infer fact by backwards inference*/
infer(Goal) :- infer(Goal,now).
infer(Goal,Sit) :- fact(Goal,Sit).
infer(Goal,Sit) :-
backward_rule(Ant,Goal), infer(Ant,Sit).
infer(Goal1&Goal2,Sit) :-
infer(Goal1,Sit), infer(Goal2,Sit).

*assert fact and trigger forwards inference*/
assert_fact(Fact) :- assert_fact(Fact,now).
assert_fact(Fact,Sit) :-
assert(fact(Fact,Sit)),
writef('asserting %t in situation %t\n', [Fact,Sit]),
forall(forward_rule(Fact,New),assert_fact(New,Sit)).
*mandc*

**Missionaries and Cannibals program**

Alan Bundy 6.11.79*

*top level*/

mandc :- gofrom([3,3,1],[0,0,0],[]).

ofrom([0,0,0],Right,OldLefts) :- writef('which is goal state\n\n',[]).

ofrom(Left,Right,OldLefts) :-
  legal(Left), legal(Right),
  not(member(Left,OldLefts)),
  applymove(Left,Right,[ML,CL,BL],[MR,CR,BR]),
  writef('\nThe left bank now contains %t missionaries and %t cannibals \n',[]),
  writef('\nThe right bank now contains %t missionaries and %t cannibals \n',[]),
  gofrom([ML,CL,BL],[MR,CR,BR],[Left|OldLefts]).

ofrom(_,_,_):- writef('Backing up one move \n',[]), fail.

*decide which way to move*/

applymove([ML,CL,BL],Right,NewL,NewR):-
  (BL=1) -> moveload([ML,CL,BL],Right,NewL,NewR),
  moveload(Right,[ML,CL,BL],NewR,NewL).

*move boatload over*/

moveload(Source,Target,NewS,NewT):-
  move(BoatLoad), maplist(>=, Source, BoatLoad),
  mlmaplist(minus, [Source, BoatLoad, NewS]),
  mlmaplist(add, [Target, BoatLoad, NewT]),
  BoatLoad=[M,C,B],
  writef('\n move %t missionaries and %t cannibals \n',[]).

*the moves*/

move([0,1,1]).
move([2,0,1]).
move([0,2,1]).
move([1,1,1]).
move([1,0,1]).

*is situation legal*/

egal([M,C,B]) :-
  (M=0; M > = C), !.

egal([M,C,B]) :- writef('\n %t cannibals can eat %t missionaries \n',[],fail.

*add and minus are + and - on lists*/

add([X,Y,Z]) :- Z is X+Y.
minus([X,Y,Z]) :- Z is X-Y.
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
**EMAS 2972 EMAS*** ECMIO2 A. Bundy
ALAN_BUNDY_HOPE_PARK_SQUARE
ALAN_BUNDY_HOPE_PARK_SQUARE
ALAN_BUNDY_HOPE_PARK_SQUARE
ALAN_BUNDY_HOPE_PARK_SQUARE
ALAN_BUNDY_HOPE_PARK_SQUARE
ALAN_BUNDY_HOPE_PARK_SQUARE
ALAN_BUNDY_HOPE_PARK_SQUARE
ALAN_BUNDY_HOPE_PARK_SQUARE
ALAN_BUNDY_HOPE_PARK_SQUARE
90954 TEACH_READ 1K LISTED T15 LP15

*read file term by term*/
co :- see(mandc), repeat, read(T), write(T), nl,
    T=(:=end), seen.
* Read a sentence */

:- mode initread(-).
:- mode readrest(+,-).
:- mode word(-,?,?).
:- mode words(-,?,?).
:- mode alphanum(+,-).
:- mode alphanums(-,?,?).
:- mode digits(-,?,?).
:- mode digit(+).
:- mode lc(+,-).

. module(readin, [read_in(1)]).

* Read sentence */

read_in(P):-initread(L), words(P, L, []), !.

initread([K1,K2|U]):-get(K1), get0(K2), readrest(K2, U).

readrest(46, []):-!.
readrest(63, []):-!.
readrest(33, []):-!.
readrest(K, [K1|U]):-K<32, !, get(K1), readrest(K1, U).
readrest(K, [K2|U]):-get0(K2), readrest(K2, U).

ords([V|U]) --> word(V), !, blanks, words(U).
ords([]) --> [].

ord(U1) --> [K], Elc(K, K1)!, alphanums(U2), Ename(U1, [K1;U2])L.
\( 'nb(N)) --> [K], Edigit(K)!, digits(U), Ename(N, [K;U])L.
\( V) --> [K], Ename(V, [K])L.

lphanums([K1|U]) --> [K], Elphanum(K, K1)!, alphanums(U).
lphanums([]) --> [].

lphanum(K, K1):-lc(K, K1).
lphanum(K, K):-digit(K).

gits([K|U]) --> [K], Edigit(K), !, digits(U).
gits([]) --> [].

lanks --> [K], Ek<32L, !, blanks.
lanks --> [].

git(K):-K>47, K<58.
c(K, K1):-K>64, K<91, !, K1 is K\8'40.
c(K, K):-K>96, K<123.
*Random number generator*

*Get Num from Seed and update seed*/

```
random(Range, Num) :-
    seed(Seed),
    Num is (Seed mod Range) + 1,
    retract(seed(Seed)),
    NewSeed is 125*Seed+1,
    assert(seed(NewSeed)).
```

*random number seed*/

```
seed(13).
```

*Choose random element of list*/

```
random_pick(List, El) :-
    length(List, L), random(L, N), nth(N, List, El).
```

*Find nth element of a list*/

```
th(i, [Hd|T1], Hd) :- !.

nth(N, [Hd|T1], El) :-
    N>1, !, N1 is N-1,
    nth(N1, T1, El).
```
*random/\n*Random number generator
Alan Bundy 22.11.79*/

*Get random Num in Range*/
random(Range, Num) :-
    seed(Seed), random(Range, Seed, Num).

*Get Num ffrom Seed first time*/
random(Range, Seed, Num) :- Num is (Seed mod Range) + 1.

ow Num on subsequent occasions*/
random(Range, Seed, Num) :-
    NewSeed is 125*Seed + 1,
    random(Range, NewSeed, Num).

*random number seed*/
seed(13).

*Choose random element of list*/
random_pick(List, El) :-
    length(List, L), random(L, N), nth(N, List, El).

#find nth element of a list*/
th(1, [Hd|Tl], Hd) :- !.

th(N, [Hd|Tl], El) :-
    N>1, !, N1 is N-1,
    nth(N1, Tl, El).
90968 #_XWINST 1K LISTED T15 LP15

winst
rchprb
rch1prb
lockprb
solprb
airprb
inst
END

90968 #_XWINST 1K LISTED T15 LP15

**EMAS 2972 EMAS*** ECMI02 A. Bundy
**EMAS 2972 EMAS*** ECMI02 A. Bundy
**EMAS 2972 EMAS*** ECMI02 A. Bundy
**EMAS 2972 EMAS*** ECMI02 A. Bundy
**EMAS 2972 EMAS*** ECMI02 A. Bundy
**EMAS 2972 EMAS*** ECMI02 A. Bundy
**EMAS 2972 EMAS*** ECMI02 A. Bundy
**EMAS 2972 EMAS*** ECMI02 A. Bundy

ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE

ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE

ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE
ALAN_BUNDY HOPE PARK SQUARE

ST
ST
ST
ST
ST
ST
ST
ST

ST
ST
ST
ST
ST
ST
ST
ST

ST
ST
ST
ST
ST
ST
ST
ST

*arch.prb

inston arch domain
lan Bundy 5.12.80
se with winston */

* space of description trees */
pace(arch, [shapetree, touchtree, orienttree, directiontree, supporttree]).

* description tree */

r (shapetree, 1, shape(prism(wedge, block), pyramid)).

tree(touchtree, 2, touchrel(separate, touch(marries, abuts))).
default(touchtree, separate). % default predicate
tree(orienttree, 1, orientation(lying, standing)).
tree(directiontree, 2, direction(leftof, rightof)).
tree(supporttree, 2, undef(supports, unsupports)).

* Examples and near misses */

pecimen(arch1, [block(a), block(b), block(c),
standing(a), standing(b), lying(c),
leftof(a, b),
supports(a, c), supports(b, c),
maries(a, c), marries(b, c), marries(c, b)]).

pecimen(arch2, [block(a), block(b), wedge(c),
standing(a), standing(b), lying(c),
leftof(a, b),
supports(a, c), supports(b, c),
maries(a, c), marries(c, a), marries(b, c), marries(c, b)]).

pecimen(arch3, [block(a), block(b), block(c),
standing(a), standing(b), lying(c),
leftof(a, b),
supports(a, c), supports(b, c),
abuts(a, c), abuts(c, a), abuts(b, c), abuts(c, b)]).

pecimen(archn1, [block(a), block(b), block(c),
standing(a), standing(b), lying(c),
leftof(a, b),
supports(a, c), supports(b, c),
maries(a, c), marries(c, a), marries(b, c), marries(c, b),
maries(a, b), marries(b, a)]).

pecimen(archn2, [block(a), block(b), block(c),
standing(a), standing(b), lying(c),
leftof(a, b),
maries(a, c), marries(c, a), marries(b, c), marries(c, b)]).
standing(a), standing(b), lying(c),
leftof(a, b)]).

<table>
<thead>
<tr>
<th><strong>EMAS</strong></th>
<th>2972</th>
<th>EMAS***</th>
<th>ECMIO2</th>
<th>A. Bundy</th>
<th>ALAN_BUNDY HOPE PARK SQUARE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EMAS</strong></td>
<td>2972</td>
<td>EMAS***</td>
<td>ECMIO2</td>
<td>A. Bundy</td>
<td>ALAN_BUNDY HOPE PARK SQUARE</td>
</tr>
<tr>
<td><strong>EMAS</strong></td>
<td>2972</td>
<td>EMAS***</td>
<td>ECMIO2</td>
<td>A. Bundy</td>
<td>ALAN_BUNDY HOPE PARK SQUARE</td>
</tr>
<tr>
<td><strong>EMAS</strong></td>
<td>2972</td>
<td>EMAS***</td>
<td>ECMIO2</td>
<td>A. Bundy</td>
<td>ALAN_BUNDY HOPE PARK SQUARE</td>
</tr>
<tr>
<td><strong>EMAS</strong></td>
<td>2972</td>
<td>EMAS***</td>
<td>ECMIO2</td>
<td>A. Bundy</td>
<td>ALAN_BUNDY HOPE PARK SQUARE</td>
</tr>
<tr>
<td><strong>EMAS</strong></td>
<td>2972</td>
<td>EMAS***</td>
<td>ECMIO2</td>
<td>A. Bundy</td>
<td>ALAN_BUNDY HOPE PARK SQUARE</td>
</tr>
<tr>
<td><strong>EMAS</strong></td>
<td>2972</td>
<td>EMAS***</td>
<td>ECMIO2</td>
<td>A. Bundy</td>
<td>ALAN_BUNDY HOPE PARK SQUARE</td>
</tr>
<tr>
<td><strong>EMAS</strong></td>
<td>2972</td>
<td>EMAS***</td>
<td>ECMIO2</td>
<td>A. Bundy</td>
<td>ALAN_BUNDY HOPE PARK SQUARE</td>
</tr>
</tbody>
</table>
* description trees */

```prolog
ree(shapetree, 1, shape(prism(wedge, block), pyramid)).

r (touchtree, 2, touchrel(cseparate, touch(marries, abuts))).

ult(touchtree, separate). % default predicate

ree(orienttree, 1, orientation(lying, standing)).

ree(directiontree, 2, direction(leftof, rightof)).

ree(supporttree, 2, undef(supports, unsupports)).
```

* Examples and near misses */

```prolog
pecimen(arch1, [block(lp(a)), block(rp(a)), block(tm(a)),
    standing(lp(a)), standing(rp(a)), lying(tm(a)),
    leftof(lp(a), rp(a)),
    supports(lp(a), tm(a)), supports(rp(a), tm(a)),
    marries(lp(a), tm(a)), marries(rp(a), tm(a))].

pecimen(arch2, [block(lp(a)), block(rp(a)), wedge(tm(a)),
    standing(lp(a)), standing(rp(a)), lying(tm(a)),
    leftof(lp(a), rp(a)),
    supports(lp(a), tm(a)), supports(rp(a), tm(a)),
    marries(lp(a), tm(a)), marries(rp(a), tm(a))].

pecimen(arch3, [block(lp(a)), block(rp(a)), block(tm(a)),
    standing(lp(a)), standing(rp(a)), lying(tm(a)),
    leftof(lp(a), rp(a)),
    supports(lp(a), tm(a)), supports(rp(a), tm(a)),
    abuts(lp(a), tm(a)), abuts(rp(a), tm(a))].

pecimen(arch1n, [block(lp(a)), block(rp(a)), block(tm(a)),
    standing(lp(a)), standing(rp(a)), lying(tm(a)),
    leftof(lp(a), rp(a)),
    supports(lp(a), tm(a)), supports(rp(a), tm(a)),
    marries(lp(a), tm(a)), marries(rp(a), rp(a))].

pecimen(arch2n, [block(lp(a)), block(rp(a)), block(tm(a)),
    standing(lp(a)), standing(rp(a)), lying(tm(a)),
    leftof(lp(a), rp(a)),
    marries(lp(a), tm(a)), marries(rp(a), tm(a))].

pecimen(arch3n, [block(lp(a)), block(rp(a)), block(tm(a)),
    standing(lp(a)), standing(rp(a)), lying(tm(a)),
    leftof(lp(a), rp(a))].
```
**block.prb

inston block domain - simple test example
lan Bundy 6.12.80
se with winston */

* space of description tree(s) */
space(block, [shapetree]).

* description tree(s) */
tree(shapetree, 1, shape(prism(wedge, block), pyramid)).

_xample and near miss */
pecimen(block1, [block(a)]).
pecimen(wedge1, [wedge(b)]).


```
/* EMAS 2972 EMAS** ECMIO2 A. Bundy
   ALAN_BUNDY_HOPE_PARK_SQUARE */

/* EMAS 2972 EMAS** ECMIO2 A. Bundy
   ALAN_BUNDY_HOPE_PARK_SQUARE */

/* EMAS 2972 EMAS** ECMIO2 A. Bundy
   ALAN_BUNDY_HOPE_PARK_SQUARE */

/* EMAS 2972 EMAS** ECMIO2 A. Bundy
   ALAN_BUNDY_HOPE_PARK_SQUARE */

/* EMAS 2972 EMAS** ECMIO2 A. Bundy
   ALAN_BUNDY_HOPE_PARK_SQUARE */

/* EMAS 2972 EMAS** ECMIO2 A. Bundy
   ALAN_BUNDY_HOPE_PARK_SQUARE */

RECEI

isol.prb

definition of Isolation space and examples for Winston Program
Ian Bundy 18.2.81 */

* Predicate Trees */

res(occurtree,2, occur_rel(freeof, contains(singleocc, multocc))).
res(fault(occurtree, freeof)).

res(simtree,2, sim_rel(different(unrelated, inverse), ident)).
res(fault(simtree, unrelated)).

* Examples and near Misses */
pecimen(isol1, [singleocc(x, expr_at([1,2], before)),
     freeof(x, expr_at([1,1], before)),
     freeof(x, expr_at([2], before)),
     ident(expr_at([1,1], before), expr_at([2,1], after)),
     ident(expr_at([1,2], before), expr_at([1], after)),
     ident(expr_at([2], before), expr_at([2,2], after)),
     inverse(sim_at([1], before), sim([2], after)) ]).

pecimen(isol2, [singleocc(x, expr_at([1,2], before)),
     contains(x, expr_at([1,1], before)),
     freeof(x, expr_at([2], before)),
     ident(expr_at([1,1], before), expr_at([2,1], after)),
     ident(expr_at([1,2], before), expr_at([1], after)),
     ident(expr_at([2], before), expr_at([2,2], after)),
     inverse(sim_at([1], before), sim([2], after)) ]).

pecimen(isol3, [singleocc(x, expr_at([1,2], before)),
     freeof(x, expr_at([1,1], before)),
     contains(x, expr_at([2], before)),
     ident(expr_at([1,1], before), expr_at([2,1], after)),
     ident(expr_at([1,2], before), expr_at([1], after)),
     ident(expr_at([2], before), expr_at([2,2], after)),
     inverse(sim_at([1], before), sim([2], after)) ]).

pecimen(isol4, [singleocc(x, expr_at([1,2], before)),
     freeof(x, expr_at([1,1], before)),
     freeof(x, expr_at([2], before)),
     different(expr_at([1,1], before), expr_at([2,1], after)),
     ident(expr_at([1,2], before), expr_at([1], after)),
     ident(expr_at([2], before), expr_at([2,2], after)),
     inverse(sim_at([1], before), sim([2], after)) ]).

pecimen(isol5, [singleocc(x, expr_at([1,2], before)),
     freeof(x, expr_at([1,1], before)),
     freeof(x, expr_at([2], before)),
     ident(expr_at([1,1], before), expr_at([2,1], after)),
     different(expr_at([1,2], before), expr_at([1], after)),
     ident(expr_at([2], before), expr_at([2,2], after)),
     inverse(sim_at([1], before), sim([2], after)) ]).
```
pecimen(isol6, [singleocc(x, expr_at([1,2], before)), freeof(x, expr_at([1,1], before)), freeof(x, expr_at([2], before)), ident(expr_at([1,1], before), expr_at([2,1], after)), ident(expr_at([1,2], before), expr_at([1], after)), different(expr_at([2], before), expr_at([2,2], after)), inverse(sym_at([1], before), sym([2], after)) ]).

pecimen(isol7, [singleocc(x, expr_at([1,2], before)), freeof(x, expr_at([1,1], before)), freeof(x, expr_at([2], before)), ident(expr_at([1,1], before), expr_at([2,1], after)), ident(expr_at([1,2], before), expr_at([1], after)), ident(expr_at([2], before), expr_at([2,2], after)), unrelated(sym_at([1], before), sym([2], after)) ]).

pecimen(isol8, [multocc(x, expr_at([1,2], before)), freeof(x, expr_at([1,1], before)), freeof(x, expr_at([2], before)), ident(expr_at([1,1], before), expr_at([2,1], after)), ident(expr_at([1,2], before), expr_at([1], after)), ident(expr_at([2], before), expr_at([2,2], after)), inverse(sym_at([1], before), sym([2], after)) ]).

90972 **EMAS** 2972 EMAS*** ECMIO2 A. Bundy ALAN_BUNDY_HOPE_PARK_SQUARE

**EMAS** 2972 EMAS*** ECMIO2 A. Bundy ALAN_BUNDY_HOPE_PARK_SQUARE

**EMAS** 2972 EMAS*** ECMIO2 A. Bundy ALAN_BUNDY_HOPE_PARK_SQUARE

**EMAS** 2972 EMAS*** ECMIO2 A. Bundy ALAN_BUNDY_HOPE_PARK_SQUARE

**EMAS** 2972 EMAS*** ECMIO2 A. Bundy ALAN_BUNDY_HOPE_PARK_SQUARE

**EMAS** 2972 EMAS*** ECMIO2 A. Bundy ALAN_BUNDY_HOPE_PARK_SQUARE

**EMAS** 2972 EMAS*** ECMIO2 A. Bundy ALAN_BUNDY_HOPE_PARK_SQUARE

**EMAS** 2972 EMAS*** ECMIO2 A. Bundy ALAN_BUNDY_HOPE_PARK_SQUARE

**EMAS** 2972 EMAS*** ECMIO2 A. Bundy ALAN_BUNDY_HOPE_PARK_SQUARE

**EMAS** 2972 EMAS*** ECMIO2 A. Bundy ALAN_BUNDY_HOPE_PARK_SQUARE

**EMAS** 2972 EMAS*** ECMIO2 A. Bundy ALAN_BUNDY_HOPE_PARK_SQUARE
It is difficult for the untrained fisher to follow examples of this complexity so here is a simple concept: two wedges.

* space of description trees */
pace(pair, % each concept must have a space; is this **right**?
[shapetree, touchtree, orienttree]).

* description tree */

  (shapetree, 1, shape(wedge, block)).
ree(touchtree, 2, touchrel(separate, touch)).
ree(orienttree, 1, orientation(lying, standing)).

Examples
pecimen(p1, [wedge(a1), wedge(b1),
  standing(a1), lying(b1), separate(b1, a1)
]).
 pecimen(p2, [wedge(a2), wedge(b2),
  standing(a2), standing(b2), touch(a2, b2)
]).
 pecimen(p3, [wedge(a3), wedge(b3),
  lying(a3), lying(b3)
]).

Near misses
pecimen(n1, [block(a4), block(b4),
  standing(a4), lying(b4), separate(b4, a4)
]), % two similar things, but not wedges
pecimen(n2, [wedge(a5), wedge(b5), wedge(c5),
  standing(a5), standing(b5), touch(a5, c5)
]), % one wedge too many
pecimen(n3, [wedge(a6),
  standing(a6)
]), % one wedge too few
* winst
\nalional Reconstruction of Winston Learning Program
lan Bundy 1.12.80
erson for functions */

* Top Level Program — learn new concept */
* ------------------------------------- */

*First time only accept an example */
\ninst(Concept) :- !,
  writeln('Please give me an example of a %t \n', [Concept]),
  read(Ex), nl,
  make_rec(Concept,Ex,EObjs,ERec),
  maplist(gensym( plato),EObjs,CObjs),
  make_subst(EObjs,CObjs,Subst),
  subst(Subst,ERec,CRec),
  maplist(add_ups,CRec,CDefn),
  assert(definition(Concept,CObjs,CDefn)),
  winston1(Concept).

* Is grey area in definition eliminated? */
\ninsto(Concept) :- !,
  definition(Concept,CObjs,CDefn),
  checklist(same,CDefn), !,
  writeln('I have learnt the concept of %t now. \n', [Concept]).

*Subsequently accept either examples or near misses */
\ninstI(Concept) :- !,
  writeln('Please give me an example or near miss of a %t. \n', [Concept]),
  read(Ex), nl,
  writeln('Is this example (yes./no.)? \n', []),
  read(YesNo), nl,
  learn(Concept,Ex,YesNo),
  winston1(Concept).

* add default upper bounds in concept record */
\ndd_ups(record(Args,Name,Posn), define(Args,Name,[],Posn)).

* slight modify to gensym, so it can be used in maplist */
\nensym1(Prefix,_,NewConst) :- !, gensym(Prefix,NewConst).

* are upper and lower bound of concept definition the same? */
\n\name(define(Args,Name,Posn,Posn)).

* learn from this example or near miss */
\nlearn(Concept, Example, YesNo) :- !,
  definition(Concept, CObjs, CDefn),
  make_rec(Concept, Example, EObjs, ERec),
  classify(CObjs, EObjs, CDefn, ERec, Diff, Verdict),
  learnI(Concept, Diff, YesNo, Verdict).

* Make records from list of relations */
make_rec(Concept, Example, EObjs, ERec) :- !,
    specimen(Example, Relns),
    maplist(consts_in, Relns, CL), flatten(CL, EObjs),
    maplist(convert, Relns, ERec).

* Find all constants in terms */
onsts_in([], []).

onsts_in(N, []) :-
    integer(N), !.

onsts_in(Const, [Const]) :-
    atom(Const), !.

onsts_in(Exp, Consts) :-
    Exp =.. [Sym|Args], maplist(consts_in, Args, CL),
    flatten(CL, Consts).

*Flatten List */
latten([], []).

latten([Hd|T1], Ans) :-
    flatten(T1, Rest), union(Hd, Rest, Ans).

* Convert input relation style into internal representation as predicate tree */
convert(Reln, record(Args, Name, ExPosn)) :-
    Reln =.. [Pred|Args],
    length(Args, N),
    tree(Name, N, Tree),
    position(Pred, Tree, ExPosn).

* Find Position of Node in Tree */
osition(Node, Tree, []) :-
    Tree =.. [Node|SubTrees].

osition(Node, Tree, [N|Posn]) :-
    Tree =.. [Root|SubTrees],
    nth_el(N, SubTrees, SubTree),
    position(Node, SubTree, Posn).

* find nth element of list */
nth_el(1, [Hd|T1], Hd).

nth_el(N, [Hd|T1], El) :-
    nth_el(PN, T1, El), N is PN + 1.

* Is this example, non-example or in grey area, by my definition? */
* --------------------------------------------------------------- */
classify(CObjs, EObjs, CDefn, ERec, BestDiff, Verdict) :- !,
    findall(Diff, make_diff(CObjs, EObjs, CDefn, ERec, Diff), Diffs),
    best(Diffs, BestDiff),
    verdict(BestDiff, Verdict).

* Find the difference between example and concept */
make_diff(CObjs, EObjs, CDefn, ERec, Diff) :- !,
    perm(EObjs, EObjs1), make_subst(EObjs1, CObjs, Subst),
    subst(Subst, ERec, ERec1),
    pair_off(CDefn, ERec1, Diff).

*Pair off concept definition and example record to make differences */
pair_off([], [], []):- !.
maplist(new_defn, ERec, Diff).
sair_off(CDefn, [], Diff) :- !,
  maplist(extra_rec, CDefn, Diff).
sair_off([define(Args, Name, UpPosn, LowPosn) | CDefn],
  ERec,
  [differ(Args, Name, UpPosn, ExPosn, LowPosn, Verdict) | Diff]) :-
  select(record(Args, Name, ExPosn), ERec, Rest), !,
  compare(UpPosn, ExPosn, LowPosn, Verdict),
  pair_off(CDefn, Rest, Diff).

* invent new bits of definition as necessary *
new_defn(record(Args, Name, ExPosn),
  differ(Args, Name, [], ExPosn, DfPosn, Verdict)) :-
  default_posn(Name, DfPosn),
  compare([], ExPosn, DfPosn, Verdict).

* invent extra bits of example record as necessary *
extra_rec(define(Args, Name, UpPosn, LowPosn),
  differ(Args, Name, UpPosn, DfPosn, LowPosn, Verdict)) :-
  default_posn(Name, DfPosn),
  compare(UpPosn, DfPosn, LowPosn, Verdict).

* Find position of default predicate on tree */
default_posn(TreeName, Posn) :-
  default(TreeName, Pred), !,
  tree(TreeName, _, Tree), position(Pred, Tree, Posn).
default_posn(TreeName, []).

* Compare positions in tree to give verdict */
compare(U, E, L, yes) :- append(L, _, E), !.
compare(U, E, L, grey) :- append(U, _, E), !.
compare(U, E, L, no) :- !.

* Find best difference and return it */
est(Diffs, Diff) :- !,
  maplist(score, Diffs, Scores),
  lowest(Diffs, Scores, Diff, Score).

* Return difference with lowest score */
owest([], [Score], Diff, Score) :- !.
owest([Diff1 | Diffs], [Score1 | Scores], Diff2, Score2) :-
  lowest(Diffs, Scores, Diff2, Score2), Score2 < Score1, !.
ownest([Diff1 | Diffs], [Score1 | Scores], Diff, Score) :- !.

* Find score of difference */
core(Diff, Score) :- !,
  maplist(score1, Diff, Scores),
  sumlist(Scores, Score).

* Find score of individual differ */
core1(differ(_ _, _ _, _, yes), 0) :- !.
core1(differ(_ _, _ _, _, grey), 1) :- !.
core1(differ(_ _, _ _, _, no), 2) :- !.

* add up all the numbers in a list */
umlist([], 0).
umlist([N | Rest], Total) :- !,
  sumlist(Rest, SubT), Total is SubT + N.
* Make a substitution for replacing all members of one list by corresponding members of another list */
make_subst([X1], [Y1], X=Y & Subst) :
    make_subst([X1|XRest], [Y1|YRest], X=Y & Subst).

* Decide whether example falls inside definition on basis of differs */
derict(Diff, yes) :- checklist(verbict1(yes), Diff), !.
derict(Diff, no) :- some(verbict1(no), Diff), !.
derict(Diff, grey) :- some(verbict1(grey), Diff), !.

* verdict on individual differ */
derict1(V, differ(_, _, _, _, _, V)).

* adjust definition appropriately */
* ----------------------------------------------------------- */

* if new example found */
    nl(Concept, Diff, yes, grey) :- !,
        writeln('This is a new sort of X. \n', [Concept]),
        maplist(lub, Diff, New),
        retract(definition(Concept, COBJS, Old)),
        assert(definition(Concept, COBJS, New)).

* if near miss found */
earn1(Concept, Diff, no, grey) :- !,
    writeln('This limits my idea of X. \n', [Concept]),
    one_of(exclude, Diff, Diff1),
    maplist(diff_to_defn, Diff1, New),
    retract(definition(Concept, COBJS, Old)),
    assert(definition(Concept, COBJS, New)).

* if nothing new is discovered */
earn1(Concept, Diff, Agree, Agree) :- !,
    writeln('I have seen one of these before. \n', []).

* or if contradiction is discovered */
    nl(Concept, Diff, Agree, Disagree) :- !,
    writeln('Uh Oh, somethings gone wrong. I will think again. \n', []),
    fail.

* Move lower definition up a bit to include new example */
ub(differ(Args, Name, UpPosn, ExPosn, Old, grey),
    define(Args, Name, UpPosn, New)) :- !,
    common(ExPosn, Old, New).

* Lower definition already includes new example */
ub(differ(Args, Name, UpPosn, ExPosn, LowPosn, yes),
    define(Args, Name, UpPosn, LowPosn)) :- !.

* Move upper definition down a bit to exclude near miss */
xclude(differ(Args, Name, Old, ExPosn, LowPosn, grey),
    differ(Args, Name, New, ExPosn, LowPosn, grey)) :- !,
    common(ExPosn, LowPosn, Comm), append(Comm, [NJ], LowPosn),
    append(Comm, [NJ], New).

* Take unnecessary bits out of difference */
iff_to_defn(differ(Args, Name, UpPosn, ExPosn, LowPosn, Verdict),
    define(Args, Name, UpPosn, LowPosn)).

* Find common initial sublist of two lists */
common(Rest1, Rest2, Rest).

common(List1, List2, []): - !.

* change just one member of list */
ne_of(Prop, [Old|T1], [New|T1]): - apply(Prop, [Old, New]).
ne_of(Prop, [Hd|Old], [Hd|New]): - one_of(Prop, Old, New).

* Find out what grey areas still exist in concept */
rey(Concept): - !,
    writef('Grey areas in \%t are: \n', [Concept]),
    definition(Concept, CObjs, CDefn),
    checklist(grey1, CDefn).
rey1(define(Args, Name, Posn, Posn)): - !.
rey1(Defn): - !,
    write(Defn), nl.

90974 WINST_WINST 9K LISTED T15 LP15

**EMAS 2972 EMAS*** ECMIO2 A.Bundy ALAN_BUNDY HOPE PARK SQUARE
**EMAS 2972 EMAS*** ECMIO2 A.Bundy ALAN_BUNDY HOPE PARK SQUARE
**EMAS 2972 EMAS*** ECMIO2 A.Bundy ALAN_BUNDY HOPE PARK SQUARE
**EMAS 2972 EMAS*** ECMIO2 A.Bundy ALAN_BUNDY HOPE PARK SQUARE
**EMAS 2972 EMAS*** ECMIO2 A.Bundy ALAN_BUNDY HOPE PARK SQUARE
**EMAS 2972 EMAS*** ECMIO2 A.Bundy ALAN_BUNDY HOPE PARK SQUARE
**EMAS 2972 EMAS*** ECMIO2 A.Bundy ALAN_BUNDY HOPE PARK SQUARE
**EMAS 2972 EMAS*** ECMIO2 A.Bundy ALAN_BUNDY HOPE PARK SQUARE
The MAIL Command

This command allows EMAS users on machines attached to the RCONet to send each other text messages. The current version operates on the 2972 and 2980. Please send suggestions or comments to S. Shaw.

1 MAIL specification

1.1 Basic operations
1.2 Overview
1.3 Calling MAIL
1.4 Summary of commands
1.5 Addressing conventions
1.6 Message lists
1.7 Message components
1.8 Time-determined delivery
1.9 MAIL commands

Type the number of the section you want to view or the name of the MAIL command you want further information on.

Return from HELP back to MAIL by replying QUIT or a to the prompt 'View'.

1 TEXT
The MAIL command allows EMAS users on machines attached to the RCOnet to send each other text messages. The current version operates on the 2972 and 2980 but it is anticipated that it will become possible to exchange messages with other hosts on the network, notably the Computer Science Department VAX/VMS System. Communication with a large number of remote machines will eventually be possible via the PSS gateway.

Please send comments and suggestions to S. Shaw.

Contents

1.1 Basic operations
1.2 Overview
1.3 Calling MAIL
1.4 Summary of commands
1.5 Addressing conventions
1.6 Message lists
1.7 Message components
1.8 Time-determined delivery
1.9 Mail commands

1.1 Basic operations

MAIL provides a number of facilities for manipulating messages; however to get started, the subset described below should be sufficient.

1. Composing and sending a message

First invoke MAIL:

    Command MAIL
    Mail:

The prompt "Mail:" is issued whenever MAIL expects the next command to be specified. To send a message to "Bill Smith", first check on the correct form of the name to use; it may be "B. Smith" or "W. Smith" or "B. A. Smith" etc. - the asterisk is used to denote an arbitrary string of characters:

    MAIL: DIRECTORY #SMITH

<table>
<thead>
<tr>
<th>Rname</th>
<th>User</th>
<th>Host</th>
<th>Type</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>S. Smith</td>
<td>EYRE23</td>
<td>2972</td>
<td>S’name</td>
<td>Physics extn 1234</td>
</tr>
<tr>
<td>S. Smith</td>
<td>ERCC99</td>
<td>2980</td>
<td>S’name</td>
<td>ERCC extn 2642</td>
</tr>
</tbody>
</table>

Then use COMPOSE to create the draft message:

    MAIL: COMPOSE
    To:  B. Smith
    Subject: Test message
    Text:
    Here is a test message
If you are happy with the message, send it by replying "Y". If the reply "N" is given, then the contents of the draft can be amended before sending:

Mail: EDIT TEXT:

The draft may also be reviewed:

Mail: LIST DRAFT

Once the draft is satisfactory, use SEND to dispatch it (and file a copy of it):

Mail: SEND
Message sent and filed

The draft message may be filed without being sent:

Mail: FILE DRAFT

2. Receiving and displaying messages

You can receive messages into your process by use of the ACCEPT command:

Mail: ACCEPT

If there are any outstanding messages, they are taken into your message folder and a one line summary is printed for each. The program then asks if you want them listed:

Mail: ACCEPT
1 new message
in 4  (32)  B.Smith  Rs: Test message
List?

Reply "Y" or "N" to the prompt. No action is required to retain received messages; use DISCARD, described below, about deleting them.

3. Reviewing and deleting messages

A one line summary of each message held can be produced using SCAN:

Mail: SCAN ALL

The sequence number printed at the start of each one line entry can be used as a parameter to LIST or DISCARD, described below.

To review specific messages, find the required sequence number from the SCAN output, then use it with LIST; e.g., for message 3:

Mail: LIST 3

To get rid of unwanted messages use DISCARD:

Mail: DISCARD 3

To get rid of all currently held messages use the keyword ALL:

Mail: DISCARD ALL
subsequent call of TIDY it needed to purge them entirely;

MAIL: TIDY

On-line assistance is provided by HELP. Given no parameter it offers a table of contents; otherwise it lists information about a specified MAIL command. Return from viewing the help file back to MAIL by typing QUIT;

MAIL: HELP SCAN

View: QUIT

1.2 Overview

The MAIL command provides facilities for composing, sending, sending, receiving and storing messages. Messages are held in store map files called folders. All the correspondence which relates to one topic can be conveniently held in one folder.

In turn, messages themselves consist of a number of components. The body of the message consists of a text component, and the header of the message consists of all the other components, e.g.

Subject: Meeting on Wednesday, 2 p.m.
From: S Shaw
To: J Smith, J Jones

Please note the new location for our meeting is Room 2017,
3.

The process of composing a message entails placing text into the various components of the draft message (which is not held in a folder). For example, addresses go into the "To:" and "cc:" components and the body of the message goes into the "Text:" component. A draft message can be "^ at which causes its delivery to all the indicated recipients; each will receive a one-line TELL message indicating that they have outstanding mail. After Accepting messages, the user may selectively list them. Further manipulation of a message can involve Forwarding copies to additional recipients, Replying to its author, Filing for later reference or Discarding.

The messages in a folder are ordered chronologically and may be referenced by their index number (i.e. by their position in the folder), or by using special labels. At any given moment the system has a current folder and within it a current message which are under scrutiny; this avoids having to specify a folder and message index for every command.

A standard folder called MAINBOX is created by the system. When the MAIL command is invoked, the normal action is to Open the standard folder and select it as the current folder.

Only the draft message may be modified. However, any message in a folder can be Copied to the draft; similarly, the draft message can be Filed in a folder.

You are notified when a message has arrived for you by a TELL message from the executive process MAILER. At this stage the message has not been stored in your message - you must call the MAIL command and
1.3 Calling MAIL

The MAIL command can be called with no parameters - the program then issues the prompt 'Mail:' to indicate that it is expecting a MAIL directive to be given. This prompt is reissued after each directive has been performed until the directive STOP or QUIT is given:

```
Command: MAIL
Mail:
```

Alternatively, the name of a Mail directive may be specified in the call:

```
Command: MAIL(ACCEPT)
```

or

```
Command: MAIL(COMPOSE)
```

If a directive is to be given with parameters, the directive name should be followed by a comma:

```
Command: MAIL(LIST,ALL/.LP)
Mail:
```

The parameters for each MAIL command generally take the form `<input>/<output>`. If no `<output>` parameter is given then the slash (`/`) can be omitted.

A command name need not be typed out in full and in most cases may be abbreviated to a single letter. Where an ambiguous name is given, e.g. "F", then alphabetical order determines the command selected. Hence "F" will invoke "File"; "FD" will invoke "Forward". Lower case input is accepted throughout.

1.4 Summary of commands

Accept - take outstanding messages into the message folder
Accredit - add an alias R-name to the name/address directory
Compose - create a draft message and offer to SEND it
Copy - create components of the draft message
Directory - list an extract of the name/address directory
Discard - mark messages as discarded, or destroy draft components
Discredit - remove an R-name from the name/address directory
Ecc - invoke ECCE to edit a component of the draft message
Edit - invoke the standard editor to edit a component of the draft
File - copy messages to another folder, then discard the originals
Forward - package up an existing message for retransmission
Note - make the message specified the current message
Help - provide on-line information about MAIL commands
List - display messages on the console or list to a file
Next - list the next message in the folder on the console
Open - make the folder named the new current folder
Output - list a message component to a file, device or the console
Previous - list the previous message on the console
Quit - exit from MAIL and return to the Subsystem
Reply - create a draft message in reply to one received
Retrieve - remove "discarded" status from messages
Scan - produce a list of contents for the current folder
Send - package up a message and submit it for transmission
Stop - exit from MAIL and return to the Subsystem
Tidy - purge discarded messages from a folder

1.5 Addressing conventions
Messages are addressed to individuals by using their "recipient names" (R-names). Each R-name is unique and, for EMAS users, corresponds to the surname string associated with their EMAS process (the same string is printed on line printer output banners). Every EMAS user with a unique R-name has this name automatically entered into MAIL's name/address directory. This directory (the RCO directory) can be consulted using the DIRECTORY command. A user can have his surname string (and hence R-name) changed by making a request to the System Manager.

R-names take the form <name> at <directory>, e.g.

J. Jones at RCO

Hosts are regarded as having their own directories, containing usernumbers. Hence the following R-names are valid:

ERCC27 at 2972
ERCC66 2980

or RCO directory names, the "at RCO" (or "@RCO") part may be omitted.

When a specified R-name is being processed by MAIL, the case of the letters is ignored, as are spaces and dots. The following rules also apply:

- A list of recipients can be specified by using commas to separate R-names.
- Any text enclosed in parentheses "(...)" is ignored when analysing R-names.
- If an R-name contains matching angle brackets "<...>", any text outside the angle brackets is ignored.

The following address list shows these forms:

J. Jones at RCO (this text ignored),
ERCC27 2997 (usernumber addressing),
K. Jones <ERCC28 at 2980> (only text within <> taken as the address)

User can avoid having his surname string made known to other users as an R-name by setting permission NONE to the executive process MAILER:

Command: PERMIT(ALL,MAILER,N)

An arbitrary number of R-names may be associated with a given address. This may be convenient where several people share a process or where one person has several roles. Requesting additional R-names as aliases is provided as a user facility (see the command "Accredit", below).

1.6 Message lists

Within a folder, messages can be referenced by index number (position in the folder), and a collection of messages can be referenced at one time, by using commas and dashes as connectors. Hence the specification "1, 7-9, 90>89" refers to messages one, seven, eight, nine, ninety and eighty-nine. The angle bracket is like dash except that it indicates that the sub-list is in descending order.

Also, certain keywords define groups of messages, so that "new, 10-15, last" will reference all new messages, the tenth to fifteenth and the last message in the folder. The same message may appear more than once in such a list, but this does not imply any "repetition" of the message.
The terms defined below (with the exception of "draft") relate to messages in the current folder; they are as follows:

- `n` - messages `n` in the folder
- `n-a` - messages `n` to `a` (n less than a); if `n` is omitted, the value 1 is assumed. If `a` is omitted, the number of the last message is assumed.
- `n>a` - messages `n` to `a` (n greater than a); if `n` is omitted, the number of the last message is assumed. If `a` is omitted, the value 1 is assumed.

- `c=text` - where 'c' represents the name of a message component. This will select all messages in the folder whose 'c' component includes the indicated text. E.g., FROM=JONES. The text comparison ignores the case of the characters.

- `all` - all messages in the folder
- `current` - the message currently under scrutiny
- `discarded` - messages which have been discarded to a "wastebin" but which the janitor has not yet taken away (see the "Discard" command, below)
- `draft` - the single message which is currently being prepared; it is not held in any folder
- `last` - the last message in the folder
- `new` - messages which have been accepted in this MAIL session but which have not yet been LISTed
- `next` - the first message after the current one
- `old` - messages in the folder which are not discarded, new, unseen or saved
- `previous` - the first message preceding the current one
- `saved` - former draft messages which have been saved by being FILEd in the folder
- `unseen` - messages which were accepted in a previous MAIL session and have not yet been LISTed.

1.7 Message components

A message is composed of a series of components. The body of the message consists of a "Text" component - all the other components together constitute the header of the message. The following components are defined:

- **Date:** Indicates the date and time when the message was sent.
- **Subject:** Gives a brief indication of the content of the message and is displayed when the message is SCANNed.
- **From:** Indicates who sent the message. This field may be set by the
sending a message on someone else’s behalf) in this case, MAILER will add a "Sender:" component to the message to show who actually sent it.

**Sender:** Shows who actually sent the message and indicates that the "From:" component is not authentic.

**Reply to:** This allows the originator of the message to indicate where replies are to be sent.

**To:** The one or more primary recipients of the message.

**cc:** One or more secondary recipients (who receive carbon copies).

**bcc:** One or more tertiary recipients (who receive blind carbon copies, see below).

**Msg ID:** This holds a unique message identifier and is composed of three parts:

1) host name
2) a numeric identifier allocated by the server
3) date and time when the message was sent

**In reply to:** This component is added to a message by the REPLY command, and identifies the message being replied to.

**Comments:** This allows text comments to be added to the message without disturbing the contents of the message proper.

**After:** This component contains a date and time which determines when the message is to be delivered. See the section on "Time-determined delivery".

**References:** Not filled in or used by any MAIL function, this component may contain any text.

**Keywords:** Again, this component may contain any text and may be used to classify messages or to direct the operations of programs which automatically receive and manipulate messages.

**Folder:** Indicates the name of a folder in which the sender recommends the recipient to file the message.

When sending a message to a number of recipients, the sender may determine how much information each recipient receives about his co-recipients by selective use of the "To:", "cc:", and "bcc:" components.

**To:, cc:** A recipient in either of these lists has details only of the other "To:" and "cc:" recipients included in his copy of the message. The "bcc:" recipients are excluded.

**bcc:** A "bcc:" recipient will receive a copy of the message containing the "To:" and "cc:" recipients, and showing his own name alone in the "bcc:" component.

**Hence,** a message addressed

**To:** Black
**cc:** Brown
**bcc:** White, Grey

will be received by each as follows:
This feature may be useful where a distribution list containing many names is given - the sender can avoid burdening each recipient with a long list of names in which he has no interest.

Many of the commands of the MAIL program take a message component parameter. The full specification is:

```
<component name>;<message>(<folder name>)
```

For example:
```
e.g. cc:4(LETTERS)
```

The folder name can usually be omitted - by default the message referred to will be in the current folder. The `message` part of the specification may also be omitted, in which case the current message under scrutiny is assumed. Hence the message component specification "To!", refers to the "To!" component of the current message in the current folder.

Message component names can be abbreviated to a shorter form - in all cases, the first two characters of each component name give a unique abbreviation (hence "re:" is equivalent to "References!").

Note that a message keyword which may define more than one message can be used - the message selected is the first message found which matches the keyword. Hence "Subject:NEW" refers to the "Subject:" component of the first NEW message in the current folder.

1.8 Time-determined delivery

Messages may be sent with a "deliver after" specification which delays the delivery of the message until the current date and time is after the date and time specified. This is useful for sending reminders to yourself or others. The date specification may be a standard date (e.g., 15 July 1981), or a day of the week (MONDAY), or a keyword (TOMORROW). The time specification may be in hours and minutes (12:30) or mnemonic.

Dates are recognised in various forms, e.g., 15 July 81 or 15 7 81 or July 15 1981. Missing numbers are filled from today's date except in the case of a mnemonic month with no day following, when the first of the month is assumed.

Mnemonic dates are a weekday (Sunday, Monday etc.), Today, Tomorrow, Week, Month and Year. Weekdays are always in the future - if today's weekday is specified, the message is delivered in seven days time. "Week" specifies the next week (weeks start on a Sunday). "Month" specifies next month (which naturally starts on the first). "Year" specifies next year (starting 1 January).

The following mnemonic words are available: next, after, at, since.

Time specifications are in the form hh:mm on a 24 hour clock. Mnemonic times are available: Breakfast (06:00 am), Lunch (12:00), Noon (12:00), Tea (14:00 pm), Dinner (18:00 pm) and Midnight.
given in response to the "Send now?" prompt which follows the COMPOSE,
FORWARD and REPLY commands:

MAIL:SEND ,AFTER LUNCH TOMORROW
MAIL:SEND ,13 (indicates 13th of this month)

Send now? : NEXT WEEK (equals next Sunday)
Send now? : 10 AUG (10th August this year)

By default, a copy of all messages sent is filed in the current folder.
The copy may be directed elsewhere:

MAIL:SEND 4 (send message 4 and file a copy in the
current folder)
MAIL:SEND /F2 (send the draft and file a copy in folder F2)
MAIL:SEND 4,MONDAY/F3 (send message 4 for delivery next Monday and
file a copy in folder F3)

The "Send now?" prompt follows a call of COMPOSE, FORWARD or REPLY:

Send now? : Y (send and file a copy in the current folder)
Send now? : /F2 (send and file a copy in folder F2)
Send now? : NEXT WEEK/F3 (send message for delivery next week and file
a copy in folder F3)

To avoid filing a copy of the message in any folder, specify the dummy
folder .NULL:

MAIL:SEND .NULL

1.7 Mail commands

Contents

1.7.1 Accept
1.7.2 Accredit
1.7.3 Compose
1.7.4 Copy
1.7.5 Directory
1.7.6 Discard
1.7.7 Discredit
1.7.8 Edit
1.7.9 File
1.7.10 Forward
1.7.11 Boto
1.7.12 Help
1.7.13 List
1.7.14 Next
1.7.15 Open
1.7.16 Output
1.7.17 Previous
1.7.18 Quit
1.7.19 Reply
1.7.20 Retrieve
1.7.21 Scan
1.7.22 Send
1.7.23 Stop
1.7.24 Tidy
This command is used to accept messages sent to you by other users. By default, the messages are put in the current folder.

**Mail: ACCEPT**

If the `<folder name>` parameter is specified, then the messages are stored in that folder (it is made the current folder). The `<R-name>` parameter is required in two cases:

- where you want to accept mail directed to an alias R-name, e.g.
  
  `Mail: ACCEPT EMAS Suggestions/SUBBBX` (puts mail addressed to "EMAS Suggestions" into folder SUBBBX)

- where you are accepting mail within an EMAS process other than your own, e.g.
  
  `Mail: ACCEPT B Shaw`, `PASS`

After taking the outstanding messages, a SCAN of new messages (i.e., messages not yet LISTed) is performed, and you are offered the option of listing them.

1.7.2 Accredit

**Mail: ACCREDIT**

This command allows users to add additional R-names for themselves (aliases) to the name/address directory. This may be useful where several people share one EMAS process, or where one person has several roles. The facility should not be used to define names which will be meaningless to most MAIL users, such as nicknames known to only one or two people.

You may also set a password for the R-name which will make it possible to accept messages at another address (see ACCEPT). Passwords may be up to seven characters long.

It is also possible to set a "Department" field (31 characters) to be associated with the R-name. The information is displayed when a search is made of the name/address directory (see DIRECTORY) and is intended as an aid to distinguishing recipients with similar R-names.

Note that if you want to have your default R-name changed, this can only be done by application to the System Manager.

```
Mail: ACCREDIT
Rname: mail Suggestions
Password: ABCDE
Department: EMAS MAIL Suggestion Box
```

1.7.3 Compose

**Mail: COMPOSE `<component name>`**

This command offers a convenient way of creating a draft message. The draft is first cleared, then prompts are issued as indicated:

```
Mail: COMPOSE
To: .Jones
```
Text:
> Send 3/4d were going to a dance.
> Send now? : Y
Message sent and filed

Input for the "Text:" and "Comments:" components is terminated by a colon (:) or an asterisk (*) on a line by itself (the prompt issued on each line is "Y"). Alternatively, an EN character (control+Y) will terminate the input.

The input for all other components is normally terminated by a newline. Continuation lines are allowed and can be specified in two ways. If the line ends with a comma it is assumed that additional input follows. Alternatively, an explicit continuation character at the end of the line, backslash (\), allows a further line to be input. The backslash character itself is not included in the message component. Null input is accepted.

In addition to accepting text, the COMPOSE command will accept the contents of an EMAS file or of a component of an existing message. The escape character "\" must be used to indicate this form of input. You may request that prompts are issued for additional message components by giving the component names as parameters:

Mail:COMPOSE CC:
To: @ERCC27, NAMES
cc: J. Jones, R. Hill
Subject: @Subject:
Text:
@Text:4
Send now? : N
Mail:

In this example, a file is copied to the "To:" component of the draft, the "Subject:" component of the current message is copied to the "Subject:" component of the draft, and the "Text:" component of message 4 is copied to the "Text:" component of the draft.

Note that a message component, e.g. "@TEXT:" is distinguished from a file of the same name, "@TEXT", by the presence of a colon.

The response to the "Send now?" prompt may be "Y" or "N" or a date and time indicating delayed delivery (see Time-determined delivery). By default a copy of the message is filed in the current folder.

If the draft message is not sent (as in the example above), it may be modified further then dispatched using the SEND command.

1.7.4 Copy

Mail:COPY <input>/<component of the draft or DRAFT>

This command allows text to be copied to a component of the draft. Alternatively a complete message may be copied to the draft. If <input> is not specified then a prompt is issued and the text to be copied is read from the terminal. If the component of the draft is not specified, the "Text:" component is assumed. Alternative sources of input are an EMAS file or a component of an existing message.

Mail:COPY
Text: (No parameters, so input is prompted for and is taken to be for the "Text:" component of the draft message)
Mail: COPY CC: /TO:  (copies the "cc:" component of the current message to the "To:" component of the draft message)

Alternatively, a complete message (i.e., all its components) may be copied to the draft message. In this case the default for the first parameter is the current message.

Mail: COPY /DRAFT  (copies the current message to the draft)

Mail: COPY NEW/DRAFT  (copies the first new message in the current folder to the draft)

1.7.3 Directory

Mail: DIRECTORY <R-name> <mask> <output>

This command allows a search to be made of the name/address directory. The search may be for a specific R-name, or for all R-names that fit a mask. As in the Subsystem command FILES, the mask consists of up to three fields where a field is either a string of explicit characters or the symbol ",", representing any characters. Upper and lower case characters are not distinguished, and space and dot characters are ignored. Information is given on each R-name selected, under the following headings:

User   - the user number to which messages are delivered
Host   - the host on which the recipient is accredited
Type   - currently takes one of two values:
   R'name - the R-name is the standard process surname string, set by the System Manager
   Alias - the R-name was accredited by the user himself

Dept   - a user-defined field set by ACCREDIT

The <output> parameter may be null (implying output to the terminal), or a filename or device name. Examples:

Mail: DIRECTORY *show

Mail: DIRECTORY *MAC*/.LP

1.7.6 Discard

Mail: DISCARD <messages or draft components>

This command marks one or more messages in the current folder as being discarded, but does not physically remove them or re-number the remaining messages in the folder. The action is like placing a message in the wastebin - it is still available though less convenient to access, and is subject to permanent removal later by the TIDY command (see below). If no TIDY has been performed after a DISCARD then the RETRIEVE command can be used to recover the messages.
addition, for the draft only, individual components may be discarded (the component name must be followed by a colon). However, discarded components of the draft are destroyed immediately and cannot be recovered by RETRIEVE.

Mail: DISCARD 1-4, CC: SAVEd (discards messages 1-4 and all SAVED messages in the current folder, plus the "cc:" component of the draft)

Mail: DISCARD DRAFT (discards all components of the draft message)

The last message specified in the list becomes the current message if it is not the draft.

1.7.7 Discard

Mail: DISCARD

This command removes an R-name from the name/address directory. A password must be supplied if the command is called from any process other than that associated with the R-name.

DIScard is called as follows:

Mail: DISCARD
Rname: MAIL Suggestions
Password: ABCDE

1.7.8 Edit

Mail: ECCE <component or EMAS file>/<component of the draft>
Mail: EDIT <component or EMAS file>/<component of the draft>

This command allows an EMAS file or an existing component of any message file to be edited, and the result placed in a component of the draft.

If no parameters are given, then the "Text:" component of the draft message is edited. If an output draft component (i.e. one following "/") is given, then the existing contents of that component are overwritten. Note that a filename "CC" is distinguished from a message component "CC:" by the presence of a colon.

Mail: ECCE - edits the "Text:" component of the draft, creating it if none already exists

Mail: EDIT CC: - edits the "cc:" component of the draft, creating it if none exists.

Mail: ECCE Text:current/text: - edits the "Text:" component of the draft message to the "Text:" component of the draft.

Mail: EDIT /TD: - edits an empty file to the "To:" component of the draft.

Mail: EDIT MYLIST/TO: - edits an EMAS file to the "To:" component of the draft.

Mail: ECCE TR: - this will fail a component of the draft.
Mail:FILE <list of messages in the current folder>/<folder>

This command copies messages from the current folder to another folder, then discards the messages from the current folder. The input list of messages defaults to the current message. The draft message may also be filed (this is the only message which can be filed in the current folder); a filed draft message is given SAVED status (see SCAN).

The last message in the list of messages to be filed becomes the current message.

Mail:FILE <folder1> - files the current message to folder1
Mail:FILE DRAFT - files the draft to the current folder
Mail:FILE NEW, LAST/FOLDERS - files all new messages plus the first and last in the folder to folderb

1.7.10 Forward

Mail:FORWARD <messages>

This command sends a copy of one or more messages to another user or users. MAIL issues a prompt for the name of the user or users to whom you wish to forward the messages. You reply to this with the name(s) or alternatively specify an ERAS file or a message component which contains the names.

MAIL issues a further prompt, "Comments?", which allows you to add some text to the message without affecting the forwarded messages themselves.

As with COMPOSE and REPLY, you are offered the option of sending the message at the end of the operation. The last message in <messages> becomes the current message.

Mail:FORWARD 1
To: Rowland Hill
Comments:

Mail:FORWARD NEW
To: Rec
Comments:

- forward the first message in the current folder
- forward all new messages in the current folder to the list of recipients held in the "cc:" component of the current message.

1.7.11 Goto

Mail:GOTO <message>

The message specified becomes the current message. If a message keyword is used which may select more than one message, the first message found is selected.

Mail:GOTO 1 - go to the first message in the current folder
Mail: GOTO NEW, LAST - go to the first new message; if there are none, go to the last message in the current folder

1.7.12 Help

Mail: HELP <command>

This command provides information about the MAIL system and includes descriptions of all the MAIL commands. It operates by viewing a file containing the help text; hence the whole of this file can be explored at one time. If no parameter is given a table of contents is printed and further input requested. Alternatively, the name of a MAIL command may be given as a parameter.

Return from viewing the help text to MAIL using Q or QUIT.

Mail: HELP
    *
    View: QUIT

Mail: HELP COMPOSE
    *
    View: QUIT

1.7.13 List

Mail: LIST <messages>/<file or device>

This command displays messages in the current folder on the console, or alternatively lists to a file or device. In the latter case, a SCAN (see below) is prepended to the listing. If <messages> is omitted, the current message is listed.

Mail: LIST - displays the current message on the console

Mail: LIST NEW,DRAFT,LP - lists all new messages plus the draft to the line printer

Note that as a side effect of LIST, the status of a NEW message is changed to OLD.

1.7.14 Next

Mail: NEXT

This command lists on the console the first undispatched message after the current message. (Note the difference in meaning between this and the "next" message-reference keyword). The message listed becomes the current message.

1.7.15 Open
This command switches primary attention to another folder, i.e., makes the indicated folder the current folder. If the folder does not already exist it is created. If no parameter is given, the standard folder MAINBOX is made the current folder.

Mail:OPEN BUSS  - creates a new folder BUSS and makes it the folder BUSS created current folder

Mail:OPEN F2    - makes an existing folder F2 the current folder

Mail:OPEN      - makes MAINBOX the current folder.

If the parameter '?' is given, then the name of the current folder and the number of messages in it is printed:

Mail:OPEN ?    
Folder MAINBOX contains 16 messages, current message = 4

1.7.16 Output

Mail:OUTPUT <component>/<file or device>

This command is used to transfer a single component of a message to a file or device or to the console. The default component is the "Text:" component and the default message the current message.

Mail:OUTPUT    - displays the "Text:" component of the current message on the console

Mail:OUTPUT CC:4(F3) - displays the "cc:" component of message 4 in folder F3 on the console

Mail:OUTPUT /OBJECT1 - outputs the "Text:" component of the current message to a file

1.7.17 Previous

Mail:PREVIOUS

This command lists on the console the first undiscarded message prior to the current message. (Note the difference in meaning between this and the "previous" message-reference keyword). The message listed becomes the current message.

1.7.18 Quit

Mail:QUIT

Exits from MAIL and returns to Subsystem command level.

1.7.19 Reply
This command provides a convenient way of replying to a received message.

If no parameter is given then a reply to the current message is produced. A prompt is issued for the "Text:" of the reply. The escape character "\" can be used at this point to indicate input from an EMAS file or from an existing message component:

```
Mail:REPLY 2
Text:
:Your message received

Send now? : tomorrow
```

```
Mail:REPLY LAST(FOLDER3)
Text:
:BACK_UPDQITIE
Send now? : y
```

If a <message> in the current folder is specified, it is made the current message.

1.7.20 Retrieve

```
Mail:RETRIEVE <messages>
```

The complement of DISCARD, this command changes the status of the specified messages in the current folder from "discarded" to "old". Note that once discarded the draft message cannot be retrieved. If <messages> is null, the current message is retrieved.

The first message in <messages> becomes the current message

```
Mail:RETRIEVE DISCARDED
Mail:RETRIEVE 4,10-12
```

1.7.21 Scan

```
Mail:SCAN <messages>/<device or file>
```

This command scans the specified messages in the current folder and produces a "list of contents" - a series of one line summaries for the messages. By default, the current message is scanned and the output is directed to the console.

The format of the one line summaries is as follows:

```
status   - null = old message
         s = saved draft message (created when a
draft message is FILEd)
         x = discarded message
         # = the draft message
         n = new messages, i.e. messages not yet
             LISTED

index   - the index number of the message within the folder
`--- for the current message the indicator "<" is
```
Mail:Send

Mail:SEND <message>,<when>/<folder name>

This command packages up a message and submit it for transmission. If <message> is omitted, the draft message is sent.

The <when> parameter is used to indicate when the message is to be delivered (see Time-determined delivery). If this is omitted, the message is delivered immediately.

Mail:SEND ,MONDAY - sends the draft for delivery next Monday and files a copy in the current folder

Mail:SEND 1(STANDARD) - sends the first message in folder STANDARD and files a copy in the current folder

By default, a copy of the message is filed in the current folder. The <folder name> parameter is used to direct the copy to another folder. A dummy folder, NULL may be specified to indicate that no copy is to be filed:

Mail:SEND /N - sends the draft without filing a copy

1.9.23 Stop

Mail:STOP

Exits from the MAIL program and returns to Subsystem command level. STOP is identical to Mail:QUIT.

1.9.24 Tidy

Mail:TIDY <folder name>

This command causes discarded messages to be purged from the indicated folder (by default, the current folder). TIDY is irreversible.

The remaining messages in the folder are ordered by transmission date (in the case of SAVED draft messages, by date of filing), and hence the message index numbers change.

If the current folder is tidied, the current message becomes the first in the folder.
<table>
<thead>
<tr>
<th>Code</th>
<th>EAASxxx</th>
<th>ECMT25</th>
<th>P. Ross</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>972</td>
<td>972</td>
<td>972</td>
<td>972</td>
<td>LAWRENCE_BYRD_HOPE_PARK_SQUARE</td>
</tr>
</tbody>
</table>

**DYD480 THYMIOUT 41K LISTED T40 LF40**
EMAS Prolos for bluffers

This note is intended for AI teaching staff and explains how to set things up so that you can usefully run the EMAS Prolos system.

1) Obtain an EMAS 2972 account (contact Peter or Lawrence if stuck here).

2) Obtain adequate documentation for the things you will use. My current recommendations are the followings:

   a) EMAS 2900 information card (lists all the EMAS commands)

   b) EMAS 2900 Manual. This is a big book but you should have a copy for your shelf if you intend any serious use.

   c) ECCE manual. This is the text editor the students will be using, and is useful to know because of its wide availability.

   d) EMAS Prolos manual. This describes the Prolos system, how to use it, and what evaluable predicates are available.

   e) "Programming in Prolos". Bill and Chris' book. This is the only proper introduction to Prolos. The manual assumes you understand the Prolos language.

All the above documentation is available in both the HPS and FH DEC10 terminal rooms. This may serve your purposes.

3) Get access to EMAS. EMAS terminals are available in George Square and at KB. They offer the most reliable access and provide the right sort of VDU's if you want to use the screen editor available (see below). It is possible to set access to EMAS from DEC10 terminals via various networks, using GRETA and RCONET. See Lawrence for further details. The RCONET link should go into service on 1 October 81, but at the time of writing (10 September 81) the link is unusable due to standge hang-ups (problems reported, I am chasing them).

4) Set up your EMAS process (login account) properly. This should be done by issuing the following commands (just type the followings and ignore any error messages - I am assuming that you may not understand EMAS command calling conventions):

   Command: OPTION # BRACKETS

   Command: OBEY(ECMI25,SETUP)

This will initialise a reasonable environment for you. From a user level your world now works as follows:

Whenever you log on your terminal will be set up for lower case and will be assumed to be a VDU (for <RUBOUT> processing etc, if you use a hard copy device you may want to turn this off).

Remember that EMAS uses the following control-characters for terminal interaction:

ESC    interrupt (prompts for a string, Use A if confused)
DEL    rubout last character
^X      delete the whole line
^R      retypre the line
end of file

(However if you are linked through GRETNA etc then GRETNA will handle your line editing using standard DEC10 conventions. ESC will still be interrupt character, and ^Y must still be used for EOF).

Your commands are accepted in the NO BRACKETS convention. This means that you should separate your command from its arguments with a space and not put brackets around them.

The following useful commands are available (Some of these are just standard EMAS commands, some are extra that I have provided links to. I list them together here for convenience):

- Command: ecce <file> - edit <file> using ECCE
- Command: em <file> - edit <file> using EM
- Command: del <file> - delete <file>
- Command: dir - lists your directory (ie your files)
- Command: dir <file> - describes the file <file>
- Command: k - loss you off from EMAS (= stop)
- Command: list <file>,.lp - list <file> on line printer
- Command: mail - run the mailings system
- Command: prolos - runs Prolos
- Command: prolos <file> - runs Prolos restoring save state
- Command: roff .... - runs ROFF (see Lawrence for details)
- Command: s <file> - edits <file> using SCREED
- Command: stop - loss you off from EMAS
- Command: ty <file> - types out <file> onto terminal
- Command: view - runs the VIEW documentation system
- Command: viewprolos - runs VIEW starting at the Prolos docm

Some of these are just aliases provided for the benefit of DEC10 hardened users who are not used to EMAS. Suggestions for additions welcome. Note that not all these commands will necessarily be made available to students. This needs some thought.

5) You can now run Prolos, and using ECCE you have the minimal set of tools to start writing programs etc. There are commands available within Prolos for setting to ECCE and back, and for having files reconsulted when you return. See the EMAS Prolos Manual for details.

If you intend to use EMAS at all seriously then I recommend looking at the following additional software facilities - commands for which were listed above.
This is a tree structured information system. It contains various bits of information about the system. More importantly I am encouraging its use with students. The EMAS Prolog Manual is available in VIEW form and I intend to make other bits of information available in this way as they arise. If you have instructions for running software packages and so forth then it would be nice if they were in VIEW form. See me (Lawrence) about setting them integrated into a general tree about AI1/AI2 stuff.

This is a pretty decent mailing facility. I intend to accept scribes about Prolog etc. through it and it should be useful for communicating with AI2 students. This mailing system will eventually be linked up to other cross-network systems. I know that 2972 <-> CS-VAX is currently under test. I don't know whether 2972 <-> DEC10 will ever be feasible (probably not this year!).

This is a screen editor. Despite being rather clumsy it does work and is a reasonable way to enter text/programs. It is easy to learn - not having many commands. To use this you will probably have to access EMAS from George Square or KB, due to the need for a decent VDU. Use through GRETNA is undoubtedly somewhat infeasible.

For UNIX lovers there is a version of the EM editor available. This can be found knocking about in KNTLIB; it is available as a command once you have done the ECMI25,SETUP described above.

There is also a version of the (UNIX) ROFF text formatter. The EMAS Prolog Manual is formatted using this, and you may like to consider using it if you want to set up large amounts of student material. It should allow stuff prepared on the departments 11/60 to be moved and maintained under EMAS - but you may prefer to just move post-formatted material from the DEC10. I have a few roff macros available for formatting in SCRIBE-like ways if you are interested (they are not clever, in fact rather simple, however they do reduce the conceptual "change of gear" involved in returning to pre-SCRIBE days).
A.1 Glossary of built-in predicates available in EMAS Prolog

**abort**  Abort execution and return to top level.

**ars(N,T,A)**  The Nth argument of term T is A.

**assert(C)**  Assert clause C.

**assert(C,R)**  Assert clause C, and set reference R.

**asserta(C)**  Assert C as first clause.

**asserta(C,R)**  Assert C as first clause, and set reference R.

**assertz(C)**  Assert C as last clause.

**assertz(C,R)**  Assert C as last clause, and set reference R.

**atom(T)**  Term T is an atom.

**atomic(T)**  Term T is an atom or integer.

**basis(X,P,B)**  The basis of instances of X such that P is provable is B.

**break**  Break at the next procedure call into a recursive top-level.

**call(P)**  Execute the procedure call P.

**clause(P,Q)**  There is an clause in the program database with head P, body Q.

**clause(P,Q,R)**  There is an clause in the program database with head P, body Q, ref R.

**close(F)**  Close file F.

**consult(F)**  Read-in program clauses from the file F.

**current_atom(A)**  One of the currently defined atoms is A.

**current_functor(A,T)**  A current functor is named A, m.s. term T.

**current_predicate(A,P)**  A current predicate is named A, m.s. goal P.

**current_op(P,T,A)**  Atom A is an operator type T precedence P.

**debus**  Switch on debussing.

**debussings**  Output debussing status information.

**display(T)**  Display term T on the terminal.

**emac(C)**  Call the EMAS command C.

**emac(C,A)**  Call the EMAS command C with the atom A as the argument str

**erase(R)**  Erase the clause or record with reference R.

**exists(F)**  The file F exists.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expand_term(T,X)</td>
<td>Term T is a shorthand which expands to term X.</td>
</tr>
<tr>
<td>fail</td>
<td>Backtrack immediately.</td>
</tr>
<tr>
<td>fileerrors</td>
<td>Enable reporting of file errors.</td>
</tr>
<tr>
<td>functor(T,F,N)</td>
<td>The principal functor of term T has name F, arity N.</td>
</tr>
<tr>
<td>set(C)</td>
<td>The next non-blank character input is C.</td>
</tr>
<tr>
<td>set0(C)</td>
<td>The next character input is C.</td>
</tr>
<tr>
<td>halt</td>
<td>Halt Prolog; (exit to EMAS).</td>
</tr>
<tr>
<td>instance(R,T)</td>
<td>A m.s. instance of the record reference R is T.</td>
</tr>
<tr>
<td>inteser(T)</td>
<td>Term T is an integer.</td>
</tr>
<tr>
<td>Y is X</td>
<td>Y is the value of the integer expression X.</td>
</tr>
<tr>
<td>(\text{\textbackslash ash}(M))</td>
<td>Setashion mode for debuggings to M.</td>
</tr>
<tr>
<td>length(L,N)</td>
<td>The length of list L is N.</td>
</tr>
<tr>
<td>listings</td>
<td>List all the clauses in the current program database.</td>
</tr>
<tr>
<td>listings(P)</td>
<td>List the interpreted procedure(s) specified by P.</td>
</tr>
<tr>
<td>name(A,L)</td>
<td>The name of atom or integer A is strings L (list of ASCII codes).</td>
</tr>
<tr>
<td>nl</td>
<td>Output a new line.</td>
</tr>
<tr>
<td>nodebug</td>
<td>Switch off debuggings.</td>
</tr>
<tr>
<td>nofileerrors</td>
<td>Disable reporting of file errors.</td>
</tr>
<tr>
<td>nonvar(T)</td>
<td>Term T is a non-variable.</td>
</tr>
<tr>
<td>remove P</td>
<td>Remove spy-points from the procedure(s) specified by P.</td>
</tr>
<tr>
<td>`not P</td>
<td>Goal P is not provable.</td>
</tr>
<tr>
<td>numvars(T,M,N)</td>
<td>Number the variables in term T from M to N-1.</td>
</tr>
<tr>
<td>op(P,T,A)</td>
<td>Make atom A a syntactic operator of type T precedence P.</td>
</tr>
<tr>
<td>Phrase(P,L)</td>
<td>List L can be parsed as a phrase of type P.</td>
</tr>
<tr>
<td>portray(T)</td>
<td>Portray term T - NOT built-in but defined by user.</td>
</tr>
<tr>
<td>print(T)</td>
<td>Portray or else write the term T.</td>
</tr>
<tr>
<td>prompt(A,B)</td>
<td>Change the prompt from A to B.</td>
</tr>
<tr>
<td>put(C)</td>
<td>The next character output is C.</td>
</tr>
<tr>
<td>read(T)</td>
<td>Read term T.</td>
</tr>
<tr>
<td>reconsult(F)</td>
<td>Read-in program clauses from the file F; (replacing procedure</td>
</tr>
</tbody>
</table>
records(K,T,R) Make term T the first record under key K, reference R.
record(K,T,R) Term T is recorded under key K, reference R.
recordz(K,T,R) Make term T the last record under key K, reference R.
rename(F,G) Rename file F to G.
repeat Succeed repeatedly.
retract(C) Erase the first interpreted clause of form C.
save(S) Save the current state of Prolog in file S.
see(F) Make file F the current input stream.
seeing(F) The current input stream is named F.
seen Close the current input stream.
(P(C) Skip input characters until after character C.
spy P Set spy-points on the procedure(s) specified by P.
tab(N) Output N spaces.
tell(F) Make file F the current output stream.
telling(F) The current output stream is named F.
told Close the current output stream.
trace Switch on debugging and start tracing immediately.
true Succeed.
var(T) Term T is a variable.
write(T) Write the term T.
writea(T) Write the term T, quoting names where necessary.

'LC' The following Prolog text uses lower case.
'NOLC' The following Prolog text uses upper case only.
!
X < Y As integer values, X is less than Y.
X =< Y As integer values, X is less than or equal to Y.
X > Y As integer values, X is greater than Y.
X >= Y As integer values, X is greater than or equal to Y.
X =:= Y As integer values, X is equal to Y.
As integer values, $X$ is not equal to $Y$.

Terms $X$ and $Y$ are equal (i.e., unified).

The functor and arguments of term $T$ comprise the list $L$.

Terms $X$ and $Y$ are strictly identical.

Terms $X$ and $Y$ are not strictly identical.