AI Multilanguage System McPOPLOG: The Power of Communication Between its Subsystems*

I. BRUHA
Department of Computer Science and Systems, McMaster University, Hamilton, Canada, L8S 4K1

There have been several attempts to combine the programming language Prolog with procedural programming languages. One practical and useful solution is to combine the programming languages Prolog and POP-11 to one environment called POPLOG. The author of this paper has implemented a version of the AI multilanguage environment POPLOG, called McPOPLOG (McMaster version of POPLOG). This paper firstly describes the chief attributes of the implemented system McPOPLOG, and then focuses on the communication means between POP-11 and Prolog in McPOPLOG.

Received March 1988, revised September 1992

1. INTRODUCTION

There have been several attempts to combine the programming language Prolog with procedural programming languages as Modula,[1] LISP, e.g. LOGLISP,[2] QLOG,[3] POP-11, i.e. POPLOG.[4] The multilanguage system POPLOG, developed at Sussex University, U.K., is an adequate environment for software problems that require utilising both procedural and declarative programming techniques. It involves incremental compilers for POP-11, Prolog, Common LISP and standard ML.

The author of this paper has developed and implemented a completely new version of POPLOG (McMaster version of POPLOG, McPOPLOG),[5,6] that is portable to any 32-bit machine and allows straightforward communication means between both languages. Section 2 of the paper describes the fundamental differences between Sussex POPLOG and McPOPLOG, and briefly introduces the way of constructing the Prolog compiler of McPOPLOG in terms of POP-11 data structures and control techniques. Section 3 describes the communication means between POP-11 and Prolog in McPOPLOG.

If the reader is not familiar with POP-11, he/she can look at it as a list processing language, like LISP. However, POP-11 exhibits a Pascal-like (or Algol-like) well-structured syntax, has a large set of various data structures including records, vectors, strings, arrays, lists, closures, etc., and involves many non-standard control structures such as processes, backtracking, pattern matching, database processing, macro facility, and so on.

To understand the POP-11 programs introduced here we should know only:

```
vars x, y;
/*declaration of POP-11 variables x, y
 */
e ->x;
/*value of the expression e is assigned
to the variable x */
e =>
/*value of the expression e is printed
  and preceded by ** */
```

* This research has been supported by NSERC research grant A8034.

```
member(a,x1)
/*POP-11 standard function that returns
  true iff a is an element of the list x1
 */
x1 <> y1
/*operation for concatenation of the
  two lists*/
function f(x,y);
end;
/*declaration of the function f with */
    ...
/* formals x , y */
function g(x) => r;
    ...
/* */
end;
/*explicit result r of the function g
  is declared */
```

2. FEW WORDS ABOUT McPOPLOG

We have followed the idea of joining the two high-level AI programming languages Prolog and POP-11 into one system, but unlike the Sussex version of POPLOG our guiding principles, were the following.

(i) The multilanguage system should support those structures, procedures, and control techniques that are important to *AI* projects only; in other words, we did not want to create a *system* language that would solve system tasks as efficiently as, for example, C does.

(ii) Our version is to be a very simple system with a reasonably small number of standard facilities, with as few memory requirements as possible.

Here are the fundamental attributes of McPOPLOG, together with the principal differences between the Sussex version and ours.

(1) The entire McPOPLOG is written in C rather than in POP-11. The system is very compact, since the Prolog subsystem utilizes only standard POP-11 data structures and control mechanisms without any change. McPOPLOG is portable to any 32-bit machine, only the code generator must be rewritten.

(2) As a consequence of its compactness and the above motto, our version became much smaller (75 kB vs. Sussex's 650 kB), thus it can run on much smaller computer systems, and is commercially much cheaper.

Our version is about two to three times faster.

(3) It has straightforward communication means between both languages (POP-11 and Prolog, see Section 3).

(4) The POP-11 subset has been substantially simplified by returning partly to the Edinburgh's POP-2,[7] taking new features of Sussex's POP-11[8] that are important to AI projects, not to system tasks (it involves less than 250 reserved words, i.e. keywords, built-in functions and system variables). Furthermore, if there was any ambiguity or a misconception we tried to solve it in a Prolog or C style. For example, the operation for list matching in our version is an extension of the operation equivalence = (as in Prolog). Some utilities have been added, e.g. besides the continuation backtracking our POP-11 provides the state-saving backtracking. On the other hand, our version comprises only the most necessary input/output routines, and the syntax procedures are available to a superuser only since, in our experience, the macro facility is sufficient for generating a new syntax.

(5) The Prolog subset is equivalent to the Edinburgh standard version. The Prolog grammar rules are optional. LISP (more precisely, a subset of Common LISP) is optional, too.

(6) McPOPLOG, following the above motto, does not involve any screen editor. We have found that most users are not willing to learn a new editor and they prefer to call their favourite editor from within McPOPLOG. The *help* facility is optional. There are no window facilities. On the other hand, this simplifies the installation of McPOPLOG, since it does not require any change of the environment.

(7) The user can partly configurate the system, e.g. he/she can form just the POP-11 system with or without backtracking, processes and the like.

The implementation of the Prolog subsystem of McPOPLOG is described in detail in Ref. 6. Here we will briefly review the way of constructing the Prolog compiler of McPOPLOG in terms of POP-11 data structures and control techniques.

First of all, it should be noticed that Prolog terms of McPOPLOG are implemented as the following POP-11 items:[5,8] Prolog constants as POP-11 numbers or words; Prolog variables as POP-11 references; Prolog structures as POP-11 vectors; but the Prolog lists are the same as POP-11 ones.

The implementation of the Prolog compiler of McPOPLOG follows the principal ideas of the Sussex version,[4] but it comprises some additional structures and differs (perhaps) at the level of detailed implementation techniques.

Following the ideas of Ref. 4, our version uses the continuation backtracking as a most suitable vehicle for the Prolog's inference engine, including closures (partly applied functions) and the function-unwinding (chaining) mechanism. The utility of function-unwinding is also used for implementing the 'cut' operator.

Similarly, Prolog variables are implemented as POP-11 references, since it is the simplest representation that is able to process matching of two or more un-instantiated Prolog variables. See Ref. 4 for details.

Prolog predicates are translated to POP-11 functions with the continuation. However, the fundamental routine for unification is written as an ordinary POP-11 function without any continuation, returning `true` (`false`) if its two arguments match (do not match). This saves time as well as memory requirements.

The mechanism of instantiating Prolog variables uses an auxiliary stack for storing instantiated Prolog variables (so-called *Prolog variable stack*). If a Prolog variable is instantiated it is pushed on to the Prolog variable stack. When a branching point of the backtracking is reached the system remembers the current offset of the Prolog variable stack. If a fail occurs, the system re-uninstantiates all Prolog variables that have been instantiated after the branching point has been encountered. The Prolog variable stack is implemented as a POP-11 vector

3. COMMUNICATION BETWEEN POP-11 AND PROLOG

3.1 Top level

There are the following means of communicaiton.

(i) Both POP-11 and Prolog are dialogue, interactive languages, and the user of McPOPLOG can communicate with either subsystem at the top level (when typing on to the terminal, or when a source file is compiled).

(ii) The user can call POP-11 functions from Prolog and vice versa; this inter-language communication can be done to any depth.

When the user is at the top level of the McPOPLOG system, the prompt : is displayed and the system expects any POP-11 statement or any Prolog question. If a text begins with ?– this ?– is recognized as the operator of a Prolog question, otherwise it is considered as a POP-11 statement. Therefore, typing in ?– followed by a Prolog question will activate the Prolog subsystem. For example,

```
: member(2,[1 2 3]) =>
/*POP-11 statement: is 2 a member of*/
/*the list [1 2 3] ? */
** true
/*answer of POP-11 subsystem*/
: ?- read(X), write(X), nl.
/*Prolog question*/
| maria.
/*term read by read(X)*/
maria
/*printed by Prolog's write(X) */
X = maria
/*instantiated variable displayed*/
yes
/*answer of Prolog subsystem*/
:
/*prompt of McPOPLOG is waiting for*/
/*next statement and/or question*/
```

Prolog clauses can be asserted into the Prolog database at the top level using the standard procedure `consult(user)` or `[user]`.

POP-11 statements and Prolog questions and clauses can also be used together in a file. For example, let the file `fred.p` contain

```
function alpha(x,y);
    sqrt(x*x + y*y)
end;
/*POP-11 function*/
?- [user].
```

```
/*compile the following Prolog
clauses*/
member(X, [X] ¦ L]).
member(X, [_¦ T]) :- member(X, T).
?- end.
/*end of Prolog source subfile*/
function beta(x,y);
    sqrt(x * y)
end;
/*POP-11 function*/
```

When compiling the above file using the POP-11 statement `compile('fred.p');` one will get the compiled POP-11 functions `alpha`, `beta` and assert the above Prolog clauses for the procedure `member` into the Prolog database.

3.2 Calling POP-11 functions from Prolog

The user of McPOPLOG can call any POP-11 function from within Prolog. For that purpose, semantics of the standard Prolog operator `is` has been extended within the system McPOPLOG, and two additional unary procedures `popval`, `popvalfail` have been supplied.

`popval(P)` evaluates the Prolog term `P` as a POP-11 expression. The expression `P` must not return any result. This goal succeeds only once.

`popvalfail(P)` evaluates the Prolog term `P` as a POP-11 expression; one result must be returned, and if it is POP-11's `false` it is interpreted as a `fail` of the given goal. This goal succeeds only once.

`X is P` evaluates the Prolog term `P` as a POP-11 expression and unifies the result with `X` ; `P` must return just one result.

The Prolog term `P` in the above procedures (`popval`, `popvalfail` , `is`) is written in Prolog syntax because it is the Prolog compiler that reads this text. However, it is evaluated as a POP-11 expression. Here are the rules for *POP-11 evaluation of Prolog terms* (the rules (iii), (iv) will be explained later):

(i) if `P` is a constant (or an instantiated Prolog variable), the constant (value of the Prolog variable) is pushed on to the POP-11 user stack;

(ii) if `P` is an unstantiated Prolog variable its POP-11 representation (a reference) is pushed on to the POP-11 user stack (see the representation of Prolog terms as POP-11 items);

(iii) `closure(P)`, where `P` is a Prolog term, is evaluated by pushing `P` on to the POP-11 user stack;

(iv) `V -> A`, where `V` is an instantiated Prolog variable and `A` is a Prolog atom, is evaluated as follows: the value attached to the Prolog variable `V` is assigned to POP-11 variable `A`;

(v) Prolog structure `f(P1,....,Pn)` where `f` is n-ary functor and `P1` to `Pn` are Prolog terms, is evaluated as follows: firstly `P1` to `Pn` are evaluated as POP-11 expressions and then the POP-11 function `f` is called.

(vi) Prolog structure `P1 op P2` where `op` is an infix Prolog operation and `P1` , `P2` are Prolog terms, is evaluated as follows: `P1` , `P2` are evaluated as POP-11 expressions and then the POP-11 operation `op` is called. Similarly for prefix and postfix operations.

Thus the *POP-11 evaluation of Prolog term* `P` matches the standard evaluation of POP-11 expressions in programming language POP-11.[8]

Note: the *POP-11 evaluation of Prolog term* `P` does not affect the POP-11 user stack because `popval(P)` must not return any result; `P` of `popvalfail` returns just one result that is popped from the stack, and its value decides the success or failure of the goal; in case of `X is P` the value of `P` is popped from the stack and unified with `X`. If *POP-11 evaluation* of any Prolog term does not return the required number of results, a run-time error occurs.

Example. Define the Prolog procedure

`concat(L1,L2,L3)`

(`L3` is a concatenation of the lists `L1`, `L2`) for the istantiated `L1` and `L2`. We will use the standard POP-11 operation `<>` for lists concatenation:

```
?- op(33,xfy,<>).
/*POP-11 operation <> declared as a
  Prolog operation, McPOPLOG's is has
  precedence 40 */
concat(L1,L2,L3):- L3 is L1 <> L2.
```

Now

```
?- concat([1,2], [a,b], R).
R = [1, 2, a, b]
yes
```

However, the question

```
?- concat([1,2], [aa(1),bb], R).
```

would cause an error since the entire argument `[1,2] <> [aa(1),bb]` would be considered as a POP-11 expression so that a non-existing POP-11 function `aa` would be called with the argument `1` . To prevent a subexpression of an argument of the procedures `popval`, `popvalfail` , `is` from being considered as a POP-11 expression one has to *close* it using the functor `closure` as follows:

```
concat(L1,L2,L3) :- L3 is closure(L1)
                     <> closure(L2).
```

Now `L1` , `L2` are considered as Prolog terms rather than POP-11 exprssions, see rule (iii). Thus

```
?- concat([1,2], [aa(1),bb], R).
R = [1, 2, aa(1), bb]
yes
```

will return expected results.

Example. Consider Prolog procedure `member(X,L)`: `X` is an element of the list `L` . If the procedure is used only for instantiated `X` and `L` , then – in some cases – it could be defined in terms of the standard POP-11 function `member`:

```
member(X,L) :-
       popvalfail(member(closure(X),
       closure(L))).
```

To process a value of a POP-11 variable within a Prolog program the standard POP-11 function `valof` (value of a variable) has to be used:

```
: vars list;
/*delare a POP-11 variable*/
: [10 9 a ab aaa 0] ->list;
/*assign the given list to it*/
:
/*now sort list by Prolog's sort */
```

If we wrote

```
?- List is list, sort(List,S).
```

the Prolog variable List would be instantiated to the atom list . To introduce a value of the POP-11 variable list , we must do:

```
: ?- List is valof(list), sort(List,S),
  write (S), nl.
[0, 9, 10, a, aaa, ab]
/*printed by write(S), nl */
List = [10, 9, a, ab, aaa, 0]
S = [0, 9, 10, a, aaa, ab]
/*instantiated variables displayed*/
yes
```

The evaluation of valof is embodied in the rule (v).

A POP-11 expression in popval , popvalfail , is can also contain an assignment to a POP-11 variable, using the POP-11 operator −> , see rule (iv).

Example.

```
: vars list, sorted;
: [10 9 a ab aaa 0] ->list;
/*sort list by Prolog's sort */
:
: ?- List is valof(list),
  sort(List,S), popval(S ->sorted).
/*result of sort assigned to */
/* POP-11 variable sorted */
List = [10, 9, a, ab, aaa, 0]
S = [0, 9, 10, a, aaa, ab]
yes
: sorted =>
/*POP-11 variable sorted is the */
/* sorted list*/
** [0 9 10 a aaa ab]
```

Thus, thanks to valof and −> we can use POP-11 variables as *inputs* to Prolog programs, as well as *outputs* for storing results.

3.3 Prolog called by POP-11 functions

Similarly, any POP-11 function can call a Prolog program by introducing a Prolog question in its body. When a Prolog question is executed within a POP-11 function the Prolog subsystem does not wait for user's replies; but returns true (false) as a result, if the Prolog question has succeeded (has failed); all Prolog variables in the Prolog question will retain their values when returning from the Prolog question to the POP-11 function so that they can be processed as any other POP-11 variable.

Example. Define POP-11 function sort(list) using the standard Prolog procedure sort(L1,L2):

```
function sort(list) => R;
/* R is explicit result of function*/
?- List is valof(list), sort(List,R).
/* R retains its value from Prolog */
/* question*/
end;
```

Now, the POP-11 expression sort([10 9 ab aaa 0]) yields

```
[0 9 10 a aaa ab]
```

Example. Define POP-11 function member2(x,l) using the Prolog procedure member(X,L) defined as

```
member(X, [X|_]).
member(X, [_|T]) :- member(X,T).
```

One solution is:

```
function member2(x,l);
  if ( ?- X is valof(x), L is valof(l),
  member(X,L). ) then true
  else false
  endif
end;
```

We can also utilise directly a result returned by the Prolog question:

```
function member2(x,l);
  ?- X is valof(x), L is valof(l),
  member(X,L).
end;
```

That's it! If the goal member(X,L) succeeds then true is returned, otherwise false . Thus true / false becomes the result of the POP-11 function member2 .

4. CONCLUSION

It was not straightforward procedure to compare both versions of McPOPLOG because we had at disposal the executable (binary) file of the Sussex McPOPLOG and Ref. 4 only (not a source file, nor any detailed information). Therefore we have run both simple questions like

```
?- member(100, [1,2,3, ... ,100]).
?- functor (T,aa,100).
```

and complex tasks like 'N queens' problem, Waltz algorithm of scene analysis, an expert system shell, etc. We have found that the timings depend on the percentage of calls of standard and user-defined procedures. As a result of the comparison of both actual implementations of POPLOG (McPOPLOG, version 1.2, and Sussex POPLOG, version 11) we have found out that the McPOPLOG is about 3 times faster in compilation and twice as fast in execution.

We can see that the communication between POP-11 and Prolog in the McMaster version of POPLOG (McPOPLOG) is quite straightforward and simple. Neither programming language has been changed, so a user not familiar with POP-11 can use the Prolog subset only, and vice versa. Only three new predicates (popval, popvalfail , closure) have been added to the Prolog subset, and semantics of Prolog operator is have been slightly extended.

Numerous programs written in McPOPLOG at McMaster University so far (e.g. decision-supporting system for diagnostics in neurology, an expert system shell, structural learning system with a statistical support) have indicated that the communicaiton means of McPOPLOG as discussed above are quite adequate.

Further work in this area can include the following directions.

(1) Both POP-11 and Prolog have their pattern-matching mechanisms and databases. It would be useful if both subsystems could share the same database. However, such a facility would enormously extend the set of the communication routines between the two subsystems.

(2) Both languages have their backtracking mechanisms, and at present they cannot communicate, i.e. a POP-11 function cannot explictly fail a Prolog goal, and

vice versa. However, the communication between the POP-11's backtracking and the Prolog's one would increase the power of the ensire system. For example, the POP-11 function

```
func ion collect(X,G) => Result;
  nil ->Result;
  if ( ?- call(G). ) then
    append(X,Result) ->Result;
/*add X to the list Result */
    fail
/* ... would fail the goal G */
  endif
end;
```

would return the list `Result` of all the objects `X` so that the Prolog goal `G` would be satisfied (i.e. it is an analogy to `findall(X,G,Result)`). This extension seems to be promising, and no new procedure or predicate will have to be added to the McPOPLOG system.

The McPOPLOG runs currently on VAX systems and on systems based on the Motorola 68000 processor. Detailed information on the system can be obtained from the author.

REFERENCES

1. C. Muller, Modula-Prolog: a software development tool. *IEEE Software*, pp. 39–45 (Nov. 1986).
2. J. A. Robinson and E. E. Sibert, LOGLISP: an alternative to Prolog. In *Machine Intelligence* 10. Ellis Horwood, New York (1982).
3. H. J. Komowski, QLOG – the programming environment for Prolog in LISP. In *Logic Programming*, edited K. L. Clark and S. Tarnlund. Academic Press, New York (1982).
4. C. Mellish and S. Hardy, Integrating Prolog in the POPLOG environment. In *Implementations of Prolog*, edited J. A. Campbell. Ellis Horwood, New York (1984).
5. I. Bruha, *Reference Manual of McMaster POPLOG*. Technical report 87-03. McMaster University, Department of Computer Science and Systems (1987).
6. I. Bruha, *Compact Implementation of Prolog as a Part of the Environment of McMaster POPLOG*. Technical report 87-05, McMaster University, Department of Science and Systems (1987).
7. R. Burstall, D. Collins and R. Popplestone, *Programming in POP-2*. Edinburgh University Press (1971).
8. R. Barrett, A. Ramsay and A. Sloman, *POP-11 – a Practical Language for Artificial Intelligence*. Ellis Horwood, New York (1985).

Book Review

CD-ROMS in Print: an International Guide to CD-ROM, CD-I, CDTV & Electronic Book Products
Meckler, London 1992. ISBN 0891-8198, ISSN 0-88736-780-1. Print edition, £40; CD-ROM edition, £59; joint price, £79.

We are all now familiar with the music CD, and there can be few in the computing profession who are not also familiar with the CD-ROM (compact disc read only memory) as a means of storage for large quantities of data. Over the past decade the rapid appearance of one standard reference book after another in CD-ROM format is having a significant impact on the librarian's approach both to collection development and exploitation. CD-ROMs allow not only the storage of large amounts of data within a very small space (e.g. the 3-volume set of *Kompass UK* reduced to one 4.72″ disc), but also allow multiple access to that data through the sophisticated search software available on many CDs. The type and range of CD-ROMs is now huge – dictionaries, encyclopedias, directories, catalogues, abstracting publications, financial records, statistical data, works of literature, children's books... many of which will allow you to search the entire database, to sort, download and even analyse the data presented. They can make commercially produced databases, normally available on-line, readily accessible to a public readership without the need for complex dial-up telecommunications links. Unfortunately CD-ROMs do not offer a cheap option. Prices do not appear to have come down with greater use, as initially expected. Inevitably a CD-ROM will cost several times the price of an equivalent printed copy, and in addition a workstation is needed in order to 'read' the 'automated' book.

Now in its fifth year, the 1992 edition of *CD-ROMs in Print* provides details of nearly 3000 commercially available titles, compared with the 300 listed in 1989. The print edition comprises an alphabetical listing of titles. In each case six sections of data are given: title; data provider, publisher and distributor details; compatible drives; disc software and computer hardware/software requirements; pricing structure; brief description of the product. The information can be approached through a series of indexes covering data provider, publisher, distributor, software provider and subject. The latter is extremely useful for anyone seeking to identify products within a specific subject field, although the subject headings used are rather broad. The subject 'Computer Software' lists some 180 diverse titles. The data has been collected over the past 12 months from information provided by publishers and software houses.

The CD-ROM version contains identical data, organised into 2 linked files – a titles file, comprising description, compatibility, software/hardware requirements and pricing structure; and a companies file providing details of information and software providers. Each file can be searched through a number of searchable index fields or through a global index. The search software is easy to use, with search options presented on the screen. The software allows single-concept searches such as finding all CDs on a particular subject, or retrieving details of a specific CD-ROM title. In addition, complex searching, combining a number of concepts, is easily undertaken for more specific requirements like a German dictionary or a chemistry CD that runs on a Macintosh. This CD-ROM itself demonstrates the flexibility of the medium, allowing the type of multi-concept searching not available through the single-concept indexes of the print version.

Whether in print or CD-ROM format, *CD-ROMs in Print* provides a comprehensive source of information on currently available titles and, as with previous editions, will prove to be an indispensable tool in identifying titles and checking source data.

J. MAK
London