CAAD: Education-Research and Practice
ECAADE Conference 1989
School of Architecture in Aarhus Denmark

DO YOU NEED WEAPONS TO KEEP OUT OF "ARTICHAUD MELANIE" OR HOW TO TEACH PROLOG PROGRAMMING TO CAD SYSTEM DESIGN STUDENTS

Marius Frégier

Architect / Computcr Scentist.

Labo GAMSAU (UA CNRS N° 1247)
Ecole d'Architecture de Marseille-Luminy
70 Route Léon Lachamp
13288 MARSEILLE CEDEX 9,
France.
Tel: 33 (91) 41 11 85 or 33 (42) 04 30 05

Keywords:

Education in CAD system construction, graphical extensions to prolog, experts systems, graphical interactive capture of data, intelligent graphic.

Abstract:

A course aimed on the use of prolog for studying, prototyping and developing CAD systems is presented. This course is based on a practical training. Its objectives, topics, teaching method and applications are briefly introduced . Exercises focussed on interests and capabilities of CAD designers are presented. These exercises follow a progression which integrate step by step, different aspects of the application fields. At list these exercises lead to a single application concerned with intelligent graphic.

Introduction:

During the last decade, a lot of sub-problems of CAD have been resolved; but the main problem: a systematic continuation from the preliminary design till the dimensioning of the structural elements, remains unsolved. Solutions to this problem suppose the encapsulation of a lot of skills and knowledge in a whole system. Such an encapsulation requires concentration of numerous expertises, efficient cooperation and time. Education is an answer to the concentration of knowledge and cooperation within time. Fast prototyping seems to be a solution to short encapsulation time of a lot of knowledge.

A joint study program granted by the EEC has been developed in this context [EEC 87]. This program aims to set up a basic course package which integrates teachings of the related skills and knowledge. Prolog seems to be a good candidate both for fast prototyping and knowledge representation. Nevertheless the quite theoretical teaching style of prolog doesn't feet the needs CAD system design learning.

The "Artichaud Melanie" of the equilibrated meal [GIA 85] was widely disseminated all around the world to illustrate the basic prolog concepts. Nevertheless it remains far away from the preoccupations of CAD design students. The prolog course presented here aims to adapt prolog teaching style to CAD system design studies. This course was developed in the context of the joint study program mentioned above.

Objectives of the course:

- initiation in prolog with graphic extension.
- understanding fundamental difference between declarative and imperative programming, inconvenients and advantages of both.
- some fundamental on optimization of time or space critical prolog predicates.
-Overview of possible fields of prolog applications in CAD.

Applications:

logic and data-base, logic and knowledge representation and management, logic and expert systems, logic and pattern recognition, shapes grammar programming [BUE 89], logic and declarative graphic CAD system design and development.

Teaching method:

Generally prolog is taught with emphasis on its theoretical foundations. This may be attributed to considerations resulting from the high technological aspects of 5th generation computer. On the contrary this course teaches prolog like a true programming language in both the aims of prototyping and developing applications.
All along the course, examples and application exercises concerning theoretical concepts are introduced and integrated in a single application concerned with intelligent graphic: the development of an expert system for interactive graphical handling and recognition of geometric patterns.

Emphasis is done on the following particular features of the language:

- language of specification versus programming language,
- non-typed data and declaration lacking,
- affectation lacking versus unification (notions of free and linked variables),
- no distinction between data and programs.

Topics of the course

Introduction and overview about prolog:
- origins, - specificity, what is and what isn't prolog,

The syntax of prolog II:
- identifiers, variables and constants, terms, predicates and clauses,
- facts and rules.

Concepts of prolog:
- difference between imperative and declarative programming,
- trees and lists as basis elements, - the unification process,.
- self dynamic modification of prolog programs.

Natural processing of lists and trees:
- formulation of numerous non-deterministic list predicates.
- optimizing lists manipulation using cross recursive predicates.
- mixing trees and lists to program recursive structures.

Predefined predicates:
- for controlling the execution,. - for arithmetic operation,
- for string manipulation, - for exchanging with the environment.
- for graphical capture and representation,

Capabilities of prolog to do deterministic and non-deterministic programming:

- transforming deterministic predicates into non-deterministic ones.
- deterministic programming using the *cut* symbol to suppress backtracking.
- controlling program execution by using the predefined co-routined (prolog II only) *dif* predicate versus the *cut* symbol.
- controlling program execution by using the predefined predicates "free" and "bound".
- using the predicate *freeze* for putting back certain decisions until the values need for their evaluation are known, both to get advantages of declarative programming and to keep an efficient execution.
- dynamic self modification of prolog programs using *assert* and *suppress* predicates.

Advance programming techniques:
- negation by failure. - the saturation process.
- finding all the solution before backtracking.

Advance data structures:
- binary trees. - optimal binary trees.

introducing some cookbook predicates:
- find-all. -suppress-all. -suppress-and-assert.
- plus some introduced during the practical training

Understanding and correcting error:
- some on the thumb rules and techniques are introduced here and studied during the practical training.

The practical training (support of the practical training)

<u>Exercise 1</u>

Write a Facts base and the rules which checks if a polygonal figure is a lozenge a rectangle or a square. To build the facts base you are suppose to know the following facts concerning the geometrical figures.

```
is-a-polygon(name-of-a-figure)
has-four-edges(name-of-a-figure)
all-edges-are-equal(name-of-a-figure)
all-angles-are-equal(name-of-a-figure);
```

Exercise 1 (a solution):

```
is-a-square(a-figure)->
                     is-a-polygon(a-figure)
                     has-four-edges(a-figure)
                     all-edges-are-equal(a-figure)
                     all-angles-are-equal(a-figure);

is-a-lozenge(a-figure)->
                    is-a-polygon(a-figurere)
                    has-four-edges(a-figure)
                    all-edges-are-equal(a-figure);


is-a-rectangle(a-figure)->
                    is-a-polygon(a-figure)
                    has-four-edges(a-figure)
                    all-angles-are-equal(a-figure);
```

"Example of a data base or facts base"

```
is-a-polygon(my-figurel)->:
is-a-polygon(my-figure2)->;
is-a-polygon(my-figure3)->:
is-a-polygon(my-figure4)->;

has-four-edges(my-figurel)->;
has-four-edges(my-figure2)->:
has-four-edges(my-figure3)->;

all-edges-are-equal(my-figurel)->;
all-edges-are-equal(my-figure2)->;

all-angles-are-equal(my-figurel)->:
all-angles-are-equal(my-figure3)->;

;End world: Normal
```

<u>Exercise 2:</u> solve the exercise 1 considering the fact that if a quadrilateral is both a lozenge and a rectangle then you can conclude it is a square.

Exercise 2 (a solution)

```
is-a-square(a-figure)->
                    is-a-lozenge(a-figure)
                    is-a-rectangle(a-figure);

is-a-lozenge(a-figure)->
                    is-a-polygon(a-figure)
                    has-four-edges(a-figure)
                    all-edges-are-equal(a-figure);

is-a-rectangle(a-figure)->
                     is-a-polygon(a-figure)
                    has-four-edges(a-figure)
                    all-angles-are-equal(a-figure);

"An example of facts base"

has-four-edges(my-figurel)->;
has-four-edges(my-figure2)->;
has-four-edges(my-figure3)->;

all-edges-are-equal(my-figure1)->;
all-edges-are-equal(my-figure-2)->;

all-angles-are-equal(my-figure1)->;
all-angles-are-equal(my-figure3)->;

;End world: Normal
```

<u>Exercise 3</u>

using the following base of fact

```
is-a-polygon(my-figure1)->;
is-a-polygon(my-figure2)->:
is-a-polygon(my-figure3)->:
is-a-polygon(my-figure4)->;
is-a-polygon(my-figure5)->;

has-four-edges(my-figure1)->;
has-four-edges(my-figure2)->;
has-four-edges(my-figure3)->;
has-four-edges(my-figure4)->;

all-edges-are-equal(my-figure1)->;
all-edges-are-equal(my-figure2)->;

all-angles-are-equal(my-figure1)->;
all-angles-are-equal(my-figure3)->;
```

Modify the program 2 in such a way:

1_add a predicate which checks if a figure is a quadrilateral
2_use this predicate to simplify the following predicates: "is-a-rectangle, is-a-lozenge"

verify if your program works for the following goals

is-a-square(x)	is-a-rectangle(y)	is-a-lozenge(y)
is-a-quadrilateral(y)	is-a-polygon(w)	

exercise 4:

1_modify the program 3 in such a way that you can cheek if a figure is a parallelogram.

suppose that the solutions to the literal: *two-by-two-parallel-edges-of(a-figure)* are known as facts

modify the base of facts properly.

exercise 5:

find an other way to program the predicates: *is-a-square, is-a-rectangle, is-a-lozenge*
Program them by using the predicate is-a-parallelogram plus something else

Use the following predicates: *has-two-adjacent-edges-equal, has-a-right-angle*

Don't program them like facts, but like new predicates, to do this, use the already Known facts.

exercise 5_(solution):

```
is-a-square(a-figure)->                                    (old solution using facts only)
        is-a-lozenge(a-figure)
        is-a-rectangle(a-figure):

is-a-square(a-figure)->                                    (new solution)
        is-a-parallelogram(a-figure)
        has-a-right-angle(a-figure)
        has-two-adjacent-edges-equal(a-figure);

is-a-rectangle(a-figure)->                                 (old solution using facts only)
        is-a-quadrilateral(a-figure)
        all-angles-are-equal(a figure);

is-a-rectangle(a-figure)->                                 (new solution)
        is-a-parallelogram(a-figure)
        has-a-right-angle(a-figure);

is-a-lozenge(a-figure)->                                   (old solution using facts only)
        is-a-quadrilateral(a-figure)
        all-edges-are-equal(a-figure);

is-a-lozenge(a-figure)->                                   (new solution)
        is-a-parallelogram(a-figure)
        has-two-adjacent-edges-equal(a-figure);
```

```
 is-a-quadrilateral(a-figure)->
                    is-a-polygon(a-figure)
                    has-four-edges(a-figure);

 is-a-parallelogram(a-figure)->
                    is-a-quadrilateral(a-figure)
                    two-by-two-parallel-edges-of(a-figure);

has-two-adjacent-edges-equal(a-figure)->                    (new predicate using facts only)
                    all-edges-are-equal(a-figure);
has-a right-angle(a-figure)->                               (new predicate using facts only)
                    is-a-quadrilateral(a-figure)
                    all-angles-are-equal(a-figure);
```

Note that both the solutions can he used concurrently.

Now exercises on structures:

Exercise 6: using the n-uple notation, points and segment can be denoted as follow

point(p1)	*point(<x,y>*	
segment(s)	*segment(<P1,P2>)*	*segment(<<xl,yl >,<x2,y2>>)*

In the same way give four different notations that a structure of a triangle can match.

exercise 6 (a solution)

```
triangle(f)
triangle(< < <x1,y1 >, <x2,y2> >,< <x2,y2>,<x3,y3> >, < <x3,y3>,<xl,yl>>)
triangle(<sl,<pl,p2>,<<x3,y3>,<xl,y1 >>)
```

Before the next exercise, predefined arithmetic predicate where introduced.

exercise 7:

Using the predefined predicates val and add, write the predicate translating the triangle f by the translation t, use the structure define above and represent the translation t by the tuple <x,y>.

exercise 7 (solution):

```
triangle-translation( t , < s1,s2,s3 >,< s1',s2',s3'>) ->
                    segment- translation (t, s1,s1')
                    segment-translation (t,s2,s2')
                    segment-translation (t,s3,s3');

segment -translation (t,<pl,p2>,<pl',p2'>) ->
                    point-translation(t,pl,pl')
                    point-translation(i,p2,p2');

point-translation(<x,y>,<x',y'>) ->
                    val(add(x,x').x")
                    val(add(y,y'),y");
```

Now (only!) the list notation is introduce.

<u>exercise 8:</u> using the list notation, generalise the translation of a triangle to the translation of a polygonal figure. denote the polygonal figure as a list of segments.

exercise 8 (solution):

```
figure -translation (t, ni, nil) - >;
figure-translation (t,s.f, s'.f)- >
                    segment-translation (t,s,s')                    (define above)
                    figure -translation(t,f,f');
```

<u>exercise 9:</u>

Suppose you don't know about the facts two-by-two-parallel-edges-of (my-figure) but you know the coordinates of the four points of a square and the structural topology of a quadrilateral.

Find a general logical representation of' the structure and the program which check whether or not a polygonal figure is a square.

Use the list structure representation and the predicates of the solution to exercise 5.

This stage the students know how to efficiently represent geometrical figures in prolog data structures, they are also able to do some manipulation on them.

The exercises following this stage introduce the students to the techniques of graphical interactive capture of data within prolog.

This is the occasion both to point out how prolog is inadequate to treat deterministic process, and to investigate ways to circumvent difficulties resulting from this.

These exercises use greatly the predefined graphical predicates of prolog II. The students experiment how to build and encapsulate high level graphical interaction within prolog predicates. The speed of the resulting processes is fast enough to produce an efficiently use.

<u>exercise 10:</u> using the predefined predicates gr-click, gr-getmouse, gr-lineto, gr-moveto and gr-mode, program the rubber-band function.

exercise 10 (solution):

```
 rubber-band(<x0,y0>,<xl,yl>)->
                    gr-getmouse(x2,y2)
                    draw-segment-xor(<x0,y0>, <xl,yl>)
                    draw-segment-xor(<x0,y0>,<x2,y2>)
                    rubber-band(<x0,y0>,<x2,y2>);

 draw-segment-xor(<x,y>, <x',y'>)->
                    gr-mode(0,10)
                    gr-moveto(x,y)
                    gr-lineto(x',y')
                    gr-mode(0,8);
```

exercise 11:

Using the rubber-band function program the predicate get-segment(<pl,p2>).
problem: how to stop the process, propose solutions.

```
 get-segment(<x0,y0>,<xl,yl>)->
                  gr-click(<0,x,y)/;

get-segment(<x0,y0>,<xl,yl>)->
                    gr-getmouse(x2,y2)
                    raw-segment-xor(<x0,y0>, <x 1,y1>)
                    draw-segment-xor(<x0,y0>, <x2,y2>)
                    get-segment(<x0,y0>, <x2,y2>);
```

exercise 12:

Using the predicate get-segment (<pl,p2>) program the predicate get-polyline. Use recursive techniques.

exercise 13:

Run the predicate get-segment and move the mouse around without clicking what do you notice. What does that mean. Circumvent this by, using the technique of failure and restarting-goal on the intermediary results. Before to produce the failure the intermediary results by using the assert predicate.

Now students know how to program efficient high level graphical interaction with prolog. By linking the high level graphical interactive predicates with the demonstrative predicates they are able to build a fully graphical interactive 2D geometrical expert system.

exercise 14: Link the interactive 2D shape editor with the graph expert.

exercise 15: and further ....   different kind of improvement and optimization are done

Conclusion:

Capabilities of prolog to integrate knowledge for CAD design have been introduced.
Topics of a prolog course oriented CAD design have been presented.
Solutions to exercises well suited to learning of prolog in the aim of CAD system design were presented. Because prolog is really very, well suited to fast prototyping and the course strongly integrated it has been possible to lead students to program the whole Graph Expert system within the duration of a session of 40 hours.

References:

Buelinckx H, Fregier M, *New Concepts for CAD: shape grammar towards truly declarative graphic.* International Conference on CAD & CG, Beijin 10 -12 August 1989.To be published.

EEC 1987 (Buelinckx H. Fregier M. Hemde H. Van Wildcr L.*). Form-Structure Grammars as a tool for Computer Aided Architectural Design.* (CEE JSP 85 - 86 - 87- 417-B)

Gianesini F, Kanoui H, Van Caneghem M, Prolog InterEditions, mars 1985.