MacPROLOG User Guide

for LPA Demo MacPROLOG™ 3.5. with TPM 1.1[1]
as supplied with The Open University's course 'Intensive Prolog'

The version of MacPROLOG included here is a 1991 demo version. Being a demo version it has certain restrictions, but these do not affect anything you might need to do for this course. Please note that the advice in MacPROLOG's opening dialog-box, that you cannot save your object code, does not mean that you cannot save your programs as plain (i.e. uncompiled) text. Also included here is version 1.1 of the Transparent Prolog Machine, a programming environment designed to complement MacPROLOG itself, plus various examples and documentation including the TPM Reference Manual.

The software is provided free of charge, and you are legally free to use it for non profit-making educational purposes; please read the notice about copyrights and restrictions on use, on the cover of the TPM Reference Manual. You may also duplicate the master floppy and give copies to friends and colleagues, but please give away only complete copies. The expander, SitExpand, is public domain software.

Installation

The master floppy disk contains two compressed files, an expander, and some documentation for the expander. The documentation can be opened via Apple's Teach Text applicaiton or via any text editor. To obtain the TPM 1.1 application itself, first move all four files ifrom your master disk nto an empty folder on your hard disk, then double-click TPM 1.1.sit to run the expander.

The expansion will give you a folder named TPM 1.1 *ƒ* containing the following files and sub-folders:

Demo MacPROLOG 3.5
MacPROLOG Boot
Callgraph Boot
Graphics Boot
TPM 1.1
TPM preferences

MacPROLOG Demos
TPM READ ME + examples
TPM Equation solver example
TPM Memory allocation advice

TPM info
TPM Journal Article
TPM Reference Manual

Now double-click on OU Ex. & Ans.sit. The expansion of this will give you a folder containing:

* OU Read Me
*MAIN
Examples
Exercise
Videos

[1]LPA MacPROLOG™ Copyright © 1984-1991, Logic Programming Associates, Ltd. (UK)
TPM for Macintosh Copyright © 1991, The Open University (UK)
Apple Macintosh ™ is a registered trade mark of Apple Computer, Inc. (USA)

* OU Read me explains the organisation of the four folders.

Running MacPROLOG

Open TPM 1.1; this will also run MacPROLOG 3.5 and load the necessary files for you to work in the TPM environment.[2] Your screen should now show a window as in Fig 1.

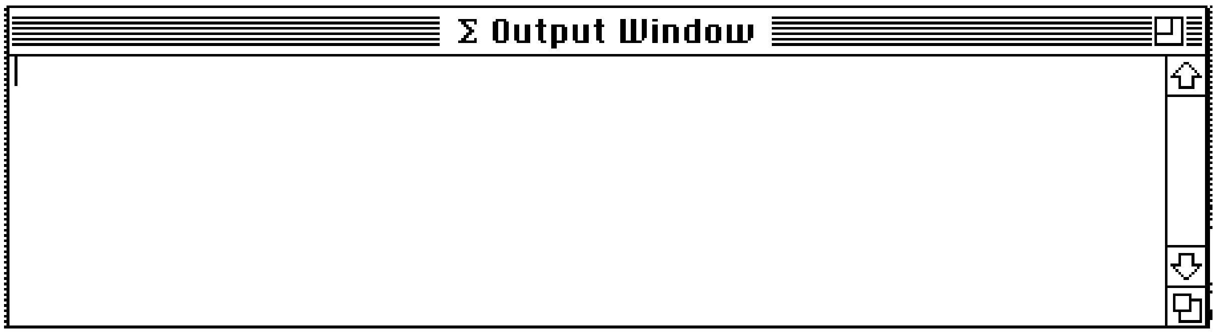


Fig. 1

If all is well you can if you wish now delete from your hard disk the four files copied from your master TPM 1.1 floppy.

What you have on screen is a perfectly normal Prolog environment, augmented by TPM's program visualisation and debugging tools. The visible window is MacPROLOG's output window. In order to use the system, you would now either create a new code window and start writing a program in it, or you would load a file containing some previously-written code. the creation of new code windows will be covered later; the following tutorial uses code which you have already installed from your master TPM 1.1 disk.

Please note that, regrettably, this demo version of MacPROLOG does not include its Help file. Therefore the 'Help...' choice in the File menu won't give you any help.

Getting started with MacPROLOG

Choose 'Open...' from the File menu and open the text file 'airline.pl' which can be found in the *MAIN folder. 'Airline.pl' contains a database of Prolog facts and rules. Notice that the very first item in the Airline program database is the fact

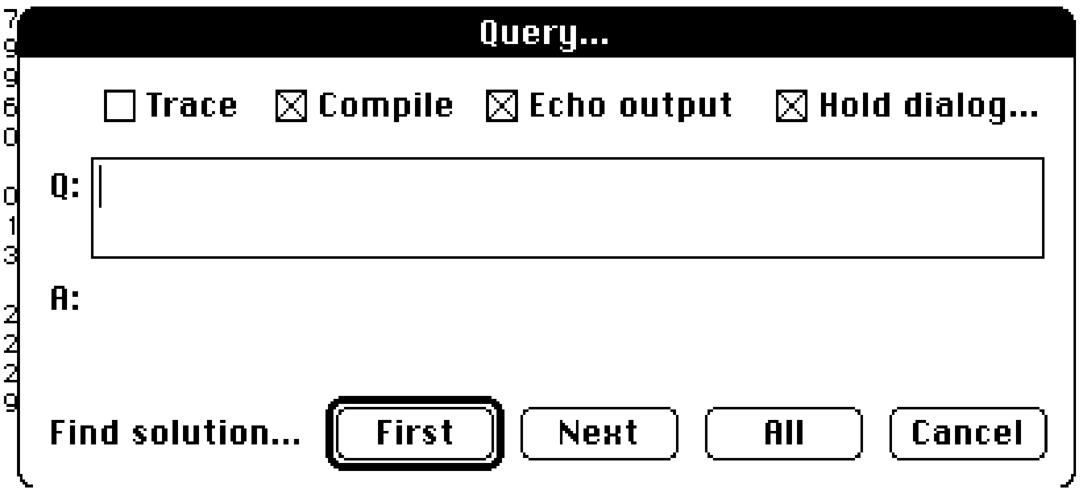
```
origin(ba137, chicago).
```

Your first practical exercise in entering queries into Prolog is going to be to insert the query

```
origin(ba137, Where).
```

Here we go, then. Choose 'Query...' from the Eval menu. Almost immediately you will see the Query Box as in Fig. 2, containing a flashing cursor which urges you to type something. Type into it the above query. Be sure to get the syntax exactly right - you can actually omit the final full stop, but other than in the query box Prolog is very intolerant of this, so it's as well not to get the habit now!

[2]If you should ever wish to run MacPROLOG without TPM, simply open Demo MacPROLOG 3.5 instead of TPM 1.1. The 'shift-click' method suggested on the disk label is necessary only if you have more than one version of MacPROLOG on your hard disk. The version of Demo MacPROLOG 3.5 supplied by us has been specially tuned to work with TPM.



☐ Trace ☒ Compile ☒ Echo output ☒ Hold dialog...

Fig. 2

When you're happy with what you've typed, click on the button beneath your query marked 'First'. If you have after all made a syntax error Prolog will tell you so, but otherwise it will respond immediately with the reply that **Where = chicago**. The reply appears both in the Query Box itself and in the output window.

Now try a variation on the above. Click on the Query Box to reselect it, then choose 'All' instead of 'First'. If you're watching closely, you will see the same reply flash briefly in the Query Box, and then the message 'No more solutions'. However, both the reply and the message will be printed in the output window. Any output resulting from your query will be printed to the output window, and similarly if your program contained instructions to print partial results or verbal messages, the printouts would also appear in the output window.

There is an important point to mention about the Query Box. Your query remains in it until you decide to remove or change it. Even if you cancel the Query Box and then choose 'Query...' from the Eval menu again, your query will still be there. When developing a program and hence needing to repeat the same query, perhaps with minor modifications, over and over again, this fact can be very helpful.

You'll notice if you try cancelling the Query Box that when you bring it back your query is already highlighted, so if you wanted to enter a completely new query you could simply start typing, rather than having to delete the old one first.

To reinforce these points, get the query box and change your query to

```
origin(Flight,london).
```

This time, the database contains three facts which can match that query. If you now click on 'First' in the Query Box you will get the first of these three answers as before. Now click on 'Next', and you'll get the second answer. Finally, click on 'Next' again and you'll get the third. If you check in the output window you'll see that all three solutions are there. Notice that 'All' prints all three solutions to the output window in one operation. The terms 'First', 'Second' etc. refer to the order in the database of the facts which match the query.

MacPROLOG special features

In other implementations of Prolog you the programmer are not forced to group facts and rules of the same name all together in the database as they are grouped in 'airline.pl', but it's good mental discipline anyway and MacPROLOG does enforce it. A similar convention is that the underscore character as in has_flight is used where in English one would use a hyphen. This is because Prolog treats the hyphen as a minus-sign.

MacPROLOG can be interrupted by the user during execution and listing by pressing the <Command> and <full stop> keys simultaneously. (Use the Command key like a shift key and then hit the full stop.) This is useful for interrupting long printouts or endless loops.

The original OU Prolog minus operator '~' (tilde) referred to in the course text reverts back to an ordinary '-' (minus), as in other Prolog dialects. Thus, adding 4 to -3 can be performed by entering the query:

```
X is 4 + (-3).
```

MacPROLOG's response is:

```
X = 1
```

The division operator '/' now performs real division, whereas the special operator '//' is used for integer division. For example the query

```
X is 5/2.
```

results in

```
X = 2.5
```

and the query

```
Y is 5//2.
```

gives

```
Y = 2
```

How TPM can help

Verbal explanations of how Prolog interprets queries do not suit everyone. More importantly, they can become impossibly complex when trying to describe in detail anything other than a very simple program. The moments when this becomes important are, of course, when you're trying to develop a more sophisticated program and find that it has bugs in it. The difficulty is to have a means of describing to yourself what is going on, so that you can see where the errors arise. Until recently Prolog (in common with other high-level languages) had only a 'verbal' debugging system: you could place 'spy points' on the various facts and rules in your program, and as it was subsequently executed you would see on screen 'sentences' telling you when the interpreter 'called' each of them and whether it then succeeded or failed.[3]

The Transparent Prolog Machine offers a more approachable, because graphical, explanation of Prolog execution. The way it works is that you enter your query via the TPM menu rather than via the Eval menu, which enables TPM to collect debugging information as your program is executed. You then have as it were a videotape of the interpreter's behaviour, which you run forwards or backwards, step through, and/or investigate in more detail.

Try this: choose 'Traced Query' from the TPM menu and enter the query **has_film(ba137)**. Make sure to ask only for the first solution. Execution will take longer than via the Eval menu's query box, because a lot more than merely execution is going on behind the scenes; but the Mac's 'wristwatch' cursor will reassure you that nothing has gone wrong. TPM will give you a

[3]There is a rather neat version of Spy Points included with MacPROLOG. But you will almost certainly prefer TPM.

new window containing a single graphical node, as in Fig. 3. (You may have to use the horizontal and/or vertical scrollbars in the window to find it; and if you find two nodes rather than one it is probably because you clicked on 'All' solutions!)

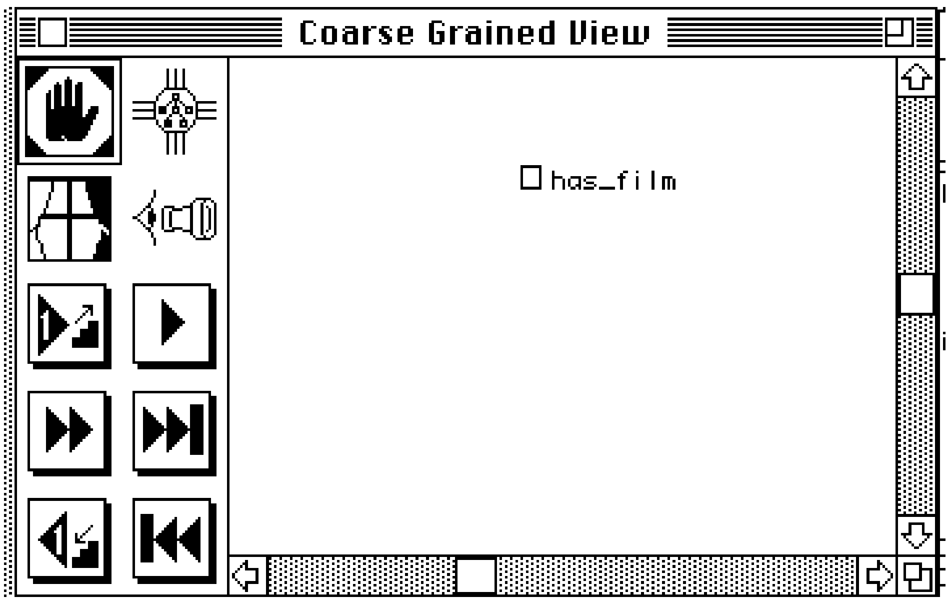


Fig. 3

The TPM Manual will probably be useful here: on page 8, section 3.1, it describes what the various symbols used to indicate nodes mean. (By the way 'procedure' and 'rule' are interchangeable terms.) From it, you'll be able to see that the single node in your TPM window is in the 'success' state. In other words your query succeeded, which you already know.

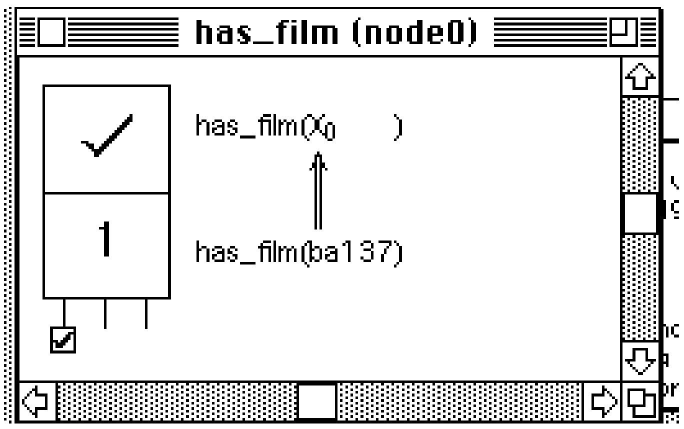


Fig. 4

Choose the 'zoom in on node' tool from the TPM window (it looks like a jeweller's eyeglass). The cursor will change to a magnifying-glass. Click on the 'has_film' node, whereupon a new window will appear (Fig. 4) named **has_film (node 0)**. This new window contains the AORTA diagram for the 'has_film' node. It is known as the Fine Grained View. AORTA diagrams are fully explained in the course; the suggestions here are only to give you a 'feel' for using TPM.

As already mentioned, the current Coarse Grained View shows the final state of the interpreter after executing your query. So click on the CGV window to select it, and then click on the 'to start' button, which resembles two leftward arrowheads pointing to a vertical bar. The two windows should now appear as in Fig. 5, to represent the start of execution; more precisely, to represent the state of the interpreter just before you pressed the 'First' button in the Traced Query Box.

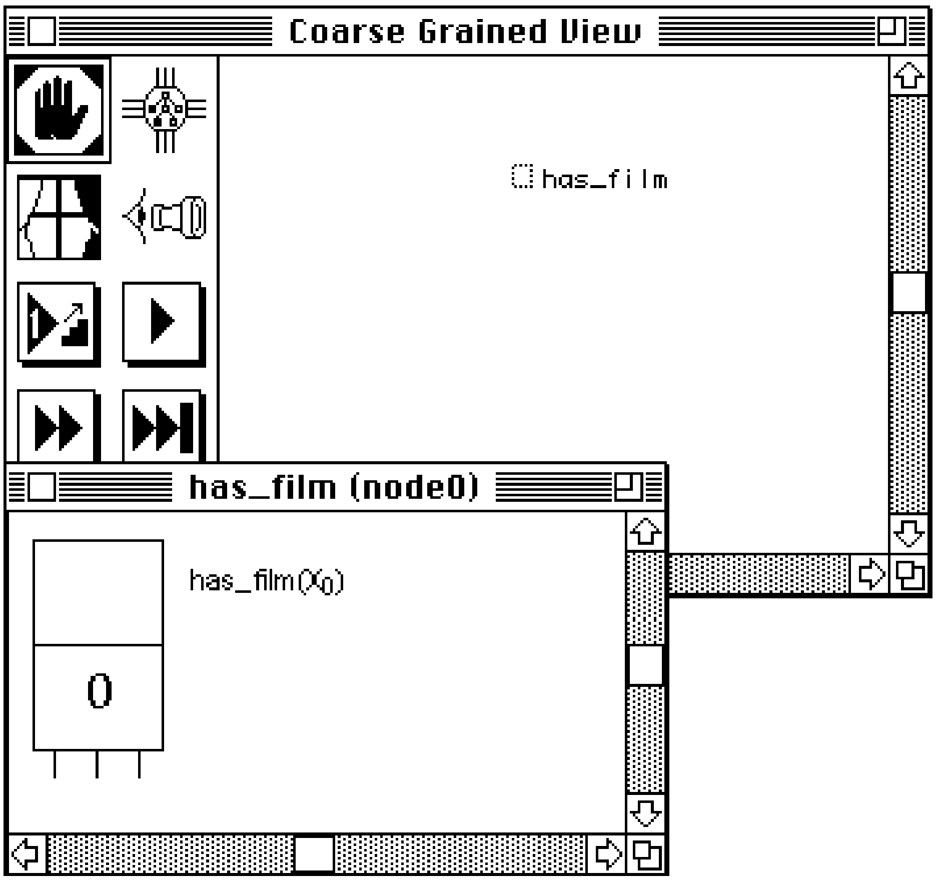


Fig. 5

Now use the Step button (a flight of stairs with an arrow pointing up them and a larger black arrow indicating 'forward by one') to step through the execution of your query. Watch carefully, because every time the FGV is updated it becomes the selected window, and you have to give an extra click to select the CGV again before you an operate its Step button. So the number of clicks isn't directly related to the number of steps in your program! There are three steps in the evaluation of the query **has_film(X)**.

The FGV contains a box with an upper and a lower part. The upper part represents a query (or subgoal), and the lower part indicates a matching database fact or rule-head. The upper part of the box is filled in, as execution proceeds, with information as to the success or failure of the query or subgoal (page 11 of the TPM Manual). Alongside each is a verbal reminder of the query or subgoal itself and the corresponding database entry, with variable/value assignments indicated by arrows.

It is worth mentioning here that if you decide to use the Step Back button (the mirror image of the Step button), all of the text will disappear from the FGV, and won't reappear even if you step forwards again. You have to use the 'to start' button and go forwards from there to see the text. This apparent awkwardness is explained on page 10 of the TPM Reference Manual.

Once having convinced yourself that TPM really is describing what it should be describing (and remember that its stored information is like a video, so that if at any point you want to start again, just hit the 'to start' button), you might like to try more complicated exercises, such as repeating the existing query but asking for all solutions, and entering as a TPM Traced Query **has_film(X)**, asking for first one and then all solutions. The display resulting from the latter is shown in Fig. 6).

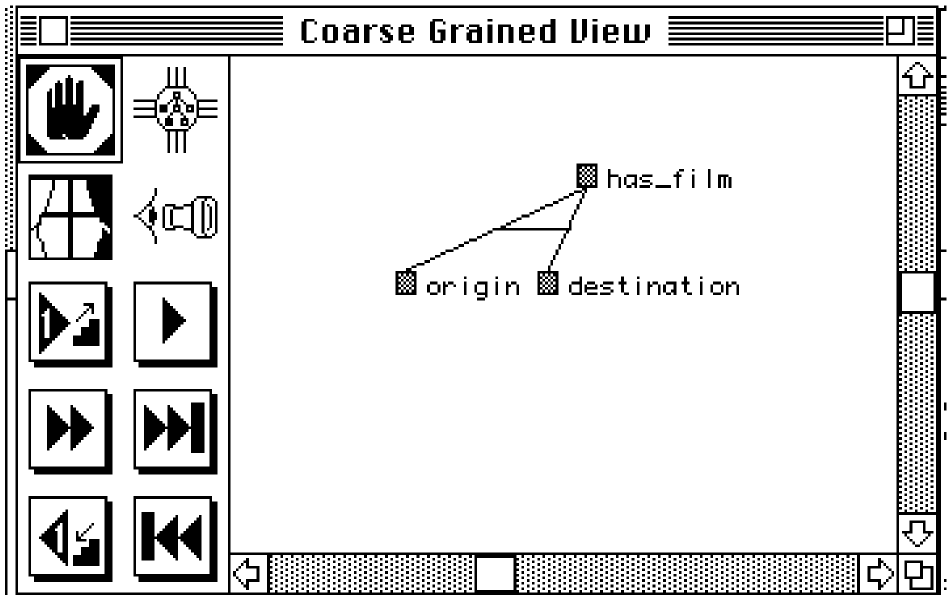


Fig. 6

You'll find if you're willing to spend a few moments exploring TPM in this way that is an excellent way of explaining how Prolog programs work, as well as being a potentially powerful debugging tool. Later on, as you come to use it in earnest to help you develop and debug 'real' programs, you'll discover from its manual that it is a good deal more powerful than has been described here.

Using the system in a real situation

This has been a quick tour of the TPM/MacPROLOG programming environment. But there will come a point when you will want to use it to develop your own programs. This section will give a description of how to create and manipulate program files and windows in MacPROLOG. Please have MacPROLOG running but with no files open; if you still have 'airline.pl' open as above, choose 'Close...' from the File menu and when a dialog-box appears click on its 'Close' button. Your screen should again look as in Fig. 1.

MacPROLOG refers to its document files on disk as 'source files'. A source file would normally contain a single program, or a set of utility routines, etc. Each source file must contain at least one program window, similar to the window which contained the Airline database above. It can contain more than one, but if it doesn't contain one it doesn't contain anything!

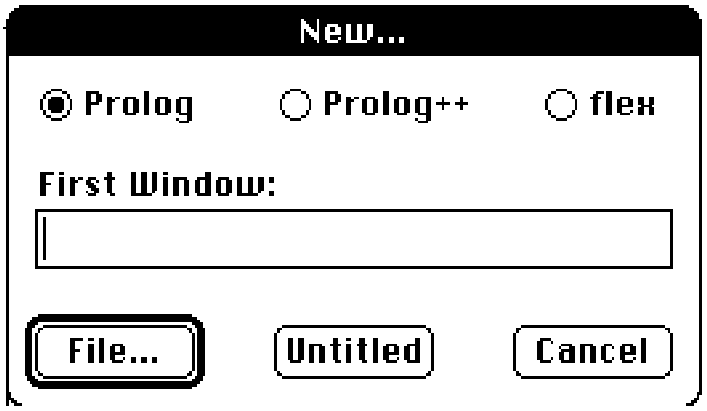
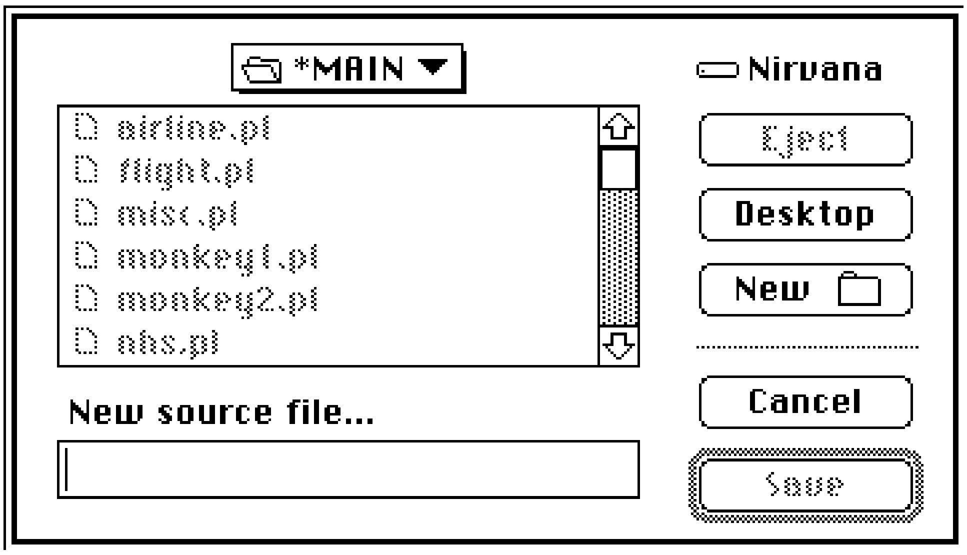


Fig. 7

In order to create a new source file, choose 'New...' from the File menu. The dialog-box which appears (Fig. 7) is asking you for the name of the first, essential, window in your new file, and *not* for the name of the file itself. So enter a window name, and then press the 'File' button. This will bring up another dialog-box (Fig. 8), which resembles a normal Save box. Enter a file name and hit the 'Save' button. Your screen should now lcontain a new window with the name you gave it.

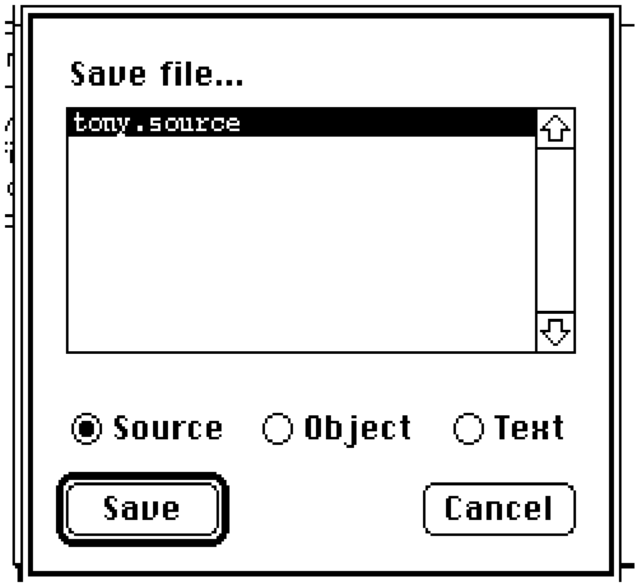
You could at this point start typing in your program. But for the sake of this quick tour please type into your new window any legal Prolog statement such as **some(prolog).**



New source file...

Fig. 8

Imagine that now you wanted to save the program you are writing. Choose 'Save file...' from the File menu. Again you'll get a dialog-box, as in Fig. 9. Click its 'Save' button. Your program is now saved. (Please remember, if you're doing this along with me, that in the process you'll have created on your hard disk a file which you'll later want to throw away!)



Save file...

tony.source

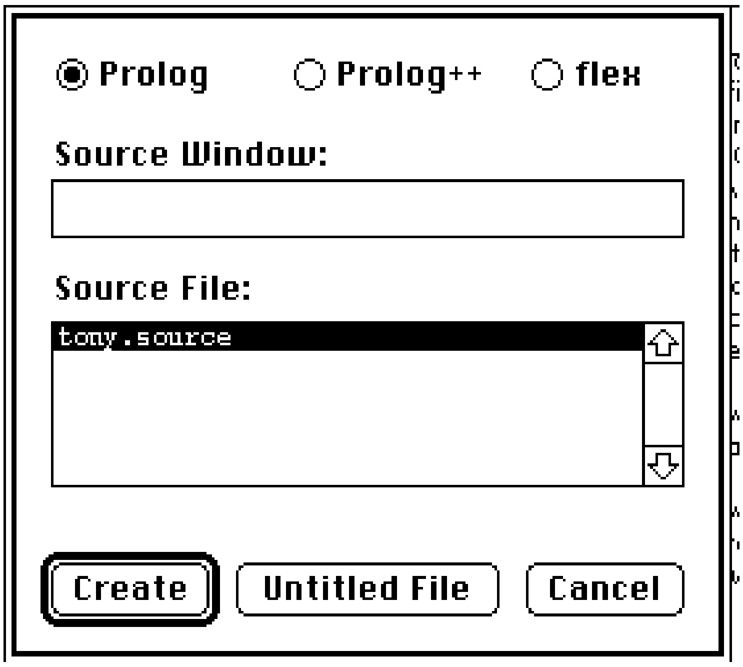
(● Source ○ Object ○ Text)

Fig. 9

Many implementations of Prolog require all program code to be written in one continuous file, and of course this is not always convenient. MacPROLOG provides two mechanisms for modularising your code. First, any source file can consist of more than one code window. To create a new window, choose 'Create...' from the Windows menu. All you need to type into it i(Fig. 10) s the name of your new window; then hit the 'Create' button. Your new window will appear and will automatically become part of your source file when you subsequently save the latter.

(Your new window will be set to Geneva 12pt text, whereas the code in the course material is written in Monaco 9pt. You can reset your window's font and fontsize via the Font menu, or if you prefer you can arrange for all new windows to come up in Monaco 9pt by choosing 'Defaults...' from the File menu.)

The second mechanism is that you are allowed to have more than one source file, each with its own windows, on screen at the same time. You simply open the second file as you opened 'airline.pl'. Any code in any on-screen window can be used as part of your program.



[✓ Prolog] Prolog++ flex

Source Window:

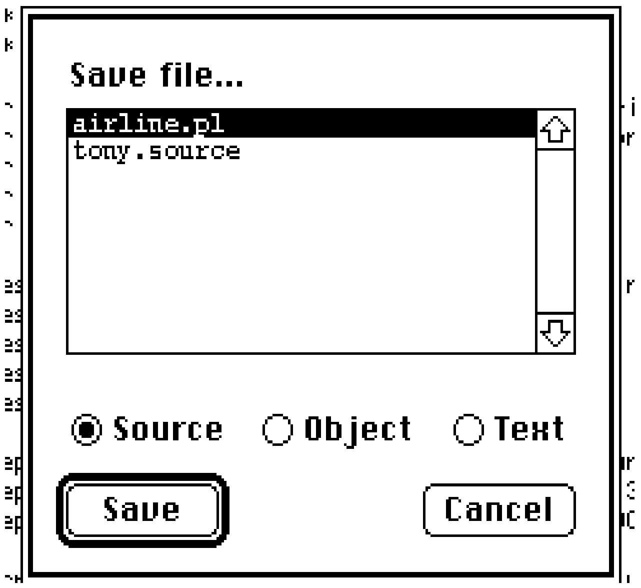
Source File:

tony.source

Fig. 10

Try opening 'airline.pl' now. If you then look at the 'Save...' dialog-box (File menu) you'll see that you can choose which of the two open files to save. In the usual way, you can choose to save more than one by holding down the shift key and selecting their names in the dialog-box (Fig. 11). Similarly, the 'Close...' option will allow you to decide which source files to close; and the 'Create...' choice in the Windows menu will allow you to specify which open source file the new window is to be assigned to.

This may seem elaborate at first sight, but is actually a very neat solution to the problems of segmenting programs into manageable chunks. However there is a slight semantic anomaly in the use of the word 'source': MacPROLOG refers to source files, which reside on disk, but also to source code, which is what you see on the screen in front of you and can edit. Hence the 'Save all source' choice in the File menu which actually saves the contents of all currently-open *windows* to their appropriate source files.



Save file...

airline.pl
tony.source

[✓ Source] Object Text

Fig. 11

The 'Source files...' choice in the File menu is a means of finding any particular window when you have a large number of them on the screen. To use it, make that choice from the File menu. The resulting dialog-box will be as in Fig. 12. Next, select the name of the relevant source file and click the 'Windows...' button. This will update the list of windows to correspond with your chosen source file; you can then click on the 'Select' button to bring that window to the front.

To repeat something said above: MacPROLOG insists that you keep all facts and rules which have the same name together in the same window and the same file. If you don't stick to this rule, you'll get an error message "Relation defined twice in same window" when you try to load your program via the 'Open...' choice. 'Relation' is MacPROLOG's term for a 'functor': the first word of a fact or rule, and what I have so far been referring to as its name. A similar error results if you try to load two source files which contain identical names for facts or rules.

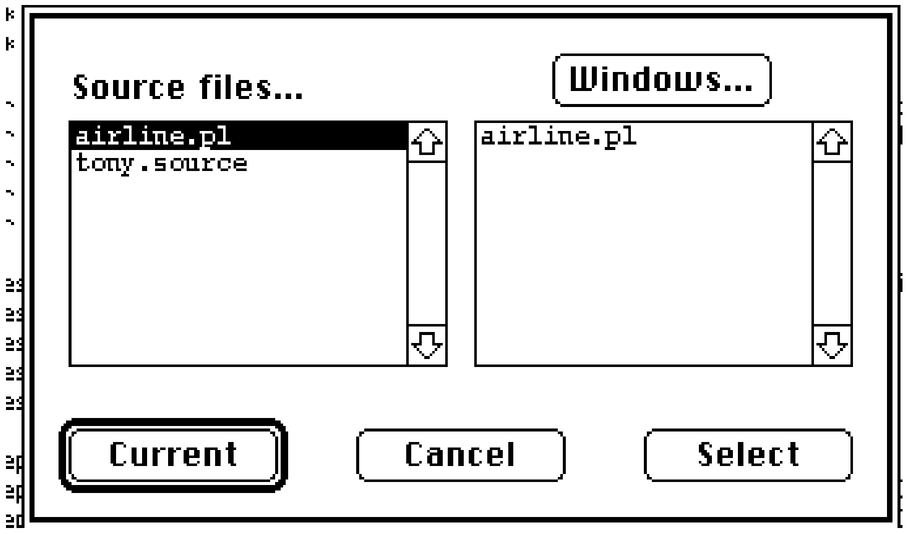


Fig. 12

Prolog will also issue a warning if you try to give to a new window a name which is already in use, that is if you have another window of the same name open on screen.

It is envisaged that like most programmers you will, when a bug arises, usually try to find it by eye. Beyond that point, TPM is your main debugging tool. It is ia powerful, graphical tracer and stepper: after a traced query you can press the 'run' button in the CGV to see your program re-execute until it bombs, and can then investigate the fault condition; or you can step through the program as above, from any point in its execution, to find out how the fault condition arose. TPM is state-of-the-art technology both in terms of program visualisation and in terms of the Prolog language.