Declarative Programming: Handout 9

Yılmaz Kılıçaslan
December 14, 2022

Key Concepts: Control, Abstract Program, Resolvent, Goal Reduction, Variable Clashing, Proof Tree, Depth-First Search, Backtracking.

INTERPRETER

1 INTRODUCTION

Let us remember once again that the full computation model of a logic programming language has two components, a unification mechanism that handles data manipulation and a control mechanism that rests on a backtracking strategy:

(1) **COMPUTATION MODEL OF A LOGIC PROGRAMMING LANGUAGE**

$$\text{Unification} + \begin{matrix}\text{Computation}\\ \text{with Backtracking}\end{matrix} = \begin{matrix}\text{Logic}\\ \text{Programming Language}\end{matrix}$$

Each Prolog interpreter has to implement these two mechanisms in order to be capable of interpreting a given query with respect to a given program:

(2) **PROLOG INTERPRETER:**

Last week we scrutinized the unification mechanism that Prolog uses. Today we will try to understand the control strategies employed in logic programming by examining the algorithm of a logic-programming interpreter.

However, before moving on you should be reminded that you will need to recall and keep in mind the definitions below in order to better understand what follows:

(3) **TERMINOLOGICAL CLARIFICATION**

INSTANCE:

A is an *instance* of *B* if there is a substitution θ such that $A = B\theta$.

COMMON INSTANCE:

C is a *common instance* of *A* and *B* if it is an instance of *A* and an instance of *B*.

UNIFIER:

A *unifier* of two terms is a substitution making the terms identical.

MOST GENERAL UNIFIER (MGU):

A *most general unifier* or *mgu* of two terms is a unifier such that the associated common instance is most general. (Two terms have a unique mgu, up to alhabetic variants.)

2 AN ABSTRACT INTERPRETER

Below is an abstract interpreter algorithm (written by Sterling and Shapiro (1992)):

(4) **AN ABSTRACT INTERPRETER FOR LOGIC PROGRAMS**

Input: A goal G and a program P

Output: An instance of G that is a logical consequence of P, or *no* otherwise

Algorithm:
Initialize the resolvent to G.
while the resolvent is not empty *do*
  choose a goal A from the resolvent
  choose a (renamed) clause $A' \leftarrow B_1,\ldots,B_n$ from P
    such that A and A' unify with mgu θ
    (if no such goal and clause exist, exit the *while* loop)
  replace A by $B_1,\ldots,B_n$ in the resolvent
  apply θ to the resolvent and to G
If the resolvent is empty, *then* output G, *else* output *no*.

As can be seen from the algorithm, a computation of a logic program starts from some initial query G, to which the so-called 'resolvent' is initialized. The aim of the computation is to prove the initially given query, G, with respect to the given program, P. If this happens the computation is taken to terminate as success, in which case the output is specified as the instance of G proved. However, the computation may also terminate as failure or may fail to terminate.

Speaking more technically, the aim of a computation of a logic program is to come up with an empty resolvent, which amounts to success. The resolvent is emptied step by step (in a *while* loop) via 'goal reduction'. At each step, a goal in the resolvent and a clause in the logic program are chosen such that the goal and the head of the clause are expected to unify.

If at a certain step of goal reduction no clause can be found the head of which unifies with the chosen goal, the reduction process is stuck. The algorithm tells us that in such a case the loop of goal reduction will be exited and, hence, we will be left with a non-empty resolvent. According to the last clause of the algorithm, this amounts to failure.

If a chosen goal and the head of a chosen clause unify:

- the variables appearing in the clause are renamed so that they do not clash with the variable names used previously in the computation;
- the goal is replaced by the body of the clause; and
- the most general unifier of the goal and the head of the clause is applied to the resolvent and to G;

which results in a new resolvent. If the new resolvent is not empty, a new cycle of the loop starts. When the resolvent has been emptied, the loop is exited but this time indicating a successful termination.

We have seen above two of the possible cases for a computation of the algorithm: a failure termination and a success termination. There is a third possibility for a computation of the given algorithm. It is left to you as an exercise to find out what that third possibility can be.

You should certainly have noted that this algorithm is an abstract one. It is left unspecified in some respects. First, the algorithm does not tell which goal to choose from the resolvent to prove. Second, it remains silent with respect to the question of how to search the program to find a clause the head of which unifies with the chosen goal. Third, it does not say anything about how to traverse the (proof) tree that shows each of the possible paths that will come into existence as a result of possible goal reduction steps. Fourth, it does not provide an answer to the question of what to do when a path of reduction is stuck. In order to turn the given abstract algorithm to a concrete one, each of these points have to be specified.

The Prolog interpreter we use always chooses the **left-most** goal from the resolvent. It searches the program to find a clause with a unifying head in a **top-down** order. It traverses the proof tree with a **depth-first** strategy. When it is stuck somewhere in its search for a solution, it **backtracks** to the last point of choice where there are still alternatives available.

3 AN EXAMPLE

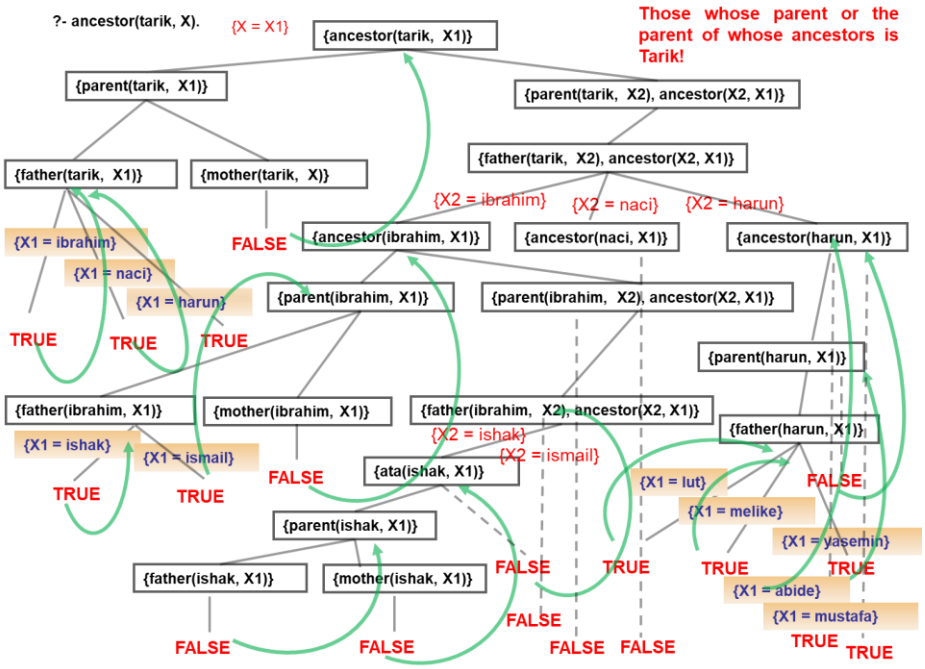
Let us illustrate what is said above with an example. Consider the following program modelling some family relations:

(5) A PROLOG PROGRAM

```
father(tarik,ibrahim).
father(tarik,naci).
father(tarik,harun).
father(ibrahim,ishak).
father(ibrahim,ismail).
father(harun,lut).
father(harun,melike).
father(harun,yasemin).
mother(sahra,ishak).
mother(melike,abide).
mother(abide,mustafa).
male(tarik).
male(ibrahim).
male(naci).
male(harun).
male(lut).
male(ishak).
male(ismail).
female(sahra).
female(melike).
female(yasemin).
female(zeliha).
female(nalan).
parent(X,Y):- father(X,Y).
parent(X,Y):- mother(X,Y).
ancestor(X,Y):- parent(X,Y).
ancestor(X,Y):- parent(X,Z),ancestor(Z,Y).
```

Below is the proof tree for the query "?- grandfather " issued for the program above:

(6) **THE PROOF TREE OF A QUERY**



4 CONCLUSION

A computation of a logic program starts from some initial query. The resolvent, which is a set that contains the goals to be resolved, is initialized to that initial query. A 'reduction' process is iteratively applied to the resolvent. At each (complete) iteration a chosen goal in the resolvent is replaced by the body of the head of a clause chosen from the program. The reduction process terminates either with an emptied resolvent, which means a 'success', or with a non-empty resolvent, which means a 'failure', or does not terminate, which means the computation has entered an infinite loop.

5 EXERCISES

A. What is the difference between a compiler and an interpreter?

B. Which one is the mgu of X = Y?

1. a(b(c))
2. a(Z)
3. Y
4. a(b(W))

C. Order the values above from the most general to the most specific.

D. In what ways can an interpreter algorithm terminate?

E. In what ways is the algorithm in Section 2 abstract?

References

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