PROLOG –
the language and its implementation compared with Lisp

David Warren
University of Edinburgh

Luis Pereira & Fernando Pereira
National Civil Engineering Laboratory
Lisbon

Our Work

What?

Have implemented
a Prolog compiler (and interpreter)
for the DECsystem10 machine
(written in Prolog).

Why?

.....

The Prolog Language

- "programming in logic"
- developed by Colmerauer et al.,
  University of Marseille
- noteworthy for:
  - fast, error-free programming
  - clear, readable, concise programs

<u>Prolog dispenses with</u>:

- <u>goto</u>
- <u>do</u>, <u>for</u> or <u>while</u> loops
- assignment
- references (pointers)

cf. pure Lisp

Viewpoint: Prolog as a generalisation of pure Lisp

- functions → general procedures
- lists → general tree structures
- function evaluation → procedure invocation
- constructors + selectors → pattern matching
- lambda calculus → classical logic (subset)

Elementary Data Objects ("terms")

- "atoms"

  a   nil   tom   n1

- "integers"

  0   -1   99999

- "variables"

  X   Y2   Jim   C1

Complex Data Objects ("terms")

"functor" ie. record-type	"arguments" ie. fields of the record	alternative form
point	(0, 10, −10)	
pair	(romeo, juliet)	
+	(a, b)	a + b
·	(Head, Tail)	Head · Tail
·	(1, ·(2, ·(3, nil)))	1·2·3·nil

Procedure Call ("goal")

"predicate" ie. procedure-name	"arguments"	alternative
gives	(tom, apple, teacher)	
reverse	((1·2·3·nil), List)	
<	(X, Y)	X < Y

Program Statements ("clauses")

interpretations

- "non-unit clause"

  $\mathbb{P}$:- $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{S}$.

  $\mathbb{P}$ is true if $\mathbb{Q}$, $\mathbb{R}$ and $\mathbb{S}$ are true.

  To satisfy goal $\mathbb{P}$, satisfy goals $\mathbb{Q}$, $\mathbb{R}$ and $\mathbb{S}$.

- "unit clause"

  $\mathbb{P}$.

  $\mathbb{P}$ is true.

  Goal $\mathbb{P}$ is satisfied.

- "question"

  ?- $\mathbb{P}$.

  Is $\mathbb{P}$ true?

  Satisfy goal $\mathbb{P}$.

Variables in Clauses

NB. The "lexical scope" of a variable is limited to a single clause.

clause	some possible readings
bird(X):- crow(X).	Any X is a bird if X is a crow.
	To find an X which is a bird, find an X which is a crow.
helps(X,X).	Everyone helps himself.
	The goal of finding a person who helps X is satisfied by X himself.
?- employs(X,X).	Does anyone employ himself?
	Find an X who is self-employed.

Geographical Statistics Example

```
pop (china, 825).        area (china, 3380).
pop (india, 586).        area (india, 1139).
pop (ussr, 252).         area (ussr, 8708).
pop (usa, 212).          area (usa, 3609).
 :

density(C,D) :- pop(C,P), area(C,A), D is (P×1000)/A.

similardensities (C1, C2) :-
      density (C1, D1), density (C2, D2),
      D1 ≥ D2, D1 × 20 < D2 × 21, C1 ≠ C2.
```

Using the Geographical Statistics Program

Question	Answer
?- similardensities(france,china).	yes
?- density(france, X).	X = 246
?- similardensities(X, pakistan).	X = indonesia
?- similardensities(pakistan, X).	no
?- similardensities(X, Y).	

X = indonesia	Y = pakistan
X = uk	Y = w_germany
X = italy	Y = philippines
X = france	Y = china
⋮	

The Declarative Semantics of Prolog

Prolog program

$\mathbb{P} :- \mathbb{Q}, \mathbb{R}.$

$\mathbb{P} :- \mathbb{R}, \mathbb{S}.$

$\mathbb{R} :- \mathbb{S}.$

$\mathbb{S}.$

$\longrightarrow$

its (declarative) semantics

$\vdots$

$\mathbb{P}_1.$

$\mathbb{P}_2.$

$\mathbb{P}_3.$

$\vdots$

$\mathbb{R}_1.$

$\mathbb{R}_2.$

$\vdots$

$\mathbb{S}_1.$

$\mathbb{S}_2.$

$\vdots$

The Declarative Semantics, Precisely Stated

A goal is <u>true</u> if it is the head of some clause instance and each of the goals (if any) in the body of that clause instance is true.

<u>where</u>

An <u>instance</u> of a clause (or term) is obtained by substituting, for each of zero or more of its variables, a new term for all occurrences of the variable.

The Procedural Semantics of Prolog

- Execute a goal by matching* it against the head of a clause and then executing the clause's body.
- Execute goals in left-to-right order.
- Try clauses in top-to-bottom order.
- If a match can't be found, "backtrack".

* Matching ("unification") find's the most general common instance of goal and clause head.

A Conventional Program for the 'similardensities' procedure

Case: ?- similardensities (C1, C2).

```
for C1 from 1 to N do
    integer D1 := density (C1);
    for C2 from 1 to N do
        integer D2 := density (C2);
        if D1 ≥ D2 and D1×20 < D2×21
                   and C1 ≠ C2
        then output (C1, C2)
    repeat
repeat
```

Concatenating Lists

Lisp:

```
append[L1;L2]=
   [null[L1] → L2;
    T → cons[car[L1]; append[cdr[L1]; L2]]]
```

Prolog:

```
append(nil,L,L).

append((X·L1), L2, (X·L3)) :-
          append(L1,L2,L3).
```

Different Uses of the 'append' Procedure

```
?-  append((a.b.c.nil),(d.e.nil),X).

?-  append(X,Y,(a.b.c.d.e.nil)).

member(X,L) :-  append(L1,(X.L2),L).
```

Differentiating an Algebraic Expression

```
d(U+V, X, DU+DV) :-
        d(U,X,DU), d(V,X,DV).

d(U×V, X, DU×V+U×DV) :-
        d(U,X,DU), d(V,X,DV).

d(X,X,1).

d(C,X,0):- atomic(C), C≠X.
```

The Equivalent Differentiation Procedure in Lisp

```
DERIV (LAMBDA (E X)
    (COND ((ATOM E) (COND ((EQ E X) 1) (T 0)))
    ((EQ (CAR E) (QUOTE PLUS))
        (LIST (CAR E)
            (DERIV (CADR E) X) (DERIV (CADDR E) X)))
    ((EQ (CAR E) (QUOTE TIMES))
        (LIST (QUOTE PLUS)
            (LIST (CAR E) (CADDR E) (DERIV (CADR E) X))
            (LIST (CAR E) (CADR E) (DERIV (CADDR E) X))))))
```

cf. Weissman's Lisp 1·5 Primer

Special Characteristics of Prolog – Part I

- general record structures
  - no type restrictions
  - no declaration of record types
- pattern matching – replaces selectors + constructors
- more flexible procedures
  - multiple outputs
  - multi-purpose procedures
  - generate multiple alternative results
- backtracking ≡ iteration
- "logical variable" ≡ assignment + references

Special Characteristics of Prolog – Part II

- program and data expressed the same way
- declarative semantics in addition to the usual procedural semantics
- the procedural semantics is totally defined

Prolog supported by a practical interactive system

- Built-in procedures for
  - arithmetic
  - input-output
  - file handling
  - maintaining an in-core "database"
  - state saving
- Debugging aids
  - tracing
  - interrupts
  - on-line program amendment
- Extra operator for limiting the generation of alternatives

Some Applications of Prolog – large-scale programs

• natural language understanding	Colmerauer + al.
• algebraic symbol processing	Kanoui + Bergman Bundy + al.
• architectural design	Markusz
• drug design applications	Darvas + Futo
• compiler writing	Warren + al.

Prolog Implementations

When	Who	Where	What	How
1972	Roussel	Marseille	interpreter	Algol-W
1974	Battani + Meloni	"	"	Fortran
1975	Szeredi	Budapest	"	CDL (Koster)
1976	Bruynooghe	Leuven (Belgium)	"	Pascal
1976	Roberts	Waterloo (Canada)	"	360/370 assembler
1976	Warren + Pereira + Pereira	Edinburgh + Lisbon	compiler → DEC10	Prolog

Innovations in Our Implementation

- Compilation
- Storage economy (with "structure sharing")
  - conventional stack mechanism to reclaim "local" variables
  - garbage collector for "global" variables
  - "mode declarations"
- Indexing of clauses

  (computed goto replaces sequence of tests)

Compiling the 'append' Procedure (first clause)

append(	**begin**
	local variables L1, L2;
	global variables X, L3;
• (	check term[1] is of type "•";
X,	X := car(term[1]);
L1),	L1 := cdr(term[1]);
L2,	L2 := term[2];
• (X, L3)):-	term[3] :≈ cons(X, L3);
append(L1, L2, L3).	**call** append(L1, L2, L3);
	return;
	end

Compiling the 'append' Procedure (second clause)

```
append(          begin
                     temporary variable L;
   nil,              if term[1] is a variable
                         then term[1] :≈ nil
                         else check term[1] is "nil";
   L,                L:= term[2];
   L).               match (term[3], L);
                     return
                 end
```

Comparative Performance Data

– time ratios for compiled code on DEC10

	Prolog	Lisp*	Pop-2
list concatenation	1	·64	3·8
differentiation	1	1·3	2·3
	1	1·7	3·7
	1	2·6	5·4
geographical stats	1	–	1·6

* Stanford, + NOUUO option.

Conclusion I

Pattern matching is better than selectors + constructors.

- More readable.
- Faster!
  - procedure argument passing not just "red tape"
  - each component only selected once
  - all components selected in a single process
    - no reloading of index registers
    - no duplication of type checks
  - ("structure-sharing" gives faster "cons")

Conclusion II

In principle, Prolog supersedes Lisp ...

- More readable
  - smaller program units
  - less complex (nested)
  - natural declarative reading
- General record structures replace lists
- General procedures replace simple functions
- Pure Prolog provides high-level substitutes for
  - iteration
  - assignment
  - references
- Prolog is practically as efficient as Lisp