AN INTRODUCTION TO MICRO-PROLOG

The MITSI Front End

Aston Academy
Aughton Road
Swallownest
Sheffield
S26 4SF

AN INTRODUCTION TO USING MICRO-PROLOG

THE MITSI FRONT END

BeebEm version

This application of micro-PROLOG is supplied on a single-sided disc, Prolog.ssd, and is for use on the Master128 version of the emulator.

Load the disc into drive 0 and execute Shift-F12 (Shift- Break) to load the MicroPROLOG ROM. Once the emulator has reset, type *PROLOG and press Enter.

From the MicroPROLOG command prompt, &, type LOAD MITSI and press Enter. It is essential that the command is typed in UPPER CASE. The MITSI front end, which is an educational environment in which to learn the basic skills of micro-PROLOG programming, will load after a few seconds and the prompt will change to >>>.

The MITSI front end uses a uniform syntax, meaningful variable names and a punctuation-driven command set. Lower case is used throughout.

LEAVING MITSI

To leave MITSI and PROLOG type:

quit!

You will be asked if you really want to quit to give you a chance to save any programs. Answer yes and press Enter to quit and you will be returned to BASIC in the emulator.

The emulator can be closed at any time but all PROLOG programs in memory will be lost.

THE STRUCTURE OF MITSI SENTENCES

MITSI can have two kinds of sentence;

(i) Facts,
(ii) Rules.

FACTS

Facts have the form

<object> <relation> <object>

An object can be either a single word (atom) or a list enclosed in brackets, eg.

John likes Mary
Sue parent_of (Adam Ethel Winston)

A Relation can be any meaningful word or even a phrase as long as the separate words are joined by underscore _ characters eg.

John likes Mary
Sue parent_of (Adam Ethel Winston)

RULES

These have the form

<sentence> if <condition> and <condition> and ...

where <sentence> and <condition> have the same structure as facts. Additionally <condition> can have the form

not <object> <relation> <object>

eg.

John likes Mary if Mary has_a computer

VARIABLES

MITSI variables can be any word starting with the prefix some or any. They can include the underscore character, eg.

someone, anybody, some_such.

ADDING FACTS OR RULES

To add a fact or rule to a MITSI database type the sentence and end it with a full stop, eg.

>>> John likes Mary.

>>> John is_happy if Mary likes computers.

ASKING QUESTIONS

Questions are asked by typing a question mark (?) at the end of the sentence. The computer will answer questions in two ways.

If there are no variables in the question, the computer will answer with YES or DON'T KNOW, depending on whether the question can be answered from the facts and rules in the database, eg.

```
>>> John likes Mary.    {fact sentence}
>>> John likes Mary?    {question }
YES                     {computer's reply}
```

If there are variables in the question the computer will answer YES and display the question with the variables filled in with matched facts, eg.

```
>>> John likes someone? {question }
YES John likes Mary
No (more) answers
```

If the question containing a variable has no answers the computer just displays the message No (more) answers, eg.

```
>>> Mary likes somebody?
No (more) answers
```

MORE ON QUESTIONS

MITSI includes a built-in why? question that will try to explain answers to questions. MITSI remembers the last sentence typed and, if this was a question, typing why? will cause this to be explained, eg.

```
>>> John likes Mary?
YES
>>> why?
John likes Mary is stated.
```

why? can also be used followed by a question. This can be used to ask for further explanation, eg.

```
>>> Fred likes Jim.
>>> Jim likes Fred.
>>> someone friend_of somebody
      if someone likes somebody and somebody likes someone.

>>> somebody friend_of somebody_else?
YES Fred friend_of Jim
YES Jim friend_of Fred
No (more) answers

>>> why Fred friend_of Jim?
Fred friend_of Jim because
      Fred likes Jim and
      Jim likes Fred
```

If there are many answers to a question, simply asking why? will explain the first answer only.

MITSI BUILT-IN PROGRAMS

Note that in these built-in programs the word linker is a hyphen or minus character, NOT AN UNDERSCORE.

sum-of
This is used to add two numbers together, check that two numbers when added produce a given result or to subtract two numbers. Only one of the numbers may be replaced by a variable.

Its syntax is

<number> sum-of (<number> <number>)

Check two numbers give a total

```
>>> 6 sum-of (4 2)?
YES
```

Add two numbers

```
>>> some_number sum-of (5 3)?
YES 8 sum-of (5 3)
No (more) answers
```

Subtract two numbers

```
>>> 7 sum-of (5 something)?
YES 7 sum-of (5 -2)
No (more) answers.
```

prod-of
This is used to multiply two numbers together, check that two numbers when multiplied produce a given result or to divide one number by another. Only one of the numbers may be replaced by a variable.

Its syntax is

```
<number> prod-of ( <number> <number> )
```

It is used in a similar manner to sum-of.

string-of
This is used to check that a list of characters make up a word, to convert from a list of characters to a word or to convert from a word to a list of characters.

Its syntax is

```
<list of characters> string-of <word>
```

Only one argument may be replaced by a variable.

Check a list of characters makes a word

```
>>> (Fred) string-of Fred?
>>> YES
```

Make a string of characters from a word

```
>>> some-list string-of Fred?
YES (Fred) string-of Fred
No (more) answers
```

Make a word from a string of characters

```
>>> (Mary) string-of some_word?
YES (Mary) string-of Mary
No (more) answers
```

less
This is used to check that one number is less than another or that one letter precedes another in the alphabet.

Its syntax is

<number> less <number> | <word> less <word>

eg.

>>> 4 less 7?
YES

>>> Michael less John?
DON'T KNOW

equals
This is used to check if two objects are the same. Only one of the arguments may be replaced by a variable.

Its syntax is

<object> equals <object>

eg.

>>> John equals John?
YES

>>> John equals Janet?
DON'T KNOW

belongs-to
This is used to check if an object is a member of a list of objects or to find the elements of a list.

Its syntax is

<object> belongs-to <list>

eg.

```
>>> John belongs-to (Fred Mary Sue John)?
YES

>>> someone belongs-to (Fred Mary Sue John)?
YES Fred belongs-to (Fred Mary Sue John)
YES Mary belongs-to (Fred Mary Sue John)
YES Sue belongs-to (Fred Mary Sue John)
YES John belongs-to (Fred Mary Sue John)
No (more) answers
```

appends-to
This is used to check that two lists can be joined together to give a third or to find two lists which when joined make a third.

Its syntax is

(<list> <list>) appends-to <list>

eg.

```
>>> ( (a b c) (d e f) ) appends-to (a b c d e f )?
YES


>>> some_names appends-to (Fred Mary Sue)?
YES ( () (Fred Mary Sue) ) appends-to (Fred Mary Sue)
YES ( (Fred) (Mary Sue) ) appends-to (Fred Mary Sue)
YES ( (Fred Mary) (Sue) ) appends-to (Fred Mary Sue)
YES ( (Fred Mary Sue) () ) appends-to (Fred Mary Sue)
No (more) answers
```