Declarative Programming: Handout 5

Yılmaz Kılıçaslan
November 01, 2023

Key Concepts: *Situtation, Fractal, Recursion, Base Clause, Recursive Clause.*

RECURSIVE LOGIC PROGRAMMING

1 INTRODUCTION

In this lecture, we will see why we require our programs to have the capability of recursion and how we can implement recursive algorithms in Prolog.

2 LOGIC PROGRAMMING AS A WAY OF DESCRIBING A SITUATION

Logic programs essentially serve to describe a **situation**. At the basic level, a situation is a set of facts. Let us illustrate this view with an example of family relationships (adopted from Sterling and Shapiro 1986):

(1) *An Example (from Sterling and Shapiro 1986):*

```
father(terach, abraham).
father(terach, nachor).
father(terach, haran).
father(abraham, isaac).
father(haran, lot).
father(haran, milcah).
father(haran, yiscah).

mother(sarah, isaac).

male(terach).
male(abraham).
male(nachor).
male(haran).
male(isaac).
male(lot).

female(sarah).
female(milcah).
female(yiscah).
```

The predicates used (i.e., *father, mother, male,* and *female*) are chosen as mnemonics (as a convention that we will usually follow) in order to suggest what they denote.

3 CODE-SHORTENING AS A GENERALIZATION STRATEGY

Suppose that everyone in our family likes apples. One way of stating this is the following:

(2) *A List of Facts:*

```
likes(terach, apples).
likes(abraham, apples).
likes(nachor, apples).
likes(haran, apples).
likes(isaac, apples).
likes(lot, apples).
likes(sarah, apples).
likes(milcah, apples).
likes(yiscah, apples).
```

As we said last week, using rules with variables we can summarize such a set of facts (i.e., a set of facts holding for every individual in the domain of discourse). The single-rule below:

(3) *One Single Rule:*

```
likes(X, apples):- male(X); female(X).
```

or, the following pair of rules:

(4) *A Pair of Rules:*

```
likes(X, apples):- male(X).
likes(X, apples):- female(X).
```

can be used as a summary of the facts in (2).

In fact, an inventory of concepts coming with a natural language, too, provides ways of shortening our natural language expressions. For example, if it does not matter whether an individual is female or male, we will refer to him / her as a person. We can transfer this flexibility to our program by encoding the following *person/1* procedure:

(5) *A person/1 Procedure:*

```
person(X):- male(X).
person(X):- female(X).
```

In this way, we will have an even shorter fragment that is semantically equivalent to each of the *likes/2* procedures given above:

(6) *An even Shorter Rule:*

```
likes(X, apples):- person(X).
```

Rules can also allow us to express complex queries in terms of simple ones. If we have the following rules:

(7) *A Set of Rules:*

```
grandparent(X, Y):- father(X, Z), father(Z, Y).
grandparent(X, Y):- father(X, Z), mother(Z, Y).
grandparent(X, Y):- mother(X, Z), father(Z, Y).
grandparent(X, Y):- mother(X, Z), mother(Z, Y).
```

then we can use the query below:

(8) *A Simple Query:*

```
?- grandparent(X, Y).
```

to ask the same question as the following query will serve to ask:

(9) *A Complex Query:*

```
?- ((father(X, Z), father(Z, Y));
    (father(X, Z), mother(Z, Y));
    (mother(X, Z), father(Z, Y));
    (mother(X, Z), mother(Z, Y))).
```

We know that the concept p*arent* can be used as a short way to cover a disjunction, namely *father or mother*. Hence, the code above can be shortened further if we have the following *parent/2* procedure:

(10) *parent/2:*

```
parent(X, Y):- father(X, Y).
parent(X, Y):- mother(X, Y).
```

This will allow us to replace the four rules above for the relation *grandparent* with one single logically equivalent rule:

(11) *One Single Rule:*

```
grandparent(X, Y):-
   parent(X, Z), parent(Z, Y).
```

So far, we have tried to shorten a finite number of alternatives under one cover term (namely *person* covering *male* and *female*, *parent* covering *father* and *mother*; and *grandparent* covering *father-of-a-father*, *father-of-a-mother*, *mother-of-a-father*, and *mother-of-a-mother*). But, can

we (and how can we) encode a possibly infinite number of alternatives? Natural languages have an abundant number of concepts for such cases.

Consider the family relationship *ancestor*. One way of trying to capture this relation is to define a series of rules like:

(12) *A Set of Rules:*

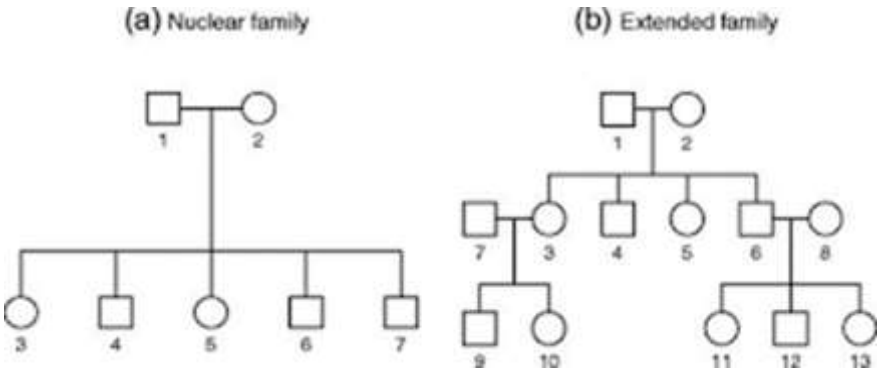
```
grandparent(X, Y):- parent(X, Z), parent(Z, Y).
greatgrandparent(X, Y):- parent(X, Z), grandparent(Z, Y).
greatgreatgrandparent(X, Y):- parent(X, Z), greatgrandparent(Z, Y).
greatgreatgreatgrandparent(X, Y):- parent(X, Z), greatgreatgrandparent(Z, Y).
```

Each of these rules states that X is the ancestor of Y. Thus, the code fragment above is a correct program. However, it is not complete. It would most likely be very far from capturing the whole list of one's ancestors. First of all, our parents are among our ancestors but the clauses do not express this. Secondly, the ancestors older than our parents are defined only up to a certain point, up to our great-great-great-grandparents. Since we are unlikely to know one's all ancestors, we somewhat have to find a way to generate an unlimited number of solutions with a fragment of code, which is, of course, of a finite length. It seems that so-called **fractal** structures provide a promising insight in this direction.

4 FRACTAL STRUCTURES / OBJECTS

A fractal is an object or structure that is self-similar at arbitrarily small scales. Trees, for example, reveal a fractal pattern in their structure. We can construct bigger and bigger trees adding sub-trees to its leaf nodes, as illustrated below with a family tree:

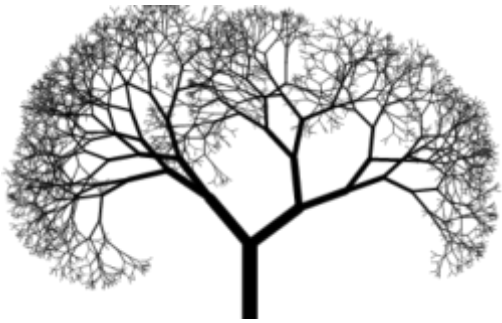
(13) *Family Trees:* [1]

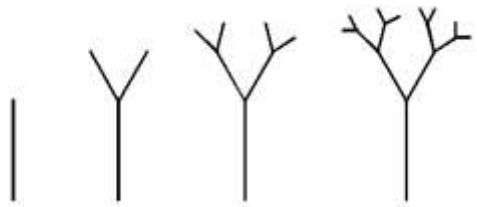


Using that strategy (but this time starting from the bottom and going upward) we can build up trees resembling real ones:

[1] Cf. researchgate.net; edrawmax.com.

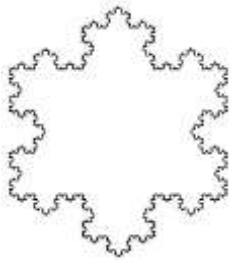
(14) *Real and Real-like Trees:*[2]

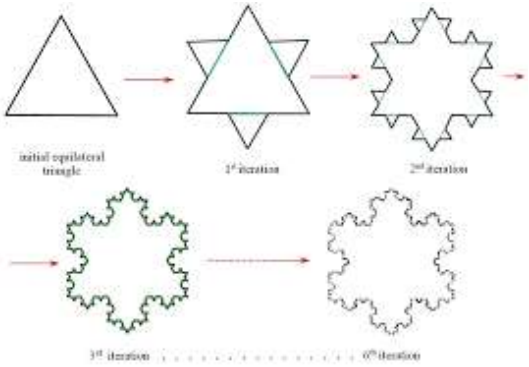




One of the earliest fractals is the Koch snowflake. It is so called because it is based on Helge von Koch's work (Helge 1904). Like other fractals, it can be built up iteratively, starting with an equilateral triangle and adding outward bends to each side of the preceding stage, with each stage yielding a shape resembling more and more a real snowflake:

(15) 2D Koch Snowflakes and Real Snowflakes:[3]

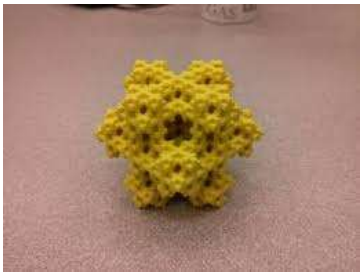




[2] Cf. blogs.timesofisrael.com/fractal-tu-bshevat; fractalsaco.weebly.com; pixels.com.
[3] Cf. researchgate.net; mypages.iit.edu; fractalsaco.weebly.com.

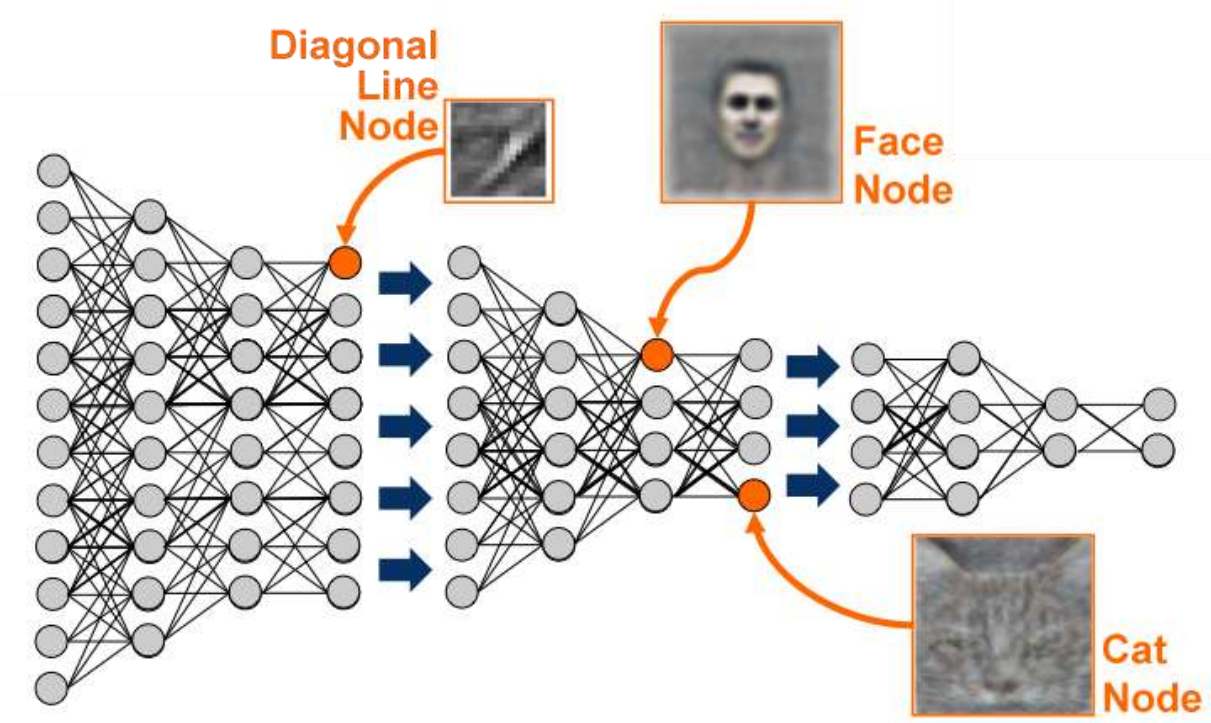
It is also possible to construct 3D Koch snowflakes, which will resemble Romanesque broccollis:

(16) 3D Koch Snowflakes and *Romanesque Brocollis*:[4]


The recurrence of a pattern at difference scales can also be observed in the structure of so-called deep neural networks. Below is an example illustrating this:

(17) Deep Learning via Multiple Layers of Neural Networks



Cf. https://krisolis.ie/deep-learning/

It is by virtue of the fact that parts of the whole network display a neural-network behavior that they are so successful at learning.

To sum up, the structural pattern of the whole of a fractal object pervades its parts but at, of course, smaller scales. There is no limit to that sort of pattern repetition. In theory, a fractal object structurally accommodates an 'infinite' number of its smaller copies.

[4] Cf. cati.com; thingiverse.com; enwikipedia.org.

5 RECURSIVE CODING AS A WAY OF CAPTURING 'INFINITY'

In fact, a similar type of pervasion that is observed in fractal objects also underlies the rules given in (12) for partially defining the ancestor relation. Below is a template for these rules:

(18) *A Recursive Template:*[5]

'$great^{N}grandparent(X, Y)$:- $parent(X, Z)$, $great^{(N-1)}grandparent(Z, Y)$.'

The body of a rule contains the goals that should be thought of as the parts of the main goal indicated by the head. Notice that the second subgoal above (i.e., $great^{(N-1)}grandparent(Z, Y)$) is a repetition of the main goal (i.e., $great^{(N-1)}grandparent(Z, Y)$) at a smaller scale.

Recall that one of the problems with the rules in (12) is that they cannot cover the ancestors older than our great-great-great-grandparents. Of course, we can increase that coverage by expanding the list of rules in accordance with the template above. However, in many cases such expansion might be limitless at worst and cumbersome and tedious at best. Notice also that for any $N \geq 0$ '$great^{N}grandparent$' will express an *ancestor/2* relationship with a certain depth. In line with this view, the rules in (12) or any expansion of it in agreement with (17) can be replaced with the following single rule:

(19) *A Recursive Rule:*

```
ancestor(X, Y):- parent(X, Z), ancestor(Z, Y).
```

Let us now complement this rule with the following:

(20) *A Base Rule:*

```
ancestor(X, Y):- parent(X, Y).
```

This rule simply says that if X is Y's parent then X is Y's ancestor and, in this way, allows us to capture the fact that our parents are among our ancestors, which is left unexpressed by the rules in (12).

Now, the two rules in (19) and (20) do not only constitute a more concise fragment of code but they also provide a complete generalization of the rules intended to encode ancestorship. Consequently, we are able to keep the track of one's ancestors with an arbitrary depth.

The *ancestor/2* procedure above is a typical example of a recursive Prolog program. A recursive program necessarily includes a **base** component and a **recursive** component. (20) constitutes the base of the *ancestor/2* procedure whereas the clause in (19) is the recursive component of that procedure. The recursive part of a program can be regarded as capable of

[5] (17) is not a legitimate Prolog code fragment. It should be taken as part of the metalanguage that we use to talk about Prolog.

encoding an unlimited set of possibilities that might hold in theory. The base component indicates where these possibilities come to an end in practice.

6 CONCLUSION

We can write a code fragment recursively if we want to capture a theoretically unlimited number of operations with it.

7 EXERCISES

A. What is the maximum number of arguments that a predicate can take?

B. Translate the rules in (5) and (6) into FOL.

C. Translate the query in (8) into FOL.

D. Translate the recursive rule in (18) into FOL.

Reference Texts

Sterling, L. and Shapiro E. (1986) The Art of prolog. Cambridge, Massachusetts: MIT Press.

Von Koch, Helge (1904). "Sur une courbe continue sans tangente, obtenue par une construction géométrique élémentaire". *Arkiv för matematik, astronomi och fysik* (in French). **1**: 681–704.