CIS 671: Programming Assignment No.4

Sameer Shende
Matthew Giger
Junying Wang
Saikiran Sheth

November 29, 1995

1. Discussion

In this programming assignment, we are trying to create a "learning agent" that will learn some knowledge based upon the background knowledge which was defined by the family tree shown in Figure 21.5 on page 638 of the text. The background knowledge was defined in the file named "background.scm". The frustrated the thing is that we did not get a working version of the learning agent even though we tried our best to implement it.

The approaches that I tried in this program was first the one described in the book as the top-down approach. The idea was to generate a list of clauses that would be matched against the knowlege base along with the initial question to determine if the new clause was completely, partially or not consistant with the initial clause. We would take the resulting lists of positive and negative examples and choose the best one (the one with the most positive and least negative examples) and use that as the best guess for the next addition to the hypothetical clause. The problem that I ran into here was that it was a very slow method and there needed to be a way to "remember" that a particular form of a clause was already chosen, such as (parent ?x ?v1), since the very next iteration of the search loop would usually produce a (parent ?x ?v2) since that was still the best choice as far as our scoring algorithm was concerned.

At this point I tried to attack the problem from the reverse direction. I tried to use a form of inverse resolution to solve it. What this consisted of was to reduce the background info to just the predicates that were listed in the desired output format. Then search the background for all occurrances of the first name in the predicate (i.e. ?x in '(uncle ?x ?y)). Do this for each name that satisfied the "uncle" clause, and take the intersection of all of these sets. This would produce facts that were in common for all of these names such as '(male ?x) and '(parent elizabeth ?x) and '(parent philip ?x) for the "uncle" clause. Given more time (and help), I would have taken each new name (elizabeth and philip) in the common clauses and did a search for all of the common facts for them as well, always looking for the ?y name to show up in a relation. Once the ?y was seen, I would then trace the path used to reach there which should be the minimal clause to satisfy the relation.

2. Implementation

(a) Generalize Conditions.
To find the maximal set of common generalizable set of conditions in the background, we implement a way to extract all relevent "connected" relationships. By this we mean that for generating the hypothesis for (son ?x ?y) in terms of father and mother for example we have the attribute of each node in the family tree (male or female) and we have the binary predicates (or relationships). To generate the generalized conditions which give us a way to traverse the tree of the relationships (using only the relevant ones - ignoring the extraneous ones like cousin, aunt etc. from the knowledge base), we try to reduce our knowledge base. This is done by first matching all son relationships from the knowledge base and then recursively applying the list of predicates with the results of the first match. By this we mean that if our list of binary predicates is (father mother ...) then we take the results of (son ?x ?y) as (son a b) (son b c) and find out the results of four queries viz., (father ?x a) (father a ?x) (father ?x b) and (father b ?x) for each entry of the son predicates and merge this to form our set (no duplicates allowed) of the generalized conditions which represent the reduced knowledge base in terms of only the relevant connected relationships. By connection, we mean that if a node in the family was connected to some other branch of the tree by a relationship not in the background, then the other disconnected node does not figure in our set of conditions as its irrelevant. This can be extended to disconnected relationships that may exist outside the family tree.
Thus, this generalized conditions gives us the set of relations in terms of the predicates given by the user and constructs the relevant knowledge base set.

(b) The hypo function.
One experimental algorithm for hypo would be defined as follows:

```
---------------------------------------------------------------------
function hypo(relationship, background, predicates)
         returns a set of Horn clauses
  inputs: relationship, the given relationship, e.g. (uncle ?x ?y)
          background,   the given background knowledge
          predicates,   the related predicates, e.g. parent, male, female
begin
  binary-subset = get all the binary relationship from the background
  unary-subset  = get all the unary relationship from the background
  relation-set  = get all the relation pairs from the background
  allpath = get all path which was related with relationship
```

```
  maximumset = get all the relationship set
  generalizableset = substitute the variable to the maximumset eliminate
                     the duplicates
  return generalizableset
end
---------------------------------------------------------------------
```

The **hypo** can propose a most specific list of (conjunction of) conditions for a given *relationship*, based upon the content of a given list of *background* descriptions in terms of the given list of *predicates*. The algorithm should do this by finding a maximal set of common, generalizable set of conditions in the background that is common to all individual terms seen to share the given relationship.

(c) The generalize condition function
The generalize condition function will substitute the variables to the maximumset which eliminate the duplicates. If the input is as follows:

```
((grandfather philip william)
  (parent charles william)
  (parent philip charles)
  (male charles)
  (male william)
  (male philip))
```

The output will be as follows:

```
(parent ?v1 ?y) (parent ?x ?v1) (male ?v1) (male ?y) (male ?x))
```

(d) Kinship relationships.
The background is a complete set of kinship relationships for a particular set of people that is sufficient to illustrate the following relationships between people: parent, son, daughter, sister, brother, uncle, aunt, cousin, grandparent, and grandchild; information regarding the unary predicates male and female is included.

(e) Extra Credit Part 1.
Theorically, it is possible to combine a set of rule conditions to simplify the background, using one rule conclusion as a rule condition for another. For example, if we have the background:

$$father(a,b) \wedge father(b,c) \wedge father(c,d)$$

then by combine a set of rule conditions:

$$father(a,b) \wedge father(b,c) \Rightarrow grandfather(a,c)$$

we can simplify the background to as follows:

$$grandfather(a,c) \wedge father(c,d)$$

(f) Extra Credit Part 2 - Incomplete Knowledge.
The effect of incomplete information in top-down learning is that if a missing clause (knowledge) /relationship is from positive examples, then it may imply that fewer than actually required literals may be used to satisfy the relationship. If the missing clause is the critical one, then all the positive examples may be exhausted and our search for the hypothesis concluded if it is not found. Thus, the hypothesis may be shorter than the one desired. But, if its redundant (the effect is covered or explained by a similar other relationship, then the hypothesis is unaltered).
If the missing relation is from the negative examples, then a literal which is used in forming the hypothesis may be treated as correct as it does not fail in any of the negative examples but might have failed if the missing clause were introduced in the set of negative examples. As a result, some literal could be passed off as correct and this literal could explain a set of positive examples and hence it would be a question of generalisation. For example, if parent were a literal that was tested in the knowledge base and if grandfather was the target hypothesis, then it could happen that a missing grandmother negative example would classify this relation as correct and a grandfather relationship could be mistakenly be considered as a grandparent relationship in the absence of the appropriate negative example missing from the knowledge base. Thus, we would end up generalizing the relationship instead of specialising it to the grandfather.

3. Software

The software for the project is located in /home/grads/sameer/cis671/p4 directory. The files are as follows :

- NewHypo.sch - contains the inverse resolution implementation.
- Restrict.sch - contains the inverse proof for a hypothesis. Main file.
- choose-literal.sch - for top down learning. Utilities.
- examples.sch - creates positive examples for top down learning.
- new-literals.sch - creates new literals for top down learning.
- satisfy4.sch - satisfy routine without new facts.
- monitor.sch - the monitor routine which add facts or process queries. This contains the query routine used to generalize conditions.
- util.sch - contains the gen-cond routine to generalise conditions.
- satisfy_test.sch - the satisfy routine which takes propositions and facts.
- match.sch - the match routine which matches a pattern with an expression.
- rule.sch - the rule routines to extract the name, conditions and conclusions from a rule list.
- update.sch - the update routine that determines the complete set of facts given new facts.

- env.sch - the env-subst routine that creates facts given the bindings and the environment.
- kin.sch - the kinship rules that determine the relationships.
- Family.sch - the data for hypo for the family relationships.
- linear.sch - the fact list layer for linear storage of facts.
- load.sch - the script that loads the files for the gen-cond routine.
- kb.sch - the knowledge base containing all the relationships. The script files for the monitor program on kinship relationships and other test cases are stored as follows :

4. Conclusions

The appropriate placement of negative examples can speed up the process of eliminating a horn clause. For example if we are matching for the literal Grandson and we have at the beginning of negative examples the relation (not (grandson (George Anne))) then we can eliminate the child as a potential candidate for (Elizabeth Charles) (Elizabeth Anne) relationships and have the Grandson be specialised in terms of its attribute (male). This speeds up the process of throwing out a literal which does explain some negative example when it shouldn't. However in implementing this we take the entire knowledge base and negate the non positive examples - so its possible to apply this heuristic to the search only in the case when the user gives the negative examples.

One problem with both of the approaches we tried was a lack of information. In the uncle example, the uncle Mark didn't have any parental relations in the background info. There also was no spousal relation specified, but that wouldn't matter if the predicate list was restricted to '(parent male female). This is a problem in general since the Mark case would give a false negative to each of these types of algorithms and would make satisfying all of the positive examples an impossibility. The solution to this is to either take out that "uncle" relation or to somehow make the acceptance routines for the clauses less restrictive in that they would allow no negative examples yet still allow not all of the positive examples to be satisfied.

5. Appendix

The appendix contains the source codes and the script files for the assignment.