Reference Summary of Prolog-X

W F Clocksin, Computer Laboratory, University of Cambridge

June, 1984

rev: 20 July 84

Summary

Prolog-X is an implemented portable interactive Prolog system in which clauses are incrementally compiled for a virtual machine. At present, the virtual machine is emulated by software, but it has been designed to permit easy implementation in microcode or hardware. Prolog-X running on the software-based emulator provides performance comparable with existing Prolog systems.

Design Goals

The goal of Prolog-X is to provide a high-performance Prolog implementation suitable for large-scale commercial and industrial applications. It is appreciated that applications of this type will run on machines that have a large (at least 16 Mb) directly-addressable virtual memory space. Consequently, Prolog-X can only be run on machines having at least a 32-bit word length, of which at least 24 bits are available for addressing.

To provide usable response with heavy memory loading, Prolog-X uses sophisticated implementation techniques to make efficient use of computing time and memory space. The system uses an optimising compiler to convert clauses into a compact byte-code instruction stream for a virtual machine which can be emulated at high speed. Internal operations, such as data structure management, are uniformly designed to be of constant overhead or at least of low complexity, scaling up without disproportionally degrading performance. Furthermore, critical parts of the Prolog-X system are designed in such as way as to imply a straightforward reimplementation in assembly language, hardware, or microcode.

Prolog-X is a compiler-based Prolog which, unlike previous such systems, is reasonably portable. Prolog-X has been ported to the HLH Orion, ICL Perq, ICL 2980, ~~GEC 63~~, and DEC VAX. To demonstrate its efficiency, compatibility, and comprehensiveness of implementation, Prolog-X has been used to compile and run *CHAT* and *Press*, two large programs written originally for DECsystem-10 Prolog. The Prolog Library can also be used, which makes available development tools and scientific routines such as an arbitrary-precision rational number arithmetic package.

The User Language

The dialect of Prolog available to the user is intended to be compatible with the core language as specified in *Programming in Prolog* (Clocksin and Mellish, 1981), of which widespread implementations exist for the DECsystem-10 (Byrd, Pereira, and Warren, 1980), VAX (Pereira, 1981), and PDP-11 (Mellish, 1980).

Unrestricted syntax compatible with the DECsystem-10 implementation is used: overloaded operator declarations with precedences in the range 1..1200, curly brackets, etc. To support large-scale applications, Prolog-X also offers lexical modules and easy interface to procedures written in the base language (C). Programs may contain Grammar Rules.

Some relevant deviations from normal practice are as follows:

- Disjunctions within clauses are interpreted rather than compiled, and "cuts" within disjunctions are interpreted correctly. The only interpretation overhead in this case is an extra procedure call per goal.
- Debugging facilities are not built-in. Because clauses are compiled, some of the information needed by a built-in debugger is lost, and to increase performance, clauses may be executed in ways not obvious to the user. However, it is possible to use an interpretive debugger (available from the Prolog Library) in the usual way, which imposes a conventional execution strategy.
- A normal Prolog "top level" is provided. As clauses are compiled, the source clause is also retained in the database together with the names of variables used within the clause. This may be used, for example, to list clauses (using `listing`) exactly as they were typed in. However, the user has the option to compile clauses without retaining the source clause in the

database. The advantage is faster compilation and less memory usage, but such clauses may not be accessed by **clause** or **retract**.

- Input must be in upper- and lower-case, so the following syntactic devices are considered obsolete and have been discarded: '`,..`', '`LC`', '`NOLC`'.
- Some commonly used predicates defined in various Prolog libraries have been built-in.

Implementation Details

The current implementation includes the following features:

- Large address space (2^{28} bytes).
- Incremental garbage collection is performed during backtracking, and when clauses are retracted.
- All clauses are compiled to a compact byte-code representation, with automatic variable mode analysis and peephole optimisations. Some built-in predicates are open-coded.
- Tail recursion optimisation, early detection of determinacy, least-committment allocation strategy.
- Automatic clause indexing on first argument of goals.
- Arbitrarily long bit-strings that are tested for equality during unification (for compact strings, big numbers).

The current release of Prolog-X does not implement the following features. They are recognised as important, and there is no reason other than lack of manpower why they cannot be implemented. Prolog-X has been designed with the intent that these features could be easily added to the existing implementation:

- Compiled disjunction.
- Incremental garbage collection of global stack in the absence of backtracking.
- Floating-point arithmetic.
- Improved clause indexing.
- Access to external databases of ground unit clauses.

Data Structures

Data structures used in Prolog-X programs have the following restrictions:

- *integers*: are in the range -134217728 to 134217727. Overflow is not checked. An integer to the base N ($0 \leq N \leq 9$) may be represented as standard, but base 0 is reserved for representing character codes (for example, `0'A == 8'101`).
- *strings*: previous implementations convert these to lists of integers. Prolog-X represents them as a distinct data structure (a packed byte array) for greater efficiency. The maximum length of a string is 16777215 characters, but in practice is subject to memory limitations. The maximum length of string that can be read by **read(X)** is approximately 255, although this can be changed. Each component of a string may contain any ordinal in the range 0..255.
- *atoms*: Atoms represent their name as a string, so the observations for strings hold for atoms.
- *variables*: A maximum of 255 differently-named variables may appear in a clause. This restriction can be lifted in later releases by the unlikely event of popular demand.
- *compound terms*: The maximum arity of a functor is 16777215, but in practice is subject to memory limitations. Only functors having arity less than 256 can be **asserted**. This restriction can be lifted in later releases by popular demand.

Errors

The error handling mechanism is accessible to the user as a Prolog procedure **error_handler**. The default action for errors is to print a message and a culprit. Additionally, syntax errors cause extra information to be printed. If the current file is not the user, then the file name and line number and column number in the file is printed. After a syntax error, recovery is attempted by skipping to what is likely to be the end of the clause, and reading again.

Typing the interrupt key (control-C on Unix systems) causes the procedure **break_handler** to be entered at the earliest opportunity. The default break handler provides a "top level" with the prompt preceded by the word `(Break)`. To exit from the break, type the end of file character (control-D or control-Z on Unix systems). The break handler can also be modified by the user.

Input and Output

The Unix standard I/O library is used. Output is flushed only (a) when a newline is printed, or (b) when input from the user is requested. The built-in procedure **ttyflush** is provided. There is a limit on the number of files than can be open at any one time.

Operator Declarations

These are the same as provided by DEC-10 Prolog. In Prolog-X, operator declarations are subject to `module` scoping, and may be imported in the same way as procedures.

Built-in Predicates Exported to the User

The built-in predicates are fully explained in the DECsystem-10 manual, the C-Prolog manual, or the Prolog textbook. In the middle column, an X means a predicate new in Prolog-X; L means a Prolog library predicate which has been built-into Prolog-X; O means that the compiler open-codes the predicate (emits a specific machine instruction). The 'O' entry should not normally concern the user.

`abolish(F,N)`		Retract all clauses for procedure F, arity N.
`abort`		Abort execution of current directive
`arg(N,T,A)`	O	The Nth argument of term T is A.
`assert(C)`		Assert clause C.
`asserta(C)`		Assert C as first clause.
`assertz(C)`		Assert C as last clause.
`atom(T)`	O	T is an an atom.
`atomic(T)`	O	T is an atom or an integer.
`bagof(X,P,B)`		The bag of Xs such that P is provable is B.
`break_handler`	X	Call the break handler.
`call(P)`	O	Call procedure P.
`clause(P,Q)`		There is a clause, head P, body Q.
`clause(P,Q,R)`		There is a clause, head P, body Q, ref R.
`compare(C,X,Y)`		C is the result of comparing terms X and Y.
`compile(F)`		Extend the program with clauses from file F, do not retain source.
`consult(F)`		Extend the program with clauses from file F.
`display(T)`		Write term T in prefix form.
`endmodule(A)`	X	End of module A.
`erase(R)`		Erase the clause with ref R.
`error_handler(N,T)`	X	Call the error handler with error number N, culprit T.
`fail`	O	Backtrack immediately.
`findall(T,G,L)`	L	Like `bagof(T,G,L)`, but free variables existentially quantified.
`findall(T,G,S,L)`	L	Like `findall(T,G,L1), append(L1,S,L)`, but cheaper.
`forall(P,Q)`	L	For all P provable, prove Q.
`functor(T,F,N)`	O	The principal functor of term T is F with arity N.
`get(C)`		The next non-blank input character is C.
`get0(C)`		The next input character is C.
`halt`		Abort and exit Prolog.
`import(P,M)`		Import procedure(s) P from module M.
`integer(T)`	O	T is an integer.
`Y is X`	O	Y is the value of arithmetic expression X.
`keysort(L,S)`		The list L sorted by key yields S.

`length(L,M)`	L	The length of list L is M.
`listing(F,A)`		List the clauses of predicate with functor F, arity A.
`name(A,L)`		The name of atom or string or number A is list L.
`name_sub(T,N,C)`	X	C is the Nth character of atom or string T.
`nl`		Output a newline.
`nonvar(T)`	O	Term T is a non-variable.
`not(P)`		Like `\+(P)`, but a warning is printed if P is not ground.
`numbervars(T,M,N)`		Number the variables in term T from M to N-1.
`op(P,T,A)`		Make atom A an operator of type T, precedence P.
`put(C)`		Output the character C.
`read(T)`		Read term T.
`read(T,L)`	X	Read term T, with variable name list L.
`recorda(K,T,R)`		Record T first under key K, with ref R.
`recorded(K,T,R)`		T is recorded under key K, with ref R.
`recordz(K,T,R)`		Record T last under key K, with ref R.
`repeat`		Succeed.
`retract(T)`		Erase the first clause matching T.
`save(F)`		Save the current state of Prolog in file F.
`see(F)`		Make file F the current input stream.
`seeing(F)`		The current input stream is name F.
`seen`		Close the current input stream, revert to **`user`**.
`setof(X,P,S)`		The set of Xs such that P is provable is S.
`simpleterm(T)`	L	Term T is not a compound term.
`skip(C)`		Skip input characters until after character C.
`sort(L,S)`		The list L sorted into canonical order is S.
`string(T)`	X	Term T is a string.
`succ(M,N)`	L	integer N is the successor of integer M.
`system(P)`	L	P is a built-in predicate.
`tab(N)`		Output N spaces.
`tell(F)`		Make the current output stream F.
`telling(F)`		The current output stream is named F.
`told`		Close the current output stream, revert to **`user`**.
`true`		Succeed once.
`unknown(O,N)`		Set the "unknown procedure" action from O to N.
`var(T)`	O	T is a variable.
`visa(L)`	X	Confer permissions on exportable procedures.
`write(T)`		Write T.
`writedepth(O,N)`		Set the write depth from O to N.
`writeq(T)`		Write T, quoting if necessary.
`writewidth(O,N)`	X	Set the output linewidth from O to N.
`!`	O	Cut any choices taken in the current procedure.
`\+(P)`		P is not provable.
`P -> Q ; R`		If P is provable, the prove Q, else prove R.
`P -> Q`		Like `P -> Q ; fail`.
`X < Y`	O	As integer expressions, X is less than Y.
`X =< Y`	O	As integer expressions, X is less than or equal to Y.
`X > Y`	O	As integer expressions, X is greater than Y.
`X >= Y`	O	As integer expressions, X is greater than or equal to Y.
`X =:= Y`	O	As integer expressions, X is equal to Y.
`X =\= Y`	O	As integer expressions, X is not equal to Y.
`X = Y`	O	X unifies with Y.
`X \= Y`		X does not unify with Y.
`T =.. L`		The functor and arguments of T are elements of list L.
`X == Y`		Terms X and Y are strictly identical.
`X \== Y`		Terms X and Y are not strictly identical.
`X @< Y`		Term X precedes term Y in the canonical ordering.

`X @=< Y`	Term X precedes or is identical with Y.
`X @> Y`	Term X follows term Y.
`X @>= Y`	Term X follows or is identical with Y.

Arithmetic Expressions

The following expressions can make up arithmetic expressions to be used on the right-hand side of an **is**. X and Y are expressions. Open-code is generated for arithmetic expressions known at compile-time. Variables in arithmetic expression may be bound to either integers or other arithmetic expressions; in the latter case, the bound expression will be interpreted at run-time. *Arithmetic exceptions are not detected.*

`+X`	Unary addition.
`-X`	Unary subtraction.
`X + Y`	Addition.
`X - Y`	Subtraction.
`X * Y`	Multiplication.
`X / Y`	Division.
`X mod Y`	Remainder.
`X /\Y`	bit conjunction.
`X \/ Y`	bit disjunction.
`X << Y`	bit shift X left by Y bits.
`X >> Y`	bit shift X right by Y bits.
`\X`	bit negation.
`cputime`	CPU time since the start of the session, in milliseconds.
`calls`	Number of Prolog procedure calls since the start of the session.
integer	The value of any integer.
atom	The ASCII code of the first character of any atom.
string	The ASCII code of the first character of any string.
list	The first element of the list is evaluated as an expression.