A framework for network modeling in Prolog

Zdravko I. Markov
Institute of Engineering Cybernetics and Robotics
Bulgarian Academy of Sciences
Acad.G.Bonchev str. bl.29A, 1113 Sofia, Bulgaria

Abstract

A new formalism for building network models in Prolog is proposed. It is based on a new semantic interpretation of the Prolog variables and clauses, which are used to represent the network links and nodes respectively. The problem of attributed graph model of polyhedral objects is used as an application area, where the proposed methods are examined.

I. Introduction

Graph models are commonly used in AI. Their expressive power makes them suitable in almost all AI fields. However, in the framework of Prolog programming graph algorithms have a restricted meaning of graph searching techniques. Searching graphs is a fundamental technique for solving many AI problems as state-space represented problems, game playing, planning etc. However, there are other operations on graphs not less useful in the context of AI. Such are for example graph isomorphism (graph matching) and constraint satisfying (constraint propagation). Graph models can be viewed more generally as network models, such as neural networks, knowledge representing networks (semantic networks, frames) and many other distributed computational models grouped around the term connectionism.

The present paper deals with a graph model of a class of visual objects polyhedra. The Prolog implementation of the problem inspired a new concept for representing graphs and networks. The proposed representation scheme requires some extensions of the standard Prolog mechanisms, which are introduced in an experimental version of the language developed by the author.

2. Attributed graph representation of polyhedra

The polyhedra are a large class of graphical objects frequently used in geometric modeling and vision. They can be used for modeling curved objects as well by approximation. This class is suitable for object recognition since there exist well developed methods for extracting linear features from real images. This aspect of the problem is discussed in [Engelbrecht and Wahl, 1986]. The Prolog based approach for polyhedron recognition is described in [Markov and Risse, 1988]. In the field of CAD the class of polyhedral objects is used for the 3-D interpretation of 2-D sketches. This problem is recently considered in the framework of network models, which we discuss in the next sections. Here we outline only the basic ideas concerning polyhedral modeling in Prolog, which are used later for building an alternative network model of the same problem domain.

A polyhedron can be considered as a structured collection of its parts - vertices and edges. For example a four-side figure can be expressed in Prolog by the list

[edge(l,2),edge(2,3),edge(3,4),edge(4,l)],

where the structure edge(M,N) represents an edge connecting the vertices M and N. The above list represents a too large class of geometric figures. To obtain a sub-class (e.g. parallelogram, rhombus etc.) we should introduce more geometrical information. In our model this means some constraints i.e.

attributes attached to each edge. For this purpose the following form of the edges can be used:

```
  edge(VI ,V2,Slope .Length)
```

Using this notation a parallelogram can be represented as:

```
  [ edge(I,2,0,20),edge(2,3,30,50),
    edge(3,4,0,20),edge(4,1,30,50) ]
```

The edges of the above figure are separated in 2 groups, distinguished by their slopes and lengths. These groups are implicitly defined by edge attributes having equal values, which we call same-valued classes. A very important feature of this representation scheme is the possibility to define a class of such figures, using variables instead of fixed names or values standing for the vertex names and attributes. The above shown list representing an instance of a parallelogram can be transformed in the following form:

```
  [ edge(A,B,SI,LI),edge(B,C,S2.L2).
    edge(C,D,SI,LI),edge(D,E,S2,L2) ]
```

In this list the equal names or values are replaced by shared variables. Using variables as arc attributes ensures that the class representation is free of any specific geometric properties as size, orientation, etc. The only fact taken into account is the equality or inequality of the attributes, and this is provided by the built-in Prolog unification mechanism. The real values of the attributes can be of any type and measurement units. Since parallelism and same length are properties, preserved by the parallel projection, this class representation can be used for 3-D polyhedra.

Our aim now is using the described representation scheme to define a procedure, which can check whether an instance belongs to a certain class. In the graph terminology this is called graph isomorphism or graph matching. We propose a graph matching procedure based on the following idea: a graph is a subgraph of another graph, if and only if the set of all arcs of the first graph is a subset of all arcs of the second graph. During the subset operation the variables, representing the class nodes and attributes are successively instanti ated to the corresponding instance nodes and attributes. Then the consistency of these instantiations is checked and possi bly backt ra cki ng oc cur s. The wh ol e process is hidden in the recursion used in the definition of the predicate match, which is a slight modification of the predicate subset, described in [Clocksin and Mellish, 1981].

```
match([A!X],Y):-member(A,Y),match(X,Y).
match([],_)--I.

membeKedge(X,Y,S,L),[edge(X,Y,S,L)IJ).
member(edge(X,Y,S,L),[edge(Y,X,S,L)IJ).
member(X,[_IT]):-membeKX,T).
```

The pure subgraph matching problem is NP-complete. However, in some cases a proper representation may be found to make the graph matching algorithm applicable in practice. The aim is to minimize the number of the backtracking steps occurring in the "bad" ordering combinations obtained enumerating the graph arcs. In our case the use of attributes in the graph improves the efficiency as it is shown in [Markov and Risse, 1988]. However, there is a "second-order" problem, which appears in practice where more than one class is used. The overall efficiency in such a case depends very much on the order of the selected classes to be recognized, since a possible matching between the instance and each one of the classes is tested sequentially. A solution of this problem can be found in matching the instance against all classes simultaneously. Such a scheme is proposed in the next section.

3. Database representation of attributed graphs

As we mentioned in the previous section the problem of the different ordering combinations of the edges in the class and in the instance is solved by involving set operations. Besides the lists, the Prolog database is also suitable for storing set elements. An element belongs to such a set if its activation as a goal succeeds. Thus a subset operation is just an execution of a conjunction of goals. For example the following program checks whether an instance of a rhombus belongs to the class of all rhombuses.

```
/" An instance of a rhombus */
edge( 1,2,0,100).
edge(2,3,45,100).
edge(3,4,0,100).
edge(4,1,45,100).

/x The class of all rhombuses "/
?- edge(A,B,SI,L),edge(B,C,S2,L),
   edge(C,D,SI ,L),edge(D,A,S2,L).
```

The above scheme is much more efficient than the corresponding subset operation on lists, since two Prolog built-in mechanisms are directly used - database access and backtracking. However, it is impossible using this representation to check a sub-graph to graph isomorphism (with missed edges in the instance, e.g. hidden edges in a real image). That is a substantial shortage, which makes this approach inappropriate for our model. To avoid that we introduce an extension of Prolog, which allows shared variables among different clauses in the database.

The proposed extension, which we call "net-clause", is syntactically a Prolog structure with the functor ":" and arguments - normal clauses. The clauses contained in a net-clause form a special domain in the Prolog database, which has the same semantic as the normal Prolog clauses, taking into account the special behavior of the variables contained.

A natural semantic interpretation of the proposed net-clause is a network, where the nodes are represented by clauses and the connections - by shared variables. The nodes can be used as processing units and the connections - as channels for information transfer. This interpretation is our basic paradigm, which will be elaborated further in the paper.

The features of the net-clause variables are used to improve the attributed graph model of polyhedra. Besides for representing the topology and geometric constraints shared variables are used to represent the part-of hierarchy between the vertices and the objects they constitute, so that after a successful matching of an instance, the instantiated variables (vertices) indicate the corresponding class, where they belong. The network illustrating this idea is shown in Figure 1.

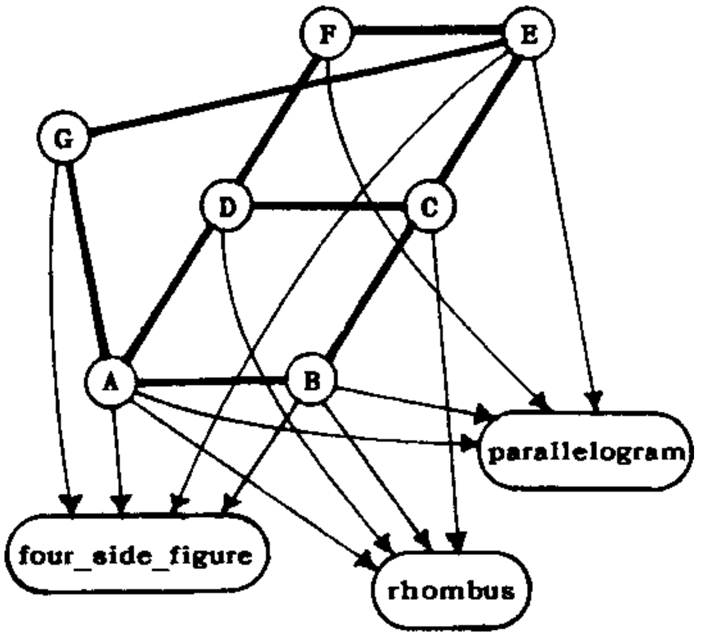

Figure 1. An example network for representation of polyhedra (the topology and the "part-of" links are shown in different line styles)

The following program implements the network from Figure 1. (The use of a special Prolog extension, ensuring all different variables to be bound to different objects, is essential in this example. Such an extension is also available in Prolog III.)

```
/* Edges of the figures "/
edge(A,B,SI,LI):
edge(B,C,S2,LI):
edge(C,D,SI,LI):
edge(D,A,S2,LI):
edge(B,E,S2,L2):
edge(E.F,SI,LI):
edge(F,A,S2,L2):
edge(E,G,S3,L3):
edge(G,A,S4,L4):

/* Classes of figures "/
class(four_side_figure,A,B,E,G):
class(parallelogram,A,B,E,F):
class(rhombus,A,B,C,D).

/* Network activation "/
?- edge(I,2,0,20),edge(2,3,45,30),
   edge(3,4,0,20),edge(4,I,45,30),
   class(Name,X,Y,Z,P),nonvar(X),
   nonvar(Y),nonvar(Z),nonvar(P)-

Name=parallelogram
```

The network from Figure 1 has a shortage inherited from the static behavior of described model. Let us explain this. The facts edge can be considered as inputs and class - as outputs of the network. To obtain an answer we should activate both the inputs and the outputs (call them as goals, specifying the input/output values as arguments). This scheme is good as long as we know the outputs previously. But that is not the case in our example. We enter a pattern (the instance) and then scan all possible classes to find which one is matched. In other words if we have many outputs in the network we should check all of them to find which one is active.

To make the net-clause network more dynamic we extend the semantics of the clauses to local processing units, which are activated on certain conditions. In terms of Prolog this means the possibility to activate the goals in the body of the clauses. At the normal Prolog computation a clause is activated when the current goal unifies its head. In our case there is no current goal, instead there are instantiated variables and we can use them as an indication that some processing unit has enough information to do its work. So, we define the following activation rule: a clause body is activated when the clause head is "enough" instantiated. The quantitative measure of the clause head instantiation is the number of its uninstantiated variables - arguments.

An implementation of the above idea is the built-in predicate activate(N), supported by our experimental Prolog version. It activates all clause bodies, whose heads have less than N uninstantiated variables - arguments. In this way the variables in the head of a clause can be viewed as activating links. We introduce also inhibitory links, represented syntactically as unary minus followed by the variable name, e.g. ~X. Such links are interpreted by the activation procedure as a negation of the instantiation condition, i.e. if X is instantiated, ~X is treated as an uninstantiated variable and vice versa.

The best suited computational scheme for the outlined network model is the parallel one. Since we could not avoid the sequential activation of the clauses we simulate the functional behavior of a parallel execution based on the following features:

(a) Multiple scanning the net-clause database and executing each time all active clauses, so that each clause is executed successfully just once. This scheme ensures a special activation order guided only by the activation rule

(b) Delaying the execution of the active clauses which have failed. This rule is based on the assumption that an active clause may fail due to some conditions in the network, not present at the current pass through the net-clause database. That is a way to synchronize the work of the hidden network units or to organize switching between branches of the network. Functionally the same effect could be reached using inhibitory links. In fact the two mechanisms reflect the two paradigms - logic programming and network modeling, integrated in the proposed formalism.

(c) According to (a) and (b) the clauses are checked repeatedly and those of them which satisfy the activation condition are executed. This process terminates when no more variable instantiations have been done during a pass through the net-clause database.

(d) Executing all active clauses in a common environment. This feature of the activation procedure is very important, since it ensures that all newly established connections between the clauses (variable instantiations) are kept until all active local procedures in all units of the network are executed.

Using the newly defined predicate activate, the program representing the network for recognition of planar figures, shown in Figure 1 can be modified by replacing the facts class with the following clauses:

```
four side figure(A,B,E,G):-
  write('four_side_figure matched'):
parallelogram(A,B,E,F):-
  write('parallelogram matched*):
rhombus(A,B,C,D):-
  write('rhombus matched').
```

Now initiating the network by the question

```
?- edge(l,2,0,20),edge(2,3,45,30),
   edge(3,4,0,20),edge(4,1,45,30),
   activate(O).

parallelogram matched
```

we obtain as an answer the name of the class, which the entered pattern belongs to.

In the 3-D polyhedral model the instance may contain less edges than the class (e.g. hidden edges). In this case the activate predicate should be used with an argument greater than 0, allowing in such a way "fuzzy" matching.

The above examples are in fact one-layer network models where the local processing capabilities of the network are not used. This is because the patterns to be recognized are quite simple. The basic properties of the characteristics specifying them are only two - equality and inequality. However there are some geometric properties, which can not be expressed in terms of equality and inequality. For example such a property is the perpendicularity of edges. To check it some calculations are required, which can be performed by intermediate nodes (hidden units) in the network. Using the edge facts from Figure 1 and adding the following clauses we obtain an extended network, capable to recognize rectangulars and squares too.

```
/* General case of four-side figures */
four_side_figure(A,B,E,G):-
   write('four_side_figure matched*):

/* Calculating perpendicularity */
perpendicular(SI ,S2):-Done-ok,
   (0 is (S1-S2) mod 90, P=ok;true):

/* Non-perpendicular figures */
parallelogram(A,B»E,F,Done,~P):-
   write('parallelogram matched9):
rhombtis(A,BtC,D,Done,~P):-
   write('rhombus matched'):

/* Figures with perpendicular edges */
rectangular(A,B,E,F,P):-
   write('rectangular matched9):
square(A,B,C,D,P):-
   write('square matched9).
```

The clauses representing the classes of parallelogram and rhombus are modified according to the use of the hidden unit perpendicular. They become candidates for activation only after the check for perpendicularity has been done (the variable Done is instantiated). This check actually distinguishes the two groups of classes - with and without perpendicular edges, indicated by the variable P, which is used respectively as an activating or an inhibitory link.

4. Connectionism in Prolog

The described results in network modeling in Prolog may be naturally discussed in the more general framework of connectionism. That is a field focusing a growing interest, characterized mainly by the quite old idea that hard problems could be solved interconnecting simple processing units. Connectionism may be characterized by several features [Arbib, 1987], which we discuss in connection with the features of our network, described in the previous section.

(a) The use of networks of active computing elements, with programs residing in the structure of interconnections. The active computing units in our model are the clauses, activated on certain conditions or unconditionally. The procedure specifying the way the patterns are processed (matched) actually resides in the structure of shared variables (i.e. the structure of interconnections) representing the topology and the geometry of the polyhedra.

(b) Massive parallelism, with no centralized control. The proposed here network formalism is implemented in a sequential computational environment. However it is designed in such a way that the sequential implementation affects minimally the network properties. Although the processing of the entered in the network pattern is based on backtracking, this process is hidden and the result does not depend on the order of the subgoals specifying the pattern. Thus the process of setting up the inputs/outputs of the network can be viewed actually as a parallel one and the one-layer models without hidden units can be considered parallel too.

The features of the activation rule, described in the previous section are in a sense steps toward simulating parallelism in multi-layer networks. The functional behavior of a parallel computation is fully reached when the network satisfies two quite natural requirements:

1. The execution of the active clauses has no "side effects" (in Prolog terminology);

2. The network is consistent. Clauses, which "undo" what have been done by other clauses, and never-fulfilled conditions may be considered as network inconsistencies.

The described network computational scheme can be viewed in the framework of the parallel computation as one without centralized control and synchronization. Actually this is a "shared memory architecture", where the role of the common memory is played by the shared variables in the net-clause.

(c) The encoding network semantics either by single network units (localized representation) or by patterns of activity in a population of such units (distributed representation). The use of the two representations is essential in the described example of a polyhedron recognition network. The semantics of the classes is locally represented by some of the network nodes and the patterns representing the class characteristics are distributed in the interconnections of the network. As it is mentioned in [Arbib, 1987] the distributed representation is based on the principle "letting the features fall where they may", which actually is the main principle of the polyhedron recognition network too.

5. Conclusion

The aim of the present paper is to outline the basic concepts of using Prolog as a framework for building network models. For lack of space we have skipped the implementation details, though they are critical for the applicability of the proposed formalism in practice. In fact the only source of inefficiency could be the activate predicate, which involves some notion of a second-order logic ignoring the clause functors as an access rule. However the activation rule ensures that only the clauses, which could do some "useful" work are activated, and all others are simply skipped. Thus, formally speaking the effectiveness of the net-clause network and a normal Prolog program must be the same, as in both cases only the "useful" clauses are executed (of course, if both models are "enough" adequate to the problem domain).

The network programming in Prolog could be a feasible tool for some applications as vision (e.g. polyhedron recognition, 3D interpretation of 2D sketches etc.), semantic networks, production systems and other schemes for knowledge representation, and natural language processing.

Further research in the above-listed fields of application needs at first deepening of the proposed formalism, which we see focused toward the following topics:

(a) Extending the functionality of the connections allowing not only equality and inequality to be propagated through the shared variables, but some calculations to be performed during the unification (solving equations, constraint satisfying etc.). This means transferring some of the hidden unit semantics to the network links.

(b) Using other activation schemes, e.g. activation by need (when a variable instantiation is required by a processing unit, the clause, capable to instantiate it, is activated).

(c) Developing learning procedures suited to the proposed network representation.

Acknowledgments

I owe special thanks to L. Sinapova for her support in preparing the paper. I am also grateful to D.Dochev and G.Agre for the helpful discussions about network modeling in Prolog.

References

[Arbib, 1987] Arbib M.A. Brains, Machines, and Mathematics (second edition).Springer-Verlag, 1987.

[Clocksin and Mellish, 1981] Clocksin, W.F, C.S.Mellish - Programming in Prolog. Springer-Verlag, 1981.

[Engelbrecht and Wahl, 1986] Engelbrecht,J., F.Wahl - Polyhedral object Recognition using Hough-Space Features. IBM Research Report RZ 1486 (#54038), IBM Zurich Research Laboratory. 1986.

[Markov and Risse,1988] Markov Z., Th.Risse. Prolog Based Graph Representation of Polyhedra. In Artificial Intelligence III, pp.187-194, North-Holland, 1988.