Prolog Installation for Systems Programmers - 26th June '78

1. Introduction

This note contains the information needed to build, modify and debug the DECsystem-10 Prolog system. It also lists all the source and derived modules which make the Prolog system.

2. Conventions

Unfortunately, there are not as yet fully obeyed coding and naming conventions for the Prolog system. However, some rules have more or less been followed:-

- a Prolog source file has extension .PL;
- a functors definition file has extension .FNS;
- a file of Prolog external definitions has extension .EXT;
- a file of commands to be used by the compiler has extension .CCL;
- a relocatable file containing modules each coming from a different source file has extension .LIB;
- the filenames of Prolog modules used in the interpreter or compiler have first letter Z;
- names of entry points or external quantities begin with $;
- files created by SUBFIL compaction have extension .MAS.

3. Where are the sources

Sources are compacted by SUBFIL in .MAS files. However, if a module has been recently modified, the copy in the .MAS file is the previous version, and the current version is available as a single file.

All the Prolog sources needed for the interpreter or the compiler are in PLS.MAS. All the Macro sources needed for the runtime system are in PLM.MAS.

To extract File1,...,Filen from name.MAS, use the MIC command

```
/XTRACT <File1,...,Filen>,name
```

4. How to create a system from scratch

The following files should exist in the directory:-

PLC.EXE	a working compiler
PLM.MAS	system's Macro sources
PLS.MAS	system's Prolog sources
START.FNS	initial functors: '[]', '.' and '.'(_,_)
O.CCL	defines the loading sequence for Prolog programs
PLC.EXT	external definitions for the compiler
O.EXT	external definitions for the interpreter
MACRO.MIC	assembles the system's Macro sources
COMMON.MIC	compiles the Prolog sources common to every program
COMLIB.CMD	creates COMMON.LIB (called from COMMON.MIC)
PROLOG.MIC	creates the interpreter
PROLIB.CMD	creates PROLOG.LIB (called from PROLOG.MIC)
PROLOG.CMD	loads the interpreter
PLC.MIC	creates the compiler
PLCLIB.CMD	creates PLC.LIB (called from PLC.MIC)
PLC.CMD	loads the compiler

Begin by exploding both PLM.MAS and PLS.MAS:-

```
.R SUBFIL
*PLM.MAS
*PLS.MAS
^C
```

After this, delete any files with extension .NEW which may have been created if there are newer sources in the directory. Now, excute the MIC command /MACRO, to assemble the Macro sources. Continue by executing the MIC command /COMMON, to compile the Prolog modules common to the interpreter and the compiler. Now, the MIC command /PROLOG will compile, load and save a new interpreter NP.EXE, and the command /PLC will compile, load and save a new compiler, PLCN.EXE.

If you are sure that the new interpreter and compiler are working well, delete or rename to other names PLC.EXE and PROLOG.EXE, and rename NP.EXE to PROLOG.EXE, PLCN.EXE to PLC.EXE.

5. Layout of the system's relocatable files

To save space and simplify handling, the relocatable modules which form the compiled Prolog system are organized (perhaps not in the best way) in libraries. We list now the correspondence between source files and relocatable libraries. Each source is tagged either with A (all programs), C (compiler), I (interpreter), R (programs which require it - the interpreter needs all these), N (only for naked programs), or B (used by compiler, interpreter and programs which require it):-

library	sources which make it up		
PLLIB2.REL	PLLIB2.MAC	A	basic runtime routines
PLRUN2.REL	PLRUN2.MAC	A	core control, I/O, etc.
PLINI2.REL	PLINI2.MAC	A	small integers, Prolog start/up
IOLIB2.REL	IOLIB2.MAC	A	'display' eval pred
MHEAP.REL	MHEAP.MAC	A	heap management
SHIFT.REL	SHIFT.MAC	A	stack shifts
GARBGE.REL	GARBGE.MAC	A	garbage collection
COMMON.LIB	ZIOCTL.PL	A	Prolog I/O interface
	ZSYNER.PL	B	syntax errors reporting
	ZOPS.PL	B	operators
	ZRECAL.PL	B	'tag', 'tagged' and 'untag'
	ZTOKE.PL	B	read tokens
	ZNAME.PL	B	'name' eval pred
	ZMISC.PL	B	common useful preds
	ZINHTA.PL	A	initialise hash table
PROLOG.LIB	ZEGALF.PL	R	'==' eval pred
	ZWRITE.PL	R	writes terms
	ZDBASE.PL	R	internal database
	ZREAD.PL	R	reads terms

	ZUNIV.PL	R	'=..' eval pred
	ZSTATS.PL	R	statistics reporting
	ZINPRT.PL	R	initialises property table
	ZSAVE.PL	R	interface to 'save'
	SAVE.MAC	R	'save' and 'restore'
PLC.LIB	ZSCAN.PL	C	parses compiler commands
	ZTMPCO.PL	C	calls other programs
	ZMREAD.PL	C	'metaread'
	ZADMN0.PL	C	compiler control
	ZADMN1.PL	C	functors and exts files
	ZADMN2.PL	C	outputs Macro code
	ZCLAUS.PL	C	compiles a clause
	ZBLOCS.PL	C	compiles a procedure
	ZGOAL1.PL	C	compiles goals
	ZGOAL2.PL	C	"
	ZDCG.PL	C	compiles grammar rules
	ZLTERS.PL	C	compiles terms in head
	ZRTERS.PL	C	compiles terms in body
	ZFLAG.PL	C	signals various conditions
	ZEVAL1.PL	C	compiles arithmetic
	ZEVAL2.PL	C	"
	ZEVAL3.PL	C	"
	ZCSTAT.PL	C	compiler statistics
	~~ABORT.MAC~~	~~N~~	~~dummy 'abort' (='halt')~~
~~ABORT.REL~~	~~ABORT.MAC~~	~~N~~	~~dummy 'abort' (='halt')~~
START.REL	START.PL	N	starts naked programs
CTRAP.REL	CTRAP.MAC	I	interpreter ^C trap
ZSVWX.REL	ZSVWX.PL	I	interpreter control
ZKNOW.REL	ZKNOW.PL	I	interpreter main loop
PROLOG.REL	produced when compiling the interpreter		
ZNOEXT.REL	ZNOEXT.PL	I	dummy main module
PLC.REL	produced when compiling the compiler		

6. How to update a module

First, get a copy of the source as described in 3. Edit it, and recompile as described in the next section, producing a relocatable file, module.REL, say. Now consult the list in the previous section. If the module is part of one of the .LIB relocatable libraries, name.LIB, say, you must update that library, using the MIC command

```
/UPDATE name,module
```

The relocatable file module.REL is included in name.LIB, module.REL is deleted and the old version in name.REL is stored in module.OLD.

7. Recompiling a module

If the module is a Macro source, module.MAC, just give the command

```
.COMPILE module
```

If the module is a Prolog source for the compiler, module.PL, type the incantation

```
          .RUN PLC
          program:PLC/S/M
          :module
          :]]
```

If there are new functors, a file PLC.MAC is created, which must be doctored by

```
          EDIT PLC.MAC
          *OR$V
          $$V
          *X
          .COMP/COMP PLC
          .DEL PLC.MAC,PLC.BAK
```

If the module is a PROLOG source for the interpreter, incant

```
          .RUN PLC
          program:PROLOG
          :module
          :]]
```

If there are new functors, 0.FNS must be updated as follows

```
          .COPY 0.FNS=PROLOG.FNS
          .DELETE PROLOG.FNS
```

If the module is part of COMMON.LIB, beware!. Compile it with the commands

```
          .RUN PLC
          program:COMMON
          :module
          :]]
```

eg.

If there are no new functors, all is OK, and you sould not produce COMMON.REL. If there are new functors, you are in trouble, because you must now recompile from scrath the interpreter and the compiler ! You can either do this, using the (time consuming - 8 minutes CPU) /PROLOG and /PLC MIC commands, or else, as an interim solution, compile the module wrt. the program you want to update (PLC, PROLOG), save the current copy of [illegible] COMMON.LIB with some other name, and then update COMMON.LIB as described in 6. The new COMMON.LIB will no longer be loadable with all Prolog programs, but only with the one you are updating and the ones which do not use the changed module. This solution must be corrected soon by recompiling from scratch, lest you forget about it and get into trouble.

8. Adding a new module to a library

If you have compiled a new module into module.REL, and want to add it to name.LIB, just incant

```
          .R MAKLIB
          *name.LIB=name.LIB,module/APPEND
          *^C
```

You should now include the name 'module.REL' in the appropriate library building control file (COMLIB.CMD, PROLIB.CMD or PLCLIB.CMD), and also the instructions for its compilation in the appropriate MIC file (MACRO.MIC, COMMON.MIC, PROLOG.MIC or PLC.MIC).

9. Loading instructions for programs

The loading instructions for Prolog programs are included by the compiler in the program file, which copies the instructions for normal programs from O.CCL, and for naked programs from NAKED.CCL. These files are in the standard CCL format for LINK-10.

Programs compiled with the S switch (scratch) have no loading instructions included. In this case (the compiler is an example) you have to create your loading instructions - they should at least include the loading of all the modules tagged with A in 6. Examine PLC.CMD to see this in the case of the compiler.

10. Standard functor files

The compiler requests automatically O.FNS, which contains all the interpreter functors, when there is no .FNS file for the program being compiled. If you want to create your functors from scratch, copy a functors file for your program from START.FNS.

11. Standard externals

The compiler uses O.EXT, containing all the external declarations for the evaluable predicates, if there is no .EXT file for the program being compiled. Notice that O.EXT must be updated whenever new eval preds are defined.

12. Universal files

There are four universal symbols files in the system, IDENS2.UNV MACROS.UNV and IDENSF.UNV for the system Macro modules, and EQS2.UNV for the assembly of compiled Prolog programs. Only EQS2.UNV must be available for general usage, in the Prolog directory.

IDENS2.UNV is produced form IDENS2.MAC, IDENSF.UNV from IDENSF.MAC, MACROS.UNV from PLMAC2.MAC, and EQS2.UNV from EQS2.MAC. The .REL files produced when those Macro files are compiled are not needed, and can be deleted.

Changes to the Prolog compiler - 15th June '78, version 1(7)

1. Virtual memory and non-virtual memory operation

See the 1st paragraph of Section 1 of PROL22.TXT

2. New or changed evaluable predicates

See Section 2 of PROL22.TXT

3. Internal changes

The instructions for LINK-10 to load the runtime modules with a compiled program are no longer given in a special MIC file (LOAD.MIC and NAKED.MIC). Instead, the compiler produces code so the runtime system is loaded automatically.

To load a program PROG, with modules MOD1, MOD2, give the Monitor command

```
                  .LOAD PROG,MOD1,MOD2
```

This command will load the program as a part of the interpreter, and you will have all the interpreter functions available to you.

If you want to load your program with selected runtime modules instead of the full interpreter, you must use one of two compiler switches, /N or /S. The N switch is required when you want to load your program with JUST the runtime modules it uses. The S switch is used if you want to control yourself what runtime modules are loaded.

Supposing your program is called MYPROG, to compile it to be loaded automatically with the runtime modules it requires, the compiler command should be

```
                  .RUN PLC
                  program:MYPROG/N
                  :       .
                      modules
                          .
                  :]]
```

With this option, your program can NOT call the evaluable predicates 'break', the various 'assert's, and 'retract'. Also, ^C interruptions will not work. The evaluable predicate 'abort', I/O and stack full errors will fail to the Monitor. To load a program compiled with the N switch, use the 'LOAD' Monitor command as above.

If you want to control yourself what runtime modules are loaded, the switch /S should be used in the compiler command above instead of /N. In that case, no runtime modules will be loaded automatically.

Remember that the correct operation of the Prolog compiler requires that you have the Prolog area in your seach list. You can achieve this with the SETSRC program or a switch to LOGIN, both described in the DEC Operating System Commands Manual.

```
; Control file to create in Magtape a Backup interchange dump of the
; Prolog system (sources excluded).
; Call by
;          /system <tape-id>
;
; where <tape-id> is the identification of the tape to be written.
; This file assumes that all files, except the User's Guide, are in the area
; where the command is being executed. The User's Guide is supposed to be in
; [400,424]. One can, however, add other directories to the current area with
; Setsrc.
;
; First, check if all files are available:
.error %
; Look for the compiler and interpreter core images
.dir plc.exe,prolog.exe
.if(error) .goto NOFIL
; Look for auxiliary compilation files
.dir start.fns,0.fns,0.ext,0.ccl,naked.ccl,eqs2.unv
.if(error) goto NOFIL
; Look for basic runtime system
.dir pllib2.rel,plrun2.rel,plini2.rel,mheap.rel,shift.rel,garbge.rel,iolib2.rel
.if(error) .goto NOFIL
; Look for interpreter modules
.dir zsvwx.rel,zknow.rel,ctrap.rel
.if(error) .goto NOFIL
; Look for module libraries
.dir common.lib,prolog.lib
.if(error) .goto NOFIL
; Look for naked program's special modules
.dir start.rel
.if(error) .goto NOFIL
; Look for documentation
.dir prol22.txt,plc7.txt,ug.mem[400,424]
.if(error) .goto NOFIL
.error ?
; Now mount the tape
.mount mta:backup/ree:'a/wen
.if(error) .goto NOTAP
; Now dump the Prolog system.
; Two identical save sets with names PROLOG and PROLOGBIS are written.
.r backup
*interchange
*ssname PROLOG
*save prolog.exe,plc.exe,0.fns,start.fns,0.ext,0.ccl,naked.ccl,eqs2.unv,-
*pllib2.rel,plrun2.rel,plini2.rel,mheap.rel,shift.rel,garbge.rel,iolib2.rel,-
*common.lib,prolog.lib,zsvwx.rel,zknow.rel,ctrap.rel,-
*start.rel,-
*prol22.txt,plc7.txt,ug.mem[400,424]
*ssname PROLOGBIS
*save prolog.exe,plc.exe,0.fns,start.fns,0.ext,0.ccl,naked.ccl,eqs2.unv,-
*pllib2.rel,plrun2.rel,plini2.rel,mheap.rel,shift.rel,garbge.rel,iolib2.rel,-
*common.lib,prolog.lib,zsvwx.rel,zknow.rel,ctrap.rel,-
*start.rel,-
*prol22.txt,plc7.txt,ug.mem[400,424]
*rewind
*print prolog.dir
*rewind
*exit
.dis backup
.print/copies:2/delete prolog.dir
```

```
.goto EXIT
NOFIL::
; Missing files
.mic abort
NOTAP::
; Tape not available
.mic abort
EXIT::
; Tape successfully written
```