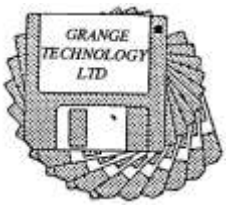
GT - PROLOG - QL (Part 1)



Oak Ridge, Tennessee, USA. Mel La Verne

Attempting to write a review of GT-Prolog (GTP) has proved to be rather more difficult than anticipated. There is, first of all, the necessity for familiarization with a new language. This, of course, was to be expected. What was not foreseen was the difficulty in reading (or rather, interpreting) the accompanying manual.

This review is necessarily written from the viewpoint of the neophyte, the newcomer to Prolog. Prior to this effort, my exposure to Prolog had been primarily through the popular, non-technical press and through some perusing of the QL Prolog manual.

For reference, all operation of GTP has been on a QL equipped with a Gold Card, v 2.32 and a Minerva rom, v 1.97.

GTP comes with a rather daunting manual of about 160 pages, approximately 40 in a "Workbench User Guide" (WUG), with the remainder in a "Reference Manual".

The Reference Manual is highly technical, assumes a good working knowledge of Prolog, and for the most part, seems about as accessible to the newcomer as, say, the "Handbook of Chemistry and Physics" would be to someone ignorant of either chemistry or physics. I will have little or nothing further to say on that portion of the overall manual.

Many manuals, and the WUG is no exception, tend to use jargon that "everyone knows" and fail to define many terms that we peasants do not know. For example, early in WUG, we find the statement "The null list value [] is not an atom.". How interesting ! Nowhere, not even in the Reference Manual, did I find a definition of "atom". The best I've been able to dig up is that an atom is the name of something. The square brackets delimit a set of items separated by commas (i.e., a list). Hence, "[]", with nothing between brackets has the value of the empty, or null, list. So, what was being said was, simply put, "A value is not a name."

In IQLR 111-5, GTP's author conceded that "information on Prolog within the QL community is somewhat lacking." He then provided an historical overview of Prolog and a brief description of how family relationships are written in Prolog. I believe it might be helpful in ensuring that we are all speaking the same language if I insert a very brief tutorial on Prolog before proceeding.

Prolog uses three kinds of statements: facts, rules, and queries. Facts define some relationship between or among given objects. Rules express new relationships among the given facts and/or rules. Finally, queries allow us to determine whether certain relations hold between objects. Obviously, Prolog is not confined to asserting family relationships. In the following, the operation of a simple digital circuit is described in a short Prolog program. In particular, we show how an "or" gate can be synthesized, using only "nand" gates.

Begin with the truth table for "or" and "nand":

x	y	or	nand
0	0	0	1
0	1	1	1
1	0	1	1
1	1	1	0

From the table we can write the "facts" about nand gates as:

```
nand(0,0,1).    nand(0,1,1).    nand(1,0,l).    nand(1,1,0).
```

These facts say that, for example, a gate with inputs 0 and 0 gives output 1 but inputs of 1 and 1 yield an output of 0. Note that each fact is terminated with a full stop (“.“).

The “rule” that defines an or gate, given the above facts, is:

```
or(X,Y,Z) :- nand(X,X,A), nand(Y,Y,B), nand(A,B,Z).
```

Here we assert that if we have a NAND gate with inputs X and output A, a second NAND gate with inputs Y and output B, and a third NAND gate with inputs A and B and output Z, then Z is the logical or of X and Y. Rules are also terminated with a full stop.

In the following Prolog program, “/*“ and “*/“ bracket a possibly multiple line comment. “%“ declares the remainder of the same line to be a comment.

```
/* A sample Prolog program defining an OR gate in terms of 2-input NAND gates */

nand(0,0,1).    % Define the four possible states of a 2-input NAND gate
nand(0,1,1).
nand(1,0,1).
nand(1,1,0).

or(X,Y,Z) :-    % Now express the rule for the OR gate
nand(X,X,A),    % With inputs tied together, we get an inverter
nand(Y,Y,B),    % Similarly for the second gate
nand(A,B,Z).    % NAND the inverted inputs to get the OR of X and Y
```

Note that the quantities in the rule’s parentheses are capitalized. This indicates that they are variables. More generally, the name of any quantity denotes a variable if it begins with a capital letter. Otherwise, we have a constant of some sort.
If we now ask Prolog (by way of a “query”) what inputs to the or gate are required for, say, an output of zero, Prolog would respond with X=0, Y=0 and a request for “More (y/n) ?“. Since there is only one solution, a “y” would result in a declaration that there is no other solution, and a reset for another query. A response of “n” simply resets for another query.

For the benefit of those who might like to investigate further, I have appended a bibliography of Prolog references that I have personally consulted at one time or another. They are ranked roughly in order of estimated difficulty, the easiest being first.

Turning now to the WUG, Chapters 1 and 2 are introductory and will be skipped over.

Chapter 3, Startup is where I first came a cropper. The instructions said that after loading the PRL (Prolog Runtime Library ?) file, GT-Prolog should be initialised with the command “exec GTProlog;flp2_Workbench_BIPS”. This indicated GTP to be on flp1_ (default), with the Workbench expected by GTP to be on flp2_. It was further said that if no file were specified, the name would be requested later. Clearly, this offered a way out of the problem posed by having all files on one disk. At the prompt, the disk was placed in drive 2 and the name Workbench_BIPS entered. The immediate response was “Invalid bootfile – please respecify”.

Not until 5 pages and 2 chapters later did I come across the statement that GTP makes no assumptions about device names, etc, and that full filenames (i.e., including drive specification) must be used.

In retrospect, had it been indicated that the "flp2_" was just a "for instance" or had the line about full filenames been presented earlier, I would not have fallen into the trap. Oh well, mea culpa.

To avoid any further problems on that score, I modified the Boot file to call TK2_EXT and added the command "EW GTProlog; flpl_Workbench_BIPS". The exec command caused the initial screen to come up with a dead cursor (which a CTRL-C soon enlivened); EW cured that minor annoyance.

The Workbench provides a convenient overall manager for GTP that allows entry to all parts of the system through a variety of menus (Workbench, Database, Break, and Error). Displayed options may be selected by using the arrow keys to highlight the option and pressing Enter or by entering the letter capitalized in the option name.

Options available from the top level Workbench Menu are Query, Edit, Database, Restart, and exit. Entering X or selecting eXit immediately terminates GTP. Restart causes re-entry afresh to the menu.

Entering Q or selecting Query causes entry into the query shell, an interactive interface that allows the user to ask questions (queries) about the current database. The shell starts the process by displaying the query prompt, **?-.** If the user responds with a query having correct syntax, the query is executed. For instance, using the sample OR-gate problem, one could ask, say, for the second input and the output if the first input is zero. The dialog would be as follows:

```
?- or(0,Y,Z).
Y:0
Z:0
More (y/n)?          (Response: y)
Y:l
Z:1
More (y/n)?          (Response: y)
No (more) solutions

?-
```

The Text Editor is normally entered by selecting the Edit option of the Workbench menu. It may also be entered directly from the Query shell by responding to the query with "edit(filename).".

A third entry to the Editor is available from the Break menu, whose options are identical to those of the Workbench menu. A caution is in order here, however. If the Break menu is arrived at from the Editor, use of the Editor option will cause an "exception" (read that as "error"); recursive use of the Editor is frowned upon.

Commands to the Editor operate in three different modes: immediate, command line, and "quick key".

Immediate commands are such as text entry or deletion, cursor movement, or those initiated with a function key.

Each command of a command line consists of a command letter followed by parameters bracketed by matching delimiter characters. For example, S/flp3_first_file/ would save the current edit file as first_file on flp3_. Multiple commands may be used on a line if separated by spaces.

Finally, we have quick key commands such as ALT+up or down arrow key, which scrolls the text by one full page, or ALT + B to move to the bottom of the file. ALT + X ends an editing session, with the edit text being saved if it has been modified since loading or the last save.

Entering ESC during editing produces a break state, during which queries and other commands may be carried out. Exiting the break allows continuation of editing. This is part of the Workbench interface.

When I first spotted the word "Tutorial" in chapter 6 of the WUG, I turned eagerly to it, hoping to be enlightened on Prolog. I was quickly disillusioned, however. Beyond a rather sketchy description of what each procedure is intended to accomplish, the program is introduced bluntly with : "The code is as follows:". I must admit, though, that the author, quite early on, honestly stated that the manual was not intended to teach the Prolog language.

The "Tutorial" does give a very good introduction to the entire process of creating the Prolog program file, compiling it, and running the program. With it, I succeeded in producing and running my first (small) Prolog program with a minimum amount of stumbling.

I did have an interesting but baffling experience in running one of the programs on the master disk. There is a file on the disk, "Queens_full", containing a procedure "queens(n)" which solves the general problem of placing n queens on the chess board, with n ranging from 1 to 8.

This seemed like a good opportunity to answer the question "How many configurations are there for n queens ?", where each configuration is a "safe" one. Following the "tutorial" instructions, I entered the query shell and, in response to the prompt, entered "queens(1).".

It was no surprise to me that the number of solutions turned out to be 8. The surprise came when, in response to the query prompt, I entered "queens(2).". The immediate response was to call up the "ERROR" menu, with "Fail" highlighted. The only option that allowed me to continue was "Succeed", which, of course, forces the system to accept the entry unless some other exception has occurred.

This hangup occurred for every entry, whether n was the same or different on successive queries. I have no explanation. Did I do something forbidden ?

Bibliography:

1."A Prolog Primer", W.F. Clocksin
BYTE, August 1987, pp. 147-150,154,156,158
A good but brief introduction to Prolog, done in tutorial fashion. This article seems to predate Reference 3, since it refers to the second edition of the book. Demonstrates, among other things, an application of Prolog to the design of a digital circuit (not the one given above) using NAND gates.

2. "A Prolog Primer", J. B. Rogers
Addison-Wesley. 1986. ISBN 0-201-06467-7
This one is excellent for the rank beginner (like me !). The first 94 pages comprise a tutorial introducing much of the basics of Prolog. As with most of the references, there are notational deviations. Unfortunately, the book is out of print but may be available on the second-hand market (I obtained the book through an interlibrary loan).

3. "Programming in Prolog", W. F. Clocksin & C. S. Mellish.
Third Edition, Springer-Verlag. 1987. ISBN 0-387-17539-3
This is more advanced than Reference 2 but it does start off gently for the novice. Since this reference uses the Edinburgh protocol, differences from GTP are minimal and occur primarily in input to and responses from the operating system.

4. "QL Prolog", Hans Lub
Quanta Library, Languages disk LA 01

This initial adaptation of Prolog to the QL is available to Quanta members only (hint). It lacks some of the niceties of GTP, such as "tail-recursion optimisation" and "garbage collection" but might be well worth playing with before taking the plunge with GTP. By the author's own admission, it is not "user-friendly" (but then, neither is GTP). As usual there are notationl variations from GTP.

5. "The Art of Prolog", L. Sterling & E. Shapiro.
MIT Press. 1986. ISBN 0-262-19250-0
Primarily for the advanced user but contains a number of examples that might be used for practice. Again, notaional differences abound.

GT PROLOG - QL (Part 2)

Oak Ridge, Tennessee, USA - Mel La Verne

In Part 1 of this review of GT-Prolog (GTP), I presented a short tutorial on Prolog for the benefit of those who, like me, were rank beginners in the language. Not a lot was given with respect to GTP operation. In this final part I propose to backtrack a bit and give somewhat more of the mechanics of running GTP.

Starting GTP with the Boot file specified in part 1, the PRL file is loaded and an initial copyright, etc screen is presented. Acknowledging the screen with an Enter brings up a screen showing the default memory sizes and I/O stream allocations. A simple "Enter" accepts each default value; entering a new value changes it.

GTP is initially configured for a QL with 512 Kb added RAM, 640 Kb total. If extra memory is attached, boosting the data, heap, and code allotments is recommended in order to reduce the frequency of "garbage collection".

I have found, by experiment, that a factor of as much as 5 can be used with a Gold Card before the dreaded "Insufficient memory available" message appears. With a Super Gold Card, an acceptable factor is 11.

Reducing the memory sizes is inadvisable; aside from its effect on garbage collection, there is the likelihood of program termination with the note "bootfile error: -3". Similar tampering with the I/O allocations can lead to spectacular crashes.

The final Enter causes the "bootstrap file", Workbench_BIPs (oddly enough, referred to on screen as the "boot" file), to be loaded and the startup windows to be displayed. For the curious, the extension BIPs refers to Built In Procedures.

The display is comprised of six windows, only four of which are visible initially. The major portion of the screen is occupied by the user window. As the manual puts it, this is the main interactive shell window.

To the right of the user area is the display window, used primarily for menus.

Beneath the user window is the message window, used for error or debugging notes and other dialogs.

The remaining area, at the lower right, is occupied by the break window, used for interrupting program execution.

The fifth area is the edit window, not initially visible. When called, this window overlays both the user and display windows and constitutes the main editor display area. The user and display windows are, of course, reclaimed when the editor is no longer active.

Finally, within the break window, we have the gc, or "garbage collector", window. A letter is displayed in this window whenever a garbage collector is activated, e.g., H for heap, S for stack. A garbage collector is automatically activated whenever a corresponding resource is used up. Frequent display of a letter may show inadequate initial assignment of memory size to that resource.

On the off chance that someone out there is not familiar with the term, let me give my defmition of "garbage collector". On accasion, a section of memory may fill up with items no longer needed. Suppose we have a routine that detects this condition and reclaims the space. The items removed are the garbage; the routine is the garbage collector.

At startup, the display window is occupied by the top level Workbench menu. This is the menu to which we attempt to return when something goes awry and we would like to start afresh. Since GTP is completely menu-driven, it might be helpful, at this point, to digress and show and comment on content of all the menus:

WORKBENCH	DATABASE	BREAK	ERROR	DEBUG
Query	Consult	Query	Fail	Step
Edit	Reconsult	Edit	Succeed	Hop
Database	compile	Database	Throw	Leap
Restart	Load	Restart	Break	Trace
eXit	reload	eXit	Restart	Notrace
				Else
				Call
				Fail
				Break
				Restart

All of the top level (WORKBENCH) entry points have been discussed previously, except for Database, which calls up (surprise !) the Database menu.

DATABASE:

"Consult" reads clauses from a specified file, compiles them, and adds them to those already in the database. My interpretation of the process is that if the same file is "consulted" more than once, then duplicate copies of the clauses will appear in the database.

Ordinarily, the above duplicating is undesirable, so we also have "Reconsult". With it, only those clauses not already in the database are added in; those already present are not duplicated. This would be the option to use in the case of adding one or several clauses to an existing Prolog file; only the new clauses should be compiled and added.

Note that with either of the above options, the compiled version is transient. When the machine is shut off, the compiled version disappears.

"compile" makes for a more permanent arrangement. Here, we require specifying two file names, one for a source file and one for an object file. Clauses read from the source file are compiled into object format and written to the object file. The object file is not added to the database at this time, but may be added later with "Load" or "reload".

Except for dealing with compiled (object) files, Load and reload are analogous in their actions to Consult and Reconsult, respectively.

BREAK:

Arrival here is the result of halting program execution by typing Esc in the Break window. Note that the options are the same as those for the Workbench menu; only the title differs.

With the possible exception of Edit, actions taken are the same as in the Workbench menu. As noted in Part 1, one should not return to the Editor from the Break menu; recursive use of the Editor is a no-no.

Depending on actions taken by GTP during the break, resumption of program execution may be possible if Esc is typed while in this menu.

ERROR:

This menu is selected as the result of an error being encountered while in a Built-In Procedure (BIP). An explanatory error message is displayed in the message window.

If Fail is selected, GTP resumes execution as if the BIP had failed.

Selecting Succeed causes execution to proceed as if the BIP had, in fact, succeeded. Recall that in Part 11 described how use of this option allowed me to run queens(n) for various values of n. I was forcing GTP to accept my query as valid.

Break, of course, returns one to the Break menu, allowing selection of most, if not all, of the top level options.

Restart drops everything and returns to the top level Workbench menu.

DEBUG:

An adequate explanation of some of the options in this menu would involve explaining the control flow model known as the "Byrd Four Port Box Model". Reference , in Part 1, has such an explanation. Together with a fairly simple example, Reference 3 takes seven and one-half pages to present it. Since 1 have neither the time, space, nor inclination for such an effort, some of the following will be simplistic and therefore imprecise. "Spypoint" I take as Prolog jargon equivalent to the more common "breakpoint".

Step is analogous to single stepping through, say, a Basic program. Slow, detailed, almost overwhelmingly so.

Hop and Leap should, to my way of thinking, indicate progressively larger jumps through a program. However, the definitions seem to contradict this conclusion. Hop is defined to stop on meeting a certain condition, call it A, ignoring spypoints met along the way. Leap, however, is defined to stop on meeting condition A or encountering a spypoint, which seems more restrictive. Perhaps the definitions are reversed??

Trace is defined like Leap with the added feature of having tracing enabled.

Notrace is easy: it turns Trace off.

Else causes local failure and backtracks to the most recent choice point (undefined, but which I take to be the nearest branching of the tree that we are descending).

Call restores the current goal to that execution state in which in which it has been invoked but candidate clauses have not yet been selected (the CALL port of the box model **!).**

Fail causes the current goal to fail (completely, according to the User Guide).

The remaining options, Break and Restart, are as before.

Digression completed, we return to running GTP.

Suppose we wish to enter and run a new program. The Workbench menu is present, so select Edit. You will be asked for a file name. Type in a name. . Now type in the program, remembering to terminate each clause with a full stop ".". You may now use the "quick key", ALT+C, to consult (i.e., compile) your program.

Having (successfully, of course) entered and compiled your program, you will now want to run it. Type Esc while still in the Editor. This presents the Break menu, from which you select the Query option. If your response has correct syntax (don't forget that full stop **!),** the program will now run.

If you wish to run an existing program, the simplest way seems to be to use the Query from the Workbench menu. In reponse to the prompt (?-**),** enter **"** edit(drive_file).". Then proceed as above.

"The time has come," the Walrus said, "to talk of many things." And one of those things whose talking time has come is an overall assessment of GTP.

Technically, GTP seems excellent, a real tour de force. Where I feel it is lacking is in providing sufficient detail to accomplish more than elementary operations. I can write, compile, and run a Prolog program (sometimes !) but little beyond that.

For example, I would like sometimes to send output to a printer. Presumably, I would use "Write" (or is it "Writeq" ?)**,** specify an output "Stream" in some as yet unknown form, and open that stream somehow to my printer. Nothing unusual there; I can determine what to do to achieve my ends. If only I could find out precisely how! Examples, examples, please.

Then, there are the questions that simply seem to be ignored. For instance, how does one exit a Query? After much fumbling, I have found that giving an empty statement works, i.e., type just the full stop and Enter. GTP responds with an error, "invalid token'. From the ensuing Error menu, Restart takes me back to Workbench, ready to start over. Surely, there must be a less clumsy way!

Perhaps the User Guide needs to expand from its present 40 pages to about 80 pages? Oh well, as the private said to the sergeant, "Details, details".