

A GUI FOR DEFINING INDUCTIVE LOGIC PROGRAMMING TASKS FOR  
NOVICE USERS

A THESIS  
SUBMITTED TO THE FACULTY OF  
UNIVERSITY OF MINNESOTA  
BY

PRIYANKANA BASAK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
FOR THE DEGREE OF  
MASTER OF SCIENCE

Advised By

DR. RICHARD MACLIN

March 2017



## Acknowledgements

I would like to thank my advisor, Dr. Richard Maclin for sharing his immense knowledge and constant motivation. He was always very patient in answering questions and guiding me in taking the next steps towards the research and writing of the thesis.

I would also like to thank my defense committee members Dr. Haiyang Wang and Dr. Marshall Hampton for their support, and Dr. Peter Willemsen, Director of Graduate Studies for his guidance. I would like to thank the computer department science faculty members, Dr. Douglas Dunham, Dr. Neda Saeedloei, Dr. Ted Pedersen, and Dr. Hudson Turner who taught me the important concepts of computer science.

I would like to thank Lori Lucia and Clare Ford for providing all the help and support whenever I needed it.

I would like to thank my family and friends for their constant support and encouragement.

## **Dedication**

I dedicate this thesis to my parents, Soumen Basak and Madhumita Basak, and my sister, Apaala Basak whose constant support and encouragement helped me to stay focused and fulfill my aspirations.

## ABSTRACT

Inductive logic programming, which involves learning a solution to a problem where data is more naturally viewed as multiple tables with relationships between the tables, is an extremely powerful learning method [2]. But these methods have suffered from the fact that very few are written in languages other than Prolog and because describing such problems is difficult. To describe an inductive logic programming problem the user needs to designate many tables and relationships and often provide some knowledge about the relationships in order for the techniques to work well.

The goal of this thesis is to develop a Java-based Graphical User Interface (GUI) for novice users that will allow them to define ILP problems by connecting to an existing database and allowing users to define such a problem in an understandable way, perhaps with the assistance of data exploration techniques from the GUI.

**Table of Contents**

1. Introduction	1
2. Background	3
2.1 Induction and Deduction	3
2.1.1 Induction	3
2.1.2 Deduction	4
2.2 Inductive Logic Programming	5
Fig 2.1 Relationship between background knowledge and the positive and negative examples	7
2.3 ILP Techniques	9
Fig 2.2 A sample dataset to represent the family relation	10
2.3.1 Generalization	11
2.3.2 Specialization	13
2.3.2 Inverse Resolution	14
Fig 2.3 Example of a top-down approach of specialization technique	14
Fig 2.4 Single resolution	15
2.4 ILP Research Areas	17
2.4.1 Incorporating Probabilistic Inference	17
2.4.2 Stochastic Search	18
2.4.3 Special Purpose Reasoners	18
2.4.4 Collaborating With Human Experts	19
2.4.5 Parallel Execution	20

2.5 ILP Systems	21
2.5.1 ALEPH	21
2.5.2 WEKA	22
2.5.2.1 Data Preprocessing	22
2.5.2.2 Attribute Selection	24
2.5.2.3 Association Rules	27
2.5.2.4 Clustering Algorithms	28
2.5.3 FOIL	31
2.5.4 Sequential Covering Algorithm	33
2.6 Graphical User Interface	34
3. Implementation	36
3.1 Model Transformation	36
Fig 3.1 Process overview of Model Transformation by Example	37
Fig 3.2 Source and target metamodels	39
Fig 3.3 Mapping Metamodel	40
Fig 3.4 Clauses in the model transformation	41
Fig 3.5 Background Knowledge with the different facts	42
Fig 3.6 Results after counter analyzing	43
Fig 3.7 Analyzing Connectivity	44
3.2 Java	45

4. Results	46
4.1 The GUI	46
Fig 4.1 Select a table from a database to select data	46
4.2 Selecting Data	47
Fig 4.2 The data from the table gets transferred into a text file	47
4.3 Examples of Results	48
Fig 4.3 The output in the form of rules learnt from the input data	48
5. Conclusions and Future Work	49
6. References	50
7. Appendix	53
7.1 Application Menu	53
7.2 Input from the Database	56
7.3 Conversion of data into a text file	59
7.4 Read from the text file	63



# 1. INTRODUCTION

The volume of data created by the internet and business enterprises is increasing by many-fold each and every year. Most of the data is unstructured [1]. Data mining is the process in which we find patterns, anomalies and relationships between large data sets to predict the outcomes.

The three major foundations of data mining [1] are:

1. The numerical study of relationships in data,
2. The human-like intelligence provided by software/machines,
3. Algorithms that learn from data in order to make predictions.

Data mining allows us to filter through the noise in data and decide what is relevant. This accelerates the speed in which decisions are made. Generally, some of the relational upgrades of data mining and concept learning systems employ first-order predicate logic as the representation language associated with background knowledge and data structures [2]. The learning systems that incorporate valid logical patterns for a particular background knowledge are classified under the research area called Inductive Logic Programming (ILP).

ILP can be defined as the intersection of inductive learning and logic programming [2].

Inductive learning involves the process in which learning is performed on the basis of

examples whereas logic programming is a pattern of programming, which is based on a formal logic. These two forms of programming combine to form the basis of Inductive Logic Programming. ILP involves learning a solution to a problem where data is more naturally viewed as multiple tables with relationships between the tables is an extremely powerful learning method.

The goal of this thesis is to develop a Java-based Graphical User Interface (GUI) for the novice users that will allow them to define ILP problems by connecting to an existing database and allowing users to define such a problem in an understandable way, perhaps with the assistance of data exploration techniques from the GUI.

The databases that we used to develop Graphical User Interface (GUI) in this thesis are UCI Diabetes Database and the Breast Cancer Database. The aim is to generate rules that would determine the blood insulin level in the person at a future point of time by analyzing the past data. The positive and negative data are separated and processed.

The front end of the GUI is based mostly on the Java Swing Framework. The GUI allows the user to select a particular database and select the tables and columns from them. Then the data from those columns get transferred to a text file, which is sent to the code that works on the data by following the Sequential Covering Algorithm to create the rules and display them to the user. The connectivity of the text file to the ILP code has to be integrated in order to transfer all the data from tables successfully to it.

## **2. BACKGROUND**

This background chapter discusses the concepts of Inductive Logic Programming in detail. It starts by differentiating Induction and Deduction and then progresses into the concept of ILP with different examples. It then discusses different techniques used in ILP, followed by ILP research areas. It further discusses various GUI methods and their implementations. Finally, some existing ILP and data mining systems are discussed briefly.

### **2.1 Induction and Deduction**

ILP mostly uses induction instead of deduction to provide solution to the problems. Given below are the processes in details.

#### **2.1.1 Induction**

The process in which the given logic is generalized to reach conclusions with the process of reasoning, which may or may not be supported by or are biased towards false examples, is known as induction.

Induction often follows a bottom-up approach that is it goes from specific to general.

For example, let us consider we have the following logic:

This bag is green.

That bag is green.

Third bag is green.

The induction can lead to the following conclusion:

All bags are green

### **2.1.2 Deduction**

The process, which starts with logic statements and then reasons to find conclusions assuming the logic to be true, is known as deduction.

Deduction follows a top-down approach that is it goes from general to specific.

For example, let us consider we have the following logic:

Copper conducts electricity.

This wire is made of copper.

The deduction can lead to the following conclusion:

This wire will conduct electricity.

## 2.2 Inductive Logic Programming

Inductive Logic Programming (ILP) [1] is a merging point of inductive learning and logic programming. It involves the uniform representation of background knowledge, hypotheses and examples. The background knowledge is presented in the form of facts, which can be represented in a logical database and an ILP system, such as Aleph (discussed later) can derive a rule based on all the positive examples and taking into account none of the negative examples [20].

For example, given the following set of facts:

```
child(john, ann) .
```

```
child(john, alex) .
```

```
child(ann, bob) .
```

It should be noticed that a fact has no negative literals and always expresses the positive knowledge that is definite.

As can be observed from above, the knowledge that John is a child of Ann and Alex, and that Ann is a child of Bob, can be expressed by two facts which are given above.

If we add a tag for the example:

```
grand_child(john, bob) .
```

We might learn the rule:

```
grand_child(X,Y) :- child(X,Z), child(Z,Y).
```

though this is unlikely from just one example. A rule has one or more negative literals, which can or cannot be the body of the clause. It is used to learn certain predicates from the other predicates.

As can be observed, the knowledge that a grandchild is the child of a child, can be used to learn the above given rule.

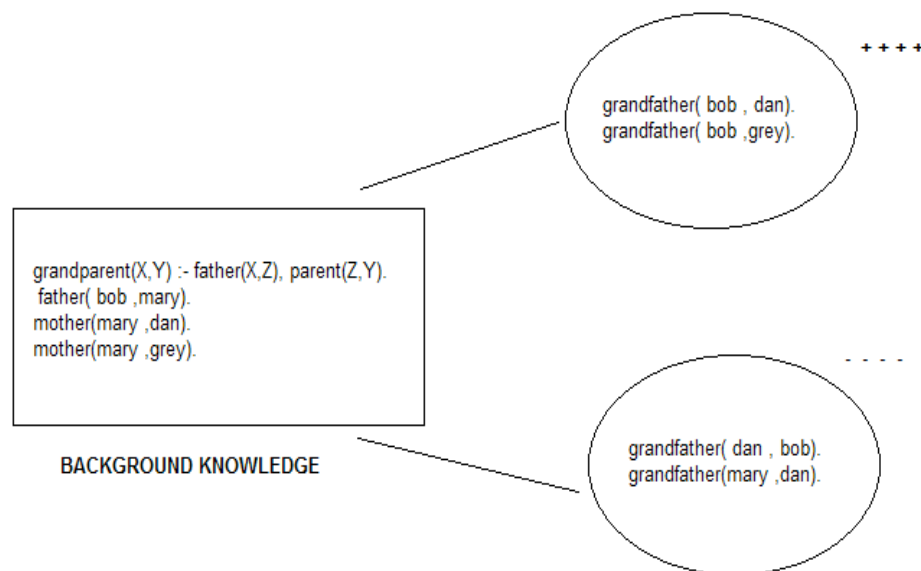
ILP is attractive generally reasons because it is able to reason about not only predicates, but connections between those predicates in a database. ILP is mostly well-suited to perform a variety of tasks because of its abilities to take into account the background knowledge and structured data that is used to produce human-comprehensible results [4].

The standard ILP framework talks about learning from entailment [10]. Here, the learner is provided with positive and negative examples, denoted by  $E^+$  and  $E^-$  respectively and a background knowledge  $K$ , which is a set of definite clauses. The goal is to induce a hypothesis  $H$ , which is also a set of definite clauses, such that  $K \cup H$  covers all of the positive examples and none of the negative ones (though, in real-world situations there is an expectation that a rule might cover one or more negative examples due to the presence of noise).

Induction in ILP is interpreted as abduction in combination with justification [10]. Abduction can be described as the process of hypothesis formation from facts. On the other hand, justification is the degree of belief in a certain hypothesis when there is a certain amount of evidence present.

As a working example, we are considering some traditional family relationships. Figure 2.1 describes this example in detail.

The background knowledge  $\mathcal{K}$  contains the following clauses. Some positive examples  $E^+$  can be given and also some negative facts  $E^-$  can be listed.



**Figure 2.1 Relationship between background knowledge and the positive and negative examples**

By default, a hypothesis is set up only for clauses used in the positive and negative examples. But, there is some (limited) support for abductive learning [1] when Aleph, described later, which is considered to be a powerful inductive logic programming engine, will be able to derive hypothesis not only for the clause covered by positive and negative examples. Such as

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

There are basic four characteristics of Inductive Logic Programming. They are:

1. Incremental/ Non-Incremental: The examples are given as input before the learning process starts. In Incremental ILP, examples are given as input one by one during the learning process.
2. Interactive/Non-Interactive: In interactive system, the learner asks the user questions, that help to determine if the intended ILP system can be achieved or not while implementing an incremental model. In non-interactive systems the learner does not ask any questions.
3. Single/Multiple Predicate Learning: The systems can learn from single predicate or from multiple predicates.
4. Theory Revision: The system which accepts an existing hypothesis as input for learning is called Theory Revision. It is an incremental multiple predicate learner.



## 2.3 ILP Techniques

ILP techniques are the different processes that are used to generate hypotheses from given background knowledge, positive examples and negative examples. This section talks about the basic ILP techniques such as Generalization, Specialization and Inverse Resolution which are used to generate hypotheses.

Consider the following defined predicates:

`father(X,Y) - X is the father of Y`

`parent(X,Y) - X is a parent of Y`

`male(X) - X is a male`

Our target here is to find the relation “father” using the relations parent and male.

The background knowledge, positive examples and negative examples can be considered as below:

Background Knowledge (B):

`parent(John, Steve) .`

`parent(John, Ann) .`

`parent(Steve, Bob) .`

`parent(Ann, Dan) .`

```
parent(Julie, Steve).
```

```
male(John).
```

```
male(Steve).
```

```
male(Dan).
```

Positive Examples (E+):

```
father(John, Steve).
```

```
father(John, Ann).
```

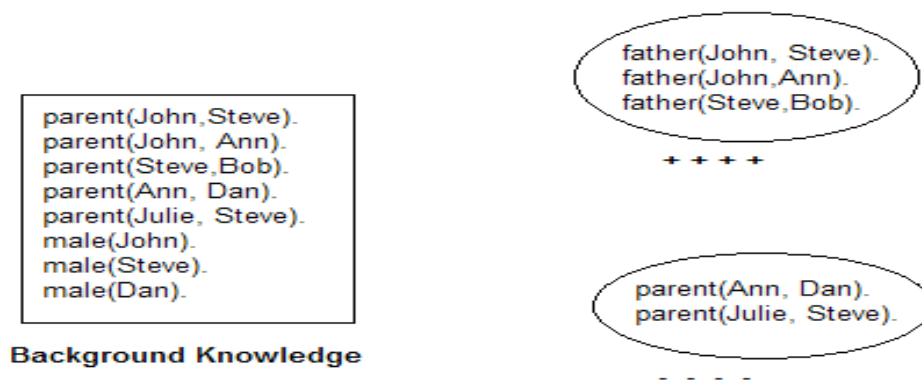
```
father(Steve, Bob).
```

Negative Examples (E-):

```
parent(Ann, Dan).
```

```
parent(Julie, Steve).
```

The same predicate dataset can be represented as shown in Figure 2.2:



**Figure 2.2:** A sample dataset to represent the family relation

### 2.3.1 Generalization

The process of finding hypotheses using a bottom-up approach can be defined as generalization. It starts from the most specific clause that covers a positive example and generalized until and unless it ceases to cover any negative example.

Least general generalization (lgg) [11] is a process that supposes that if a set of clauses are true, then the generalization of them is also true. Let us consider that we have two clauses,  $c_1$  and  $c_2$ , which are both true. So, the least general generalization of those two clauses, denoted by  $lgg(c_1, c_2)$  may also be true.

For example, from our existing background knowledge, we can define the clauses as:

```
c1 = father(John,Ann) :- male(John) , parent(John, Ann)
```

```
c2 = father(Steve,Bob) :- male(Steve) , parent(Steve,Bob)
```

These clauses can be written as:

```
c1 = {father(John,Ann) , male(John) , parent(John, Ann)}
```

```
c2 = {father(Steve,Bob) , male(Steve) , parent(Steve,Bob)}
```

The  $\text{lgg}(c1, c2)$  can be represented as under

$$\{\text{lgg}(\text{father}(\text{John}, \text{Ann}), \text{father}(\text{Steve}, \text{Bob})), \text{lgg}(\text{male}(\text{John}), \text{male}(\text{Steve})), \text{lgg}(\text{parent}(\text{John}, \text{Ann}), \text{parent}(\text{Steve}, \text{Bob}))\}$$

After computation of the lgg of the literals, the given  $\text{lgg}(c1, c2)$  can be written as:

$$\text{lgg}(c1, c2) = \{\text{father}(\text{lgg}(\text{Steve}, \text{Bob}), \text{lgg}(\text{John}, \text{Ann})), \text{male}(\text{lgg}(\text{Steve}, \text{Bob})), \text{parent}(\text{lgg}(\text{Steve}, \text{Bob}), \text{lgg}(\text{John}, \text{Ann}))\}$$

Now, substituting  $\text{lgg}(\text{Steve}, \text{Bob})$  as X and  $\text{lgg}(\text{John}, \text{Ann})$  as Y, the

$\text{lgg}(c1, c2)$  can be rewritten as under:

$$\text{lgg}(c1, c2) = \{\text{father}(X, Y), \text{male}(X), \text{parent}(X, Y)\}$$

Ultimately, the final clause can be represented as under:

$$\text{father}(X, Y) :- \text{male}(X), \text{parent}(X, Y)$$

### 2.3.2 Specialization

The process of finding hypotheses using a top-down approach can be defined as specialization. It starts from the most general clause and adds literals to that clause until and unless it ceases to cover any negative example.

The specialization techniques mostly use refinement graphs to search the hypothesis space in a top-down approach. A refinement graph is a directed and acyclic graph. During the generation of a hypothesis, a refinement graph is constructed with the nodes as clauses and the edges as refinement operators.

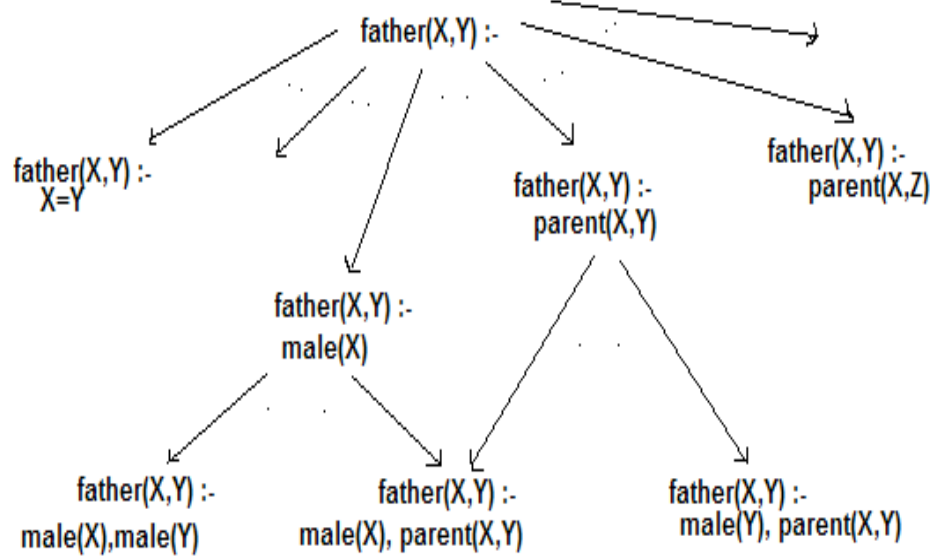
For example, from our given background knowledge, positive examples and negative examples, in order to derive the hypothesis for the father relation, the first level of refinement adds clauses to the general clause, such as:

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

The second level of refinement adds another clause that includes all the positive examples and eliminates all the negative examples to finally produce the following:

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

Given below is the refinement graph for the above example:



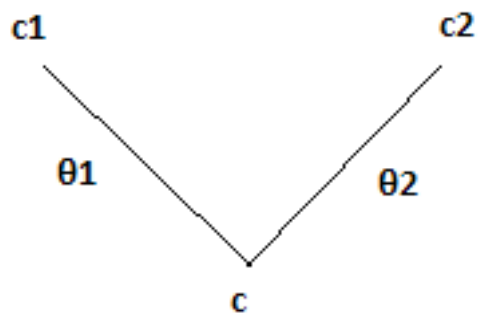
**Figure 2.3: Example of a top-down approach of specialization technique**

### 2.3.2 Inverse Resolution

Inverse Resolution [13] is another generalization technique, which can be thought of as inverting the resolution rule of the deductive inference. The resolution in this case works in a backward manner. Resolution is a clear and complete method that is used in deductive systems. Inverse resolution involves the learning of rules from examples and background knowledge by working backwards in resolution trees.

A resolution uses substitutions. Let us consider two clauses,  $c_1$  and  $c_2$ , which are used to derive the main clause  $c$ . Here, the clauses  $c_1$  and  $c_2$  are called the arm clauses whereas the clause  $c$  is called the base clause. In order to perform this action, we require two substitutions  $\theta_1$  and  $\theta_2$  for the arm clauses. Then the clause derived, that is  $c$  is called the resolvent of  $c_1$  and  $c_2$  and can be denoted as  $\text{res}(c_1, c_2)$ .

There are three inverse resolution operators: Identification operator, intra-construction operator and the inter-construction operator. The identification operator is also called the  $\vee$  operator whereas the intra-construction operator and the inter-construction operator are called the  $\text{W}$  operators, which can be formed by joining two  $\vee$  operators.



**Figure 2.4: Single resolution**

Now let us consider our family relation example in order to derive the hypothesis for:

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

The background knowledge consists of two clauses and a fact. They are:

```
b1 = male(John)
```

```
b2 = parent(John,Ann)
```

```
father(John,Ann)
```

The hypothesis can be derived in two steps using the absorption operator that can be described as under:

1. Initial hypothesis -  $\text{father}(\text{John}, Y) :- \text{parent}(\text{John}, Y)$  is derived from the background clause b2 and the given fact. Here the inverse substitution  $\theta_{1-1} = \{\text{Ann}/Y\}$  is applied.
2. Final hypothesis -  $\text{father}(X, Y) :- \text{male}(X), \text{parent}(X, Y)$  is derived from the initial hypothesis and background clause b1. Here the inverse substitution  $\theta_{2-1} = \{\text{John}/X\}$  is applied.



## 2.4 ILP Research Areas

There have been many areas of research relating to Inductive Logic Programming. For several years, systems have been developed as an addition to the existing systems to form hypotheses and rules based on predicates. The predicates are expressed as facts and then Prolog logic is applied to it. This section describes five different directions for ILP research [4].

### 2.4.1. Incorporating Probabilistic Inference

Probabilities help to model the task's inherent uncertainty by designing Bayes Nets that reason about the probability distributions efficiently [4]. However, Bayes Nets fail to capture certain properties and consider only the probabilities of the associated content. Hence ILP is used, which helps the researchers to examine and learn the clauses along with the attached probabilities. Clauses contain random variables in addition to normal logical variables [4]. In other words, the ILP algorithms can learn clauses with certain modifications. Each clause that is constructed using the ILP algorithm collect all the positive examples that are covered by the clause. The positive examples are collected by the ILP algorithm and are covered by a clause. Every positive example gives a value for a random variable that is present in the head of the clause. The example together with the background knowledge gives us the values for the random variables that are supposed to be in the body. All these values and all the covered positive examples are used as the data for constructing the conditional probability table (CPT) that accompanies the attached Bayes Net fragment [4].

### **2.4.2 Stochastic Search**

Most ILP algorithms seek a clause that maximizes some function of the size of the clause and coverage of the clause, that is, the numbers of positive and negative examples produced by the clause along with the background theory [4]. The ILP algorithms can be classified as either bottom-up (based on least general generalization) or top-down (based on refinement). The algorithms can also be further classified according to the type of search they perform, that is whether they perform, for example, a greedy search, beam search or an admissible search.

ILP algorithms generally face two difficult search problems [4]. The search of the lattice of clauses is the first of the two. The second is simply testing the coverage of a clause, which involves repeated searches for the proof of the clause. Most of the stochastic search algorithms have been genetic algorithms [4]. Stochastic local search algorithms for propositional satisfiability benefit from the ability to quickly test whether a truth assignment satisfies a formula [5].

### **2.4.3. Special Purpose Reasoners**

Incorporating special purpose reasoners benefit many ILP applications [4]. The construction of a constraint may be similar to inventing a predicate. Predicate invention within ILP has a long history [4]. Some general techniques for the creation of predicate faces the problem that the space of “inventable” predicates is unconstrained, and thus allows the predicate invention to be roughly equivalent to removing all bias from

inductive learning [4]. Removing bias might look as a good idea, but inductive learning requires bias [4]. The special purpose techniques for constraint construction are targeted to materialize the idea of performing predicate invention in way that is limited enough to be effective [4].

#### **2.4 4. Collaborating With Human Experts**

Machine learning system and humans acting as a team prove to be ideal in order to discover new knowledge [4]. ILP systems have three properties that help them collaborate with humans. First is the Declarative Background Knowledge, which is a collaboration can begin with a domain expert, who provides the learning system with general knowledge that might be useful in the construction of the hypotheses [4].

Second comes the natural descriptions of structured examples. ILP systems allow a structured example to be described naturally in terms of the objects that compose it, together with relations between those objects [4]. Third, is the Human-Comprehensible Output, which is presented to a user by ILP systems in addition to the propositional logic.

It is also true that the ILP Systems can be extended to display some capabilities such as: maintaining and summarizing alternate hypotheses that explain or describe the data, propose to perform experiments in order to differentiate between competing hypotheses, providing non-numerical justification for hypotheses, to answer an expert's questions regarding hypotheses and also consult the expert about the anomalies or surprises present in the data [4].

### **2.4.5. Parallel Execution**

Run-time and space requirements are the two significant issues faced by ILP. But most ILP systems already use broad constraints, such as maximum clause size, to hold down the exponential terms. [4]. The idea in parallel processing is a decrease in processing time, and to achieve super linear speed up, if possible in some cases. The barriers to achieve this include the overhead in communication between processes and the competition for resources between processes. So a good parallel scheme is one where the processes are relatively independent of one another and so they require little communication of any sort or resource sharing. The key observation in the designing of a parallel ILP scheme is that two competing hypotheses can be tested against the data. These hypotheses can be completely independent of one another. So the approach taken here distributes the hypothesis space among different processors to test against the data. These processors do not communicate with one another during the testing, and also they do not write to a shared memory space. Finally a parallel ILP scheme could be employed with a master-worker design. Here the master can assign different segments of the hypothesis space to workers and then the workers could test hypotheses against the data.

## 2.5 ILP Systems

The following section talks about some existing ILP Systems. It concentrates on systems such as Aleph, WEKA and FOIL. Every system has its own method of generating hypothesis for given facts and rules. They are discussed in detail below.

### 2.5.1 ALEPH

A Learning Engine for Proposing Hypotheses (ALEPH) [7] is an open source Inductive Logic Programming (ILP) system written in Prolog. Initially developed to work on the inverse entailment, it was later extended to include some other ILP functionality systems. It uses a bottom-up approach and is non incremental in nature.

The basic Aleph Algorithm can be described in four different steps. They are:

1. **Example Selection:** An example to be generalized is selected. If there is no such example present, then stop, otherwise go to Step 2.
2. **Building the most-specific-clause.** The most specific clause that entails the example selected and is within language restrictions provided is constructed. A “bottom clause”, which is usually a definite clause with many literals is selected. This step is called the "saturation" step.
3. **Searching.** A clause that is more general than the bottom clause is searched. The process of finding it is done by searching for a subset literal in the bottom clause with the "best" score. This step is called the "reduction" step.

4. **Removal of redundant clause.** In this step, the clause with the best score is added to the current theory, and consequently all the other examples that are redundant are removed. This step is called the "cover removal" step. In case if the best clause makes clauses other than the examples redundant, the algorithm returns to Step 1.

## **2.5.2 WEKA**

Waikato Environment for Knowledge Analysis (Weka) [9] is a collection of machine learning algorithms for performing the data mining tasks. These algorithms can be applied directly to a dataset or can be called from external Java code. Weka also contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. Weka is open source software issued under the GNU General Public License. Discussed below are some of the methodologies of Weka.

### **2.5.2.1 Data Preprocessing**

Data preprocessing plays a major role in the process of data mining. Preprocessing of the data is used to extract the required data, remove the noise from it and sometimes merge the data to create and transform the data into the required form. The collected data has to be reduced efficiently to produce a data set that can be mined. The data set can be reduced in two ways, one by removing the unnecessary attributes and the other being to remove the number of the tuples in the data set.

An attribute-oriented induction method has its own drawbacks. First it is based on concept hierarchies, which generalizes most symbols but fails at the generalization of numbers. Second, there is no rule that determines the threshold number of tuples that are to be selected; so a smaller number might overgeneralize and a larger number might under-generalize the data.

There are certain discretization algorithms that automatically discretize the numeric attributes and remove the irrelevant ones. They determine a criterion on the basis of which a numeric attribute is checked to determine if it has any connection to the concept used for learning. DBChiMerge is one of the algorithms that use the  $\chi^2$  statistics to determine if the relative class frequencies of the nearby intervals are similar so that they can be merged into a single one. The algorithm talks about a formula to calculate the threshold number. In conclusion, if the classes are dependent on the interval, then they are merged; Otherwise if they are not dependent, they are kept separate.

Another algorithm, namely DB-Horizontal Reduction algorithm talks about controlling the data reduction to a point where the distortions are avoided. Each and every attribute is examined one by one in this case. If a numeric attribute is found to be irrelevant in determining any of the classes, it is simply removed. After the process completes, after quite a bit of generalization and discretization, many different tuples become identical, whereas most repeated tuples are merged. This decreases the number of tuples horizontally.

The main goal of preprocessing data is to effectively reduce the amount of data so that no essential information is lost. The numeric attributes are divided into intervals. If discretization occurs in any interval, then that attribute is removed. The identical tuples are merged whereas the unimportant attributes are removed. The two major benefits of the algorithm are that firstly it increases the accuracy of the mining algorithm as all the spurious data is removed and secondly it reduces the running time of the mining process.

### **2.5.2.2 Attribute Selection**

Attribute selection is a combination of search and utility estimation which has connection to the various learning schemes. Data which contains irrelevant, redundant and noisy attributes cause poor performance and much more computations. The simple nearest neighbour learner is used to retrieve all the features present in the distant computations. Some other algorithms try to distinguish relevant data from the irrelevant ones. A heuristic search procedure is usually added to an attribute utility estimator to calculate the relative merit of alternative subsets of the attributes.

Attribute selection techniques can be categorized into filter and wrapper groups. Wrappers evaluate attributes according to learning algorithms whereas filters use general data characteristics to evaluate the attributes, being independent of learning algorithms. Some attribute selection techniques carry out regression methods on a class of numeric data instead of discrete numbers.



The simplest and fastest attribute selection attribute is the Information Gain Attribute Ranking. It is used mostly in text categorization applications. The amount of entropy is calculated for each class and the decrease in amount reflects the additional information that the class provides and that is known as information gain. Each attribute is given a score based on the information gain and the higher score ones are selected.

The second method is known as Relief, which is an instance-based attribute ranking scheme that randomly samples an instance from the data and locates its nearest neighbor from the same and the opposite classes. The comparative values are then used to update the relevance scores for each attribute. It has a smoothing effect on the noise contained in the data. It can also handle multi-class data sets by weighing the prior probability of each class, sampled to the nearest neighbor. For discrete values, the difference is calculated to be 0 or 1, whereas for continuous attributes, the actual difference is normalized to the interval  $[0,1]$ .

Principal Components Analysis is a statistical technique that is used to reduce the data dimensions as a by-product of the attribute space transformation. The covariance matrix of the original attributes is computed and then the eigenvectors are calculated. These vectors are ranked according to the amount of variation. The discrete attributes are converted to binary attributes and then the attributes with more variation are selected while the others are rejected.

CFS (Correlation-based Feature Selection) evaluates subsets of the attributes and assigns a high score to the subsets, which has attributes with higher correlation to the class and lower correlation to each other. The attributes, which are redundant, have higher correlation with each other and hence they are dropped. The modified forward selection search, producing a ranked list of attributes is used to get a good set of attributes.

Consistency-based Subset Evaluation uses class consistency to evaluate, which attributes may be selected and which ones may be rejected. This method does a biased search favoring the small feature subsets with a higher class consistency. The numeric class attributes are first discretized and then the forward selection method produces a ranked list of attributes having overall contribution to the consistency of the attribute set.

Wrapper attribute selection uses target learning algorithms in order to estimate the importance of the attributes to be selected. The other process that is used to provide accuracy of a classifier on novel data is cross validation on a given subset. For the maximum accuracy, five-fold cross validation is used. The algorithm is repeated until the standard deviation crosses the mean accuracy or until five validations are performed. Due to the interaction between search and learning scheme's inductive bias, this algorithm provides a much better result set.

### 2.5.2.3 Association Rules

Association rules are used to find out the elements that occur with each other frequently within a set of data consisting of multiple independent selections of certain elements as basket data or purchases and to discover the rules that they follow.

Basket data is the kind of data records the items that are frequently purchased together or bought at regular intervals in a period of time by a customer in a supermarket. There are certain algorithms that account for the association rules within a set of items (or an itemset).

The formal model is described here. For a set of items,  $I = \{I_1, I_2, \dots, I_n\}$  and a database of transactions  $T$ , each transaction  $t$  in  $T$  can be represented as a subset of  $I$ . So now let us consider  $A$  to be a subset of  $I$ . By an association rule, we mean an implication of the form  $A \Rightarrow I_k$ , where  $A$  is a set of some items in  $I$ , and  $I_k$  is a single item in  $I$  that is not present in  $A$ . Syntactic constraints invoke restrictions on the items that appear in a rule. Support constraints on the other hand talk about the number of transactions that support a rule.

The main task is to discover the large itemsets. The Apriori Algorithm makes multiple passes on the database and determines the frontier set (extended during pass) and candidate itemsets (derived from tuples in the database). A counter is assigned to each

itemset that stores the number of corresponding transactions and is initialized to zero on creation of a new itemset.

The Candidate itemsets are generated from the frontier itemset by recursively combining with the other items in the tuple. A small itemset is not extended. To avoid the replication of the same itemset, the items are ordered. An itemset A is extended only by the items that come later in ordering than its other members. The frontier set is composed of those candidate itemsets that are expected to be small but came out to be large in a pass.

In conclusion, the Apriori Algorithm uses a "bottom up" approach, to find the subsets, which have at least a minimum confidence. The algorithm ends at a point when there are no further successful extensions are found.

#### **2.5.2.4 Clustering Algorithms**

Clusters are groups of objects which are more similar to one another than to the other objects in a different cluster. Clustering or cluster analysis is process of putting each object in a cluster and then studying each cluster. The three most important techniques in data mining are regression, classification and clustering. The greater the similarity of objects in a group, the greater is the variance between two groups and that provides a much distinct clustering. Discussed below are some of the clustering algorithms that are used in WEKA, a data mining software produced by the University of Waikato.

The simplest learning algorithm to do clustering is the K-means algorithm. It partitions all the n-observations into k-clusters where each object is put into the cluster with the nearest mean. The main step is to define k-centroids, which are as different from each other as possible. This results in the most different results. Then each object is assigned a group closest to the centroid. The next step is to calculate the position of the centroids again. This loop continues until no further changes can be done. This algorithm produces tighter and faster clusters and works best for globular clusters. The only issue is that fixed number of clusters makes it difficult to predict 'k', which has to be decided initially.

The COBWEB algorithm creates a classification tree which categorizes each cluster with a probabilistic description. It uses a heuristic evaluation measure in order to guide the construction of the tree. Any new cluster can be added at any point of time and this makes it advantageous over the k-means algorithm. It allows bi-directional search for COBWEB algorithm. The main disadvantage is that the algorithm assumes probability distributions are independent of each other, which in reality is not true as correlation between attributes exist a lot. Large number of values increases space and time complexities especially if the classification tree is not height balanced for skewed input data.

DBSCAN (Density Based Spatial Clustering of Applications with Noise) is another algorithm which is used to form clusters from the estimated density distribution of their corresponding nodes. It does not require that the number of clusters of data to be known from before. It arbitrarily finds them and also takes into consideration the noise in the data. It is insensitive to the ordering of points in the database. This algorithm fails when

the datasets have large difference in densities. The distance calculated for clustering by the function region query is directly proportional to the effectiveness of the clustering. So if the distance is not calculated properly, the clustering doesn't happen effectively.

Expectation Maximization (EM) Algorithm is used to estimate the parameters in statistical models, where the parameters depend on unobserved latent variables. The Expectation step is used to compute the log-likelihood of the current estimate whereas the Maximization step is used to compute the parameters maximizing the expected log likelihood found on the Expectation step. The classes have indices, starting from 0, 0 being the cluster with the highest probability. This algorithm provides much useful results for the real world data set and is mostly used as an upgrade to the k-means algorithm. The only disadvantage is that the algorithm is very complex in nature.

Farthest first algorithm is a fast algorithm that is best suited for high scale applications. It puts each cluster center at a point which is far away from the present ones. This point has to lie within the data area. This algorithm needs much less rearrangement and adjustment and hence is the fastest.

The OPTICS (Ordering Points To Identify the Clustering Structure) algorithm is quite similar to the DBSCAN algorithm. It builds upon the latter and introduces values stored along with each data point to overcome the necessity to supply different input parameters. The reach ability distance is calculated between two given objects. These distances help in the ordering of objects in the cluster. The clusters are now defined also with reach ability distance in addition to the core distance.

### 2.5.3 FOIL

Quinlan proposed the First Order Inductive Learner (FOIL) [17]. FOIL uses a top-down approach starting from the most generalized rule and continuing to a specific rule. Initially the algorithm starts with a generalized rule and keeps adding literals until no negative examples are covered. FOIL hill climbs using a heuristic to cover all the positive examples.

The algorithm tries to find a rule that covers as many positive examples as possible while covering no negative examples. Then it adds the rules to an initially empty hypothesis followed by removing the positive examples that are covered by the newly formed rule. After all positives are covered, the rules are then put to review and any redundancy is removed, followed by a re-ordering.

The basic algorithm can be summarized as follows [16] [17]:

1. Initialization of the rule with target relation (head of rule) and training set,  $T$ , to positive examples, not covered by any of the rules before and also negative examples.
2. While  $T$  contains the negative examples in it and is not a lot complex, the following is repeated:
  - A literal searched to add to the rule body.

- A new training set is formed, named  $T_1$  by adding all examples that are covered by the rule. If any new variables are introduced by the rule, then all examples covered by the new variable are also be added to  $T_1$ .
  - $T$  is replaced with  $T_1$ .
3. The final rule is pruned by removing any unnecessary literals.

FOIL uses a greedy approach in adding literals. So when a literal is added to the rule, then other literals are not looked at. Adding an appropriate literal is always a key. This is performed using two techniques:

1. **Gainful Literals** – Literals that help to remove any negative or unwanted examples from the training set.
2. **Determinate Literals** – Literals that introduce new variables that are useful for future literals.

A literal that satisfies any of the above techniques, is determined appropriate to be added to the rule. FOIL also employs a backtracking mechanism to reach a point that provided a better gain instead of searching ahead with literals that have no further gain.

This is achieved by checkpoints, when an added literal provides only some improvement to the whole rule compared to another literal. Literals that completes the rule are stored as a temporarily literals. In case the final rule is not an improvement of the rule that is stored temporarily, then the stored rule becomes the final rule.



The search space of literals is limited to the following constrictions:

1. The literal must contain at least one existing variable. It can be a new variable that is introduced by a previous literal.
2. In case the new literal is same as head of the rule, then possible arguments are restricted in order to prevent unexpected or uncontrolled recursions.
3. Gainful literals must allow pruning similar to alpha-beta

The first and third constrictions above are defined so as to reduce unnecessary search of literals. The second is defined to prevent unwanted literals that produce infinite recursions.

#### **2.5.4 SEQUENTIAL COVERING ALGORITHM**

The Sequential Covering Algorithm uses the best first method to learn rules for Inductive Logic Programming. The search process is limited here as it focuses best possible solutions first.

Sequential covering algorithm is a rule-based algorithm following a process of building hypothesis that would cover all positive examples and as few negative examples as possible [3]. It mimics the learn-one rule. In learn-one rule, generally a rule is learnt with a high accuracy along with any possible coverage. This algorithm deletes the positive examples covered by the rule and repeats the process of learn-one rule again.

The basic algorithm for Sequential Covering Algorithm is as follows [21]:

- Learn one rule: This performs a general to specific search for a rule, which is highly accurate but has a lower coverage. It also learns a set of rules.
- Remove the data covered: The data covered in the algorithm is removed from the search space to make the process efficient.
- Repeat: These steps are repeated until the best possible rule is generated

The algorithm takes four input arguments. Target rule is the rule to be generated, attributes are facts that are used to generate a rule, examples are positive and, negative, and threshold is performance level up to where the process has to continue. The learned rule initially is empty. The Sequential Covering Algorithm calls Learn One Rule method, which returns the best possible rule as output. This algorithm runs as a black box to the input data from the GUI developed in the thesis.

## **2.6 Graphical User Interface**

A Graphical User Interface (GUI) is a type of interface that allows the users to interact with any electronic device. It takes advantage of the computer's graphical capabilities in order to make a program easier to use with the assistance of menu, icons and connectivity to other devices.

Graphical User Interfaces are designed keeping in mind the following [14]:

1. To provide an interactive data exploration tool to the users, which allow them to view data, interact with it and not just make an upfront selection,
2. To help the user in creating very complex models and avoiding any typographical mistakes,
3. To enable the users to assess and analyze the results as well as have an interactive visualization of them.

The application developed here is a cross-platform Java based graphical user interface that has been built using the Eclipse framework. The application contains some modules and it provides various aspects of the functionality of the application as a whole.

Some of the functionality of the tool is as under:

1. The project structure consists of a main folder with all subfolders consisting of all the dataset and sample input and output files
2. The data can be selected from huge datasets, such as the Diabetes Data Set, Breast Cancer data set from the UCI Machine Learning Repository
3. The data selection process is interactive and allows the user to select different columns from different tables
4. The selected parameters and the data is then converted to a text file, which goes as input to the rest of the code
5. The output is an aggregation of rules that is specific to the input data.

## **3. Implementation**

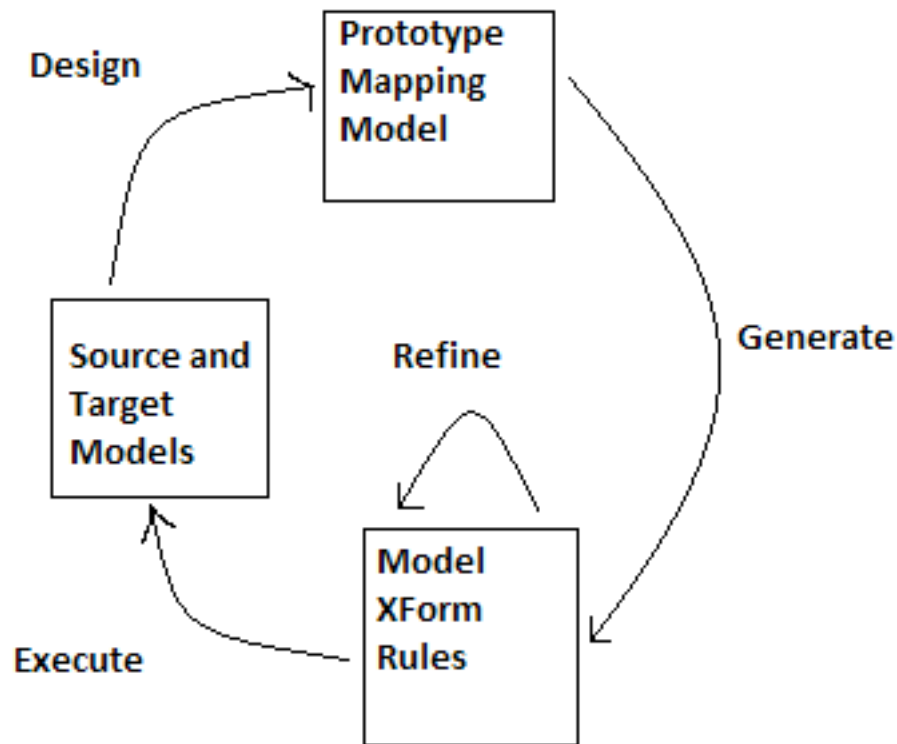
The implementation chapter discusses an example in detail. The example discussed here is Model transformation. It is a novel approach in model-driven software engineering, which is discussed in details later here.

### **3.1 Model Transformation**

Model transformation is a novel approach in model-driven software engineering in order to derive certain transformation rules, starting from an initial prototype of related source and target models that demonstrate the critical cases of model transformation issue in a complete declarative format [10].

The efficiency in designing the automated model transformations between modeling languages is becoming a major challenge to model-driven engineering (MDE) [10]. The trend in evolution of the model transformation languages is characterized by the gradual increase in the abstraction level of declarative, rule-based formalisms. There exists a deficiency in the languages and tools which is that the transformation language used is quite different from the source and target models transformed by them. So as a result, the designers who transform this model, have to understand the transformation problem, that

means the process of mapping source models to target models and they should also have significant knowledge for the transformation language itself to formalize the solution.



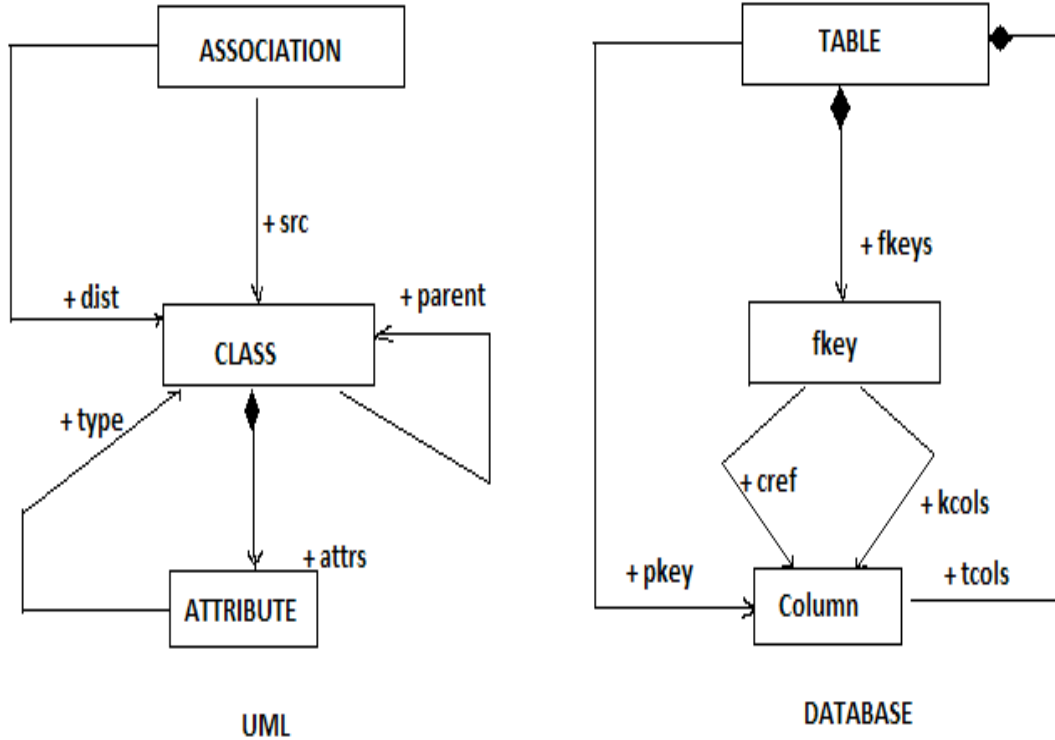
**Figure 3.1: Process overview of Model Transformation by Example**

Model transformation by example (MTBE) approach was introduced in order to bridge the gap in transformation design [10]. This approach is aimed to derive the model transformation rules from an initial prototype which is a set of interrelated source and target models that explain the critical cases of the model transformation problem. A main advantage of this approach is that the designers use the concepts of the source and target

modeling languages for the specifics of the transformation, while the implementation rules are generated quite semi-automatically. The interconnection between the source and the target models are potentially generated as a result of mapping a metamodel. Following it, the actual contextual conditions that are used in the transformation rules are derived. These conditions are based upon the prototypical source and target model pairs [10].

ILP lies at the intersection point of inductive learning and logic programming and it aims to construct the first-order clausal theories from examples and based on background knowledge. It uses induction instead of deduction as the basic mode of inference [10]. First-order logic foundations help to achieve a higher level of automation for MTBE compared to an intuitive solution based on our initial experiences.

The main example discussed here is the object-relational mapping problem. Here the UML class diagrams are mapped into relational database tables. The source language is set to UML and the target language is set to the relational databases. These are captured by the respective metamodels. In order to avoid the mixing of the notions of UML class diagrams and metamodels, the concepts of the metamodel are referred using node types and edge types for classes and associations, respectively.



**Figure 3.2: Source and target metamodels**

The UML class diagrams consist of the class nodes that are arranged into a hierarchy by inheritance. Here, the classes contain attribute nodes (`attrs`), which are typed over classes (`type`). Directed edges lead from a source (`src`) class to a destination (`dist`) class. Relational databases consist of table nodes, which are composed of column nodes by `tcols` edges. Each table has a single primary key column (`pkey`). Foreign key (`fkey`) constraints can be assigned to tables (`fkeys`). Such a key refers to columns (`cref`) of another table, and it is related to the columns of local referring table by edges (`kcols`) as described by Figure 3.2 [10].

The object-relational mapping has a top-level UML class, which is the top-most class in the inheritance tree, and is projected into a database table. There are two additional columns, derived for each top-level class. One of the classes store a unique identifier i.e. primary key, and the other class stores the type of the instances. Each attribute of a UML class appears as a table column, related to the top-level class ancestor. The foreign key constraints are used to maintain structural consistency of storing only valid object instances. Each UML association points to a table that has two columns pointing to the related tables between the source and the target classes of the association connected by foreign key constraints.

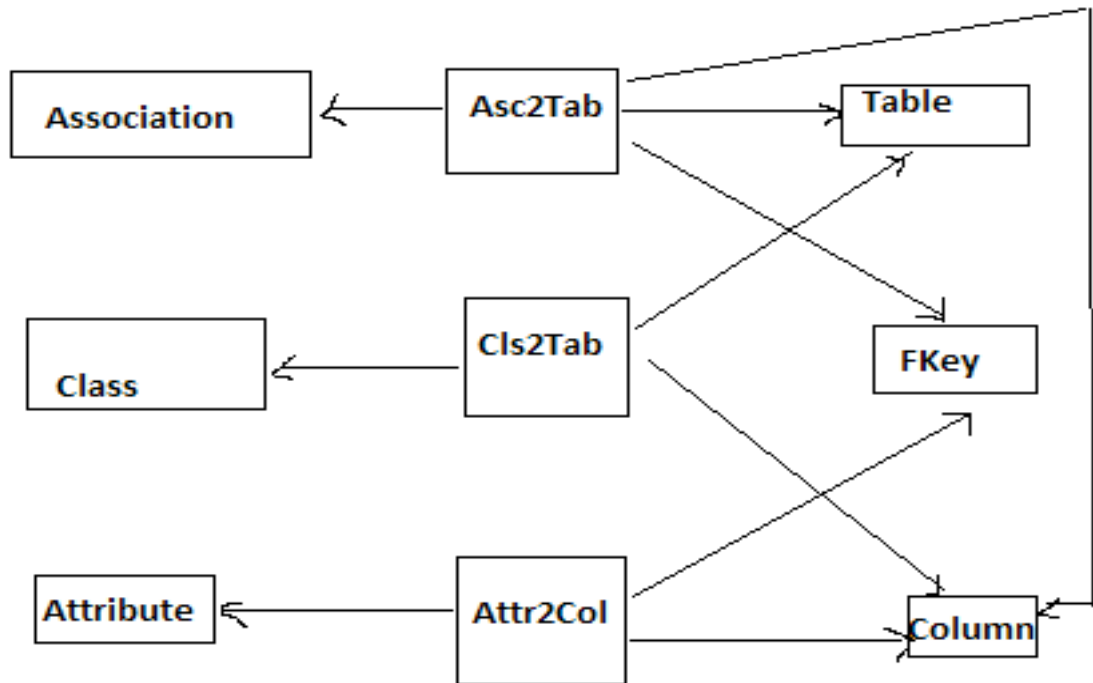
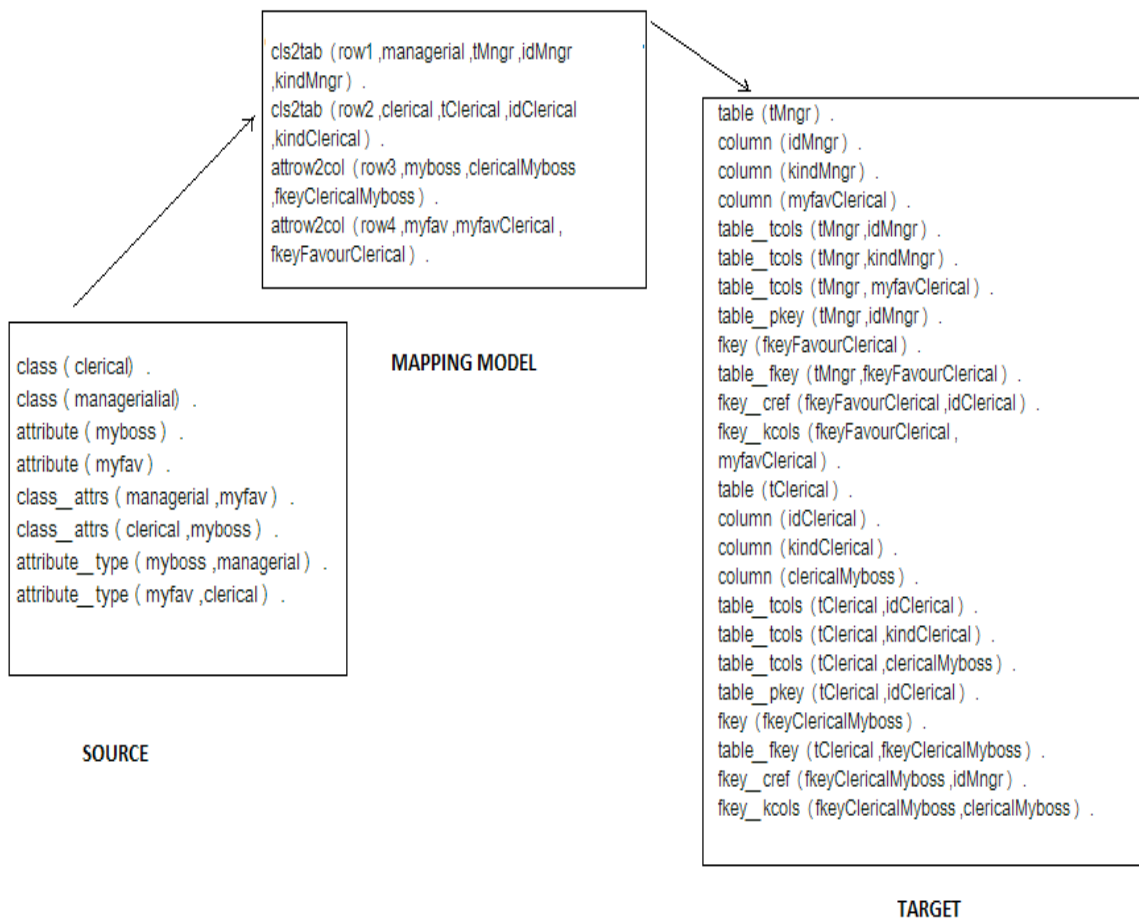


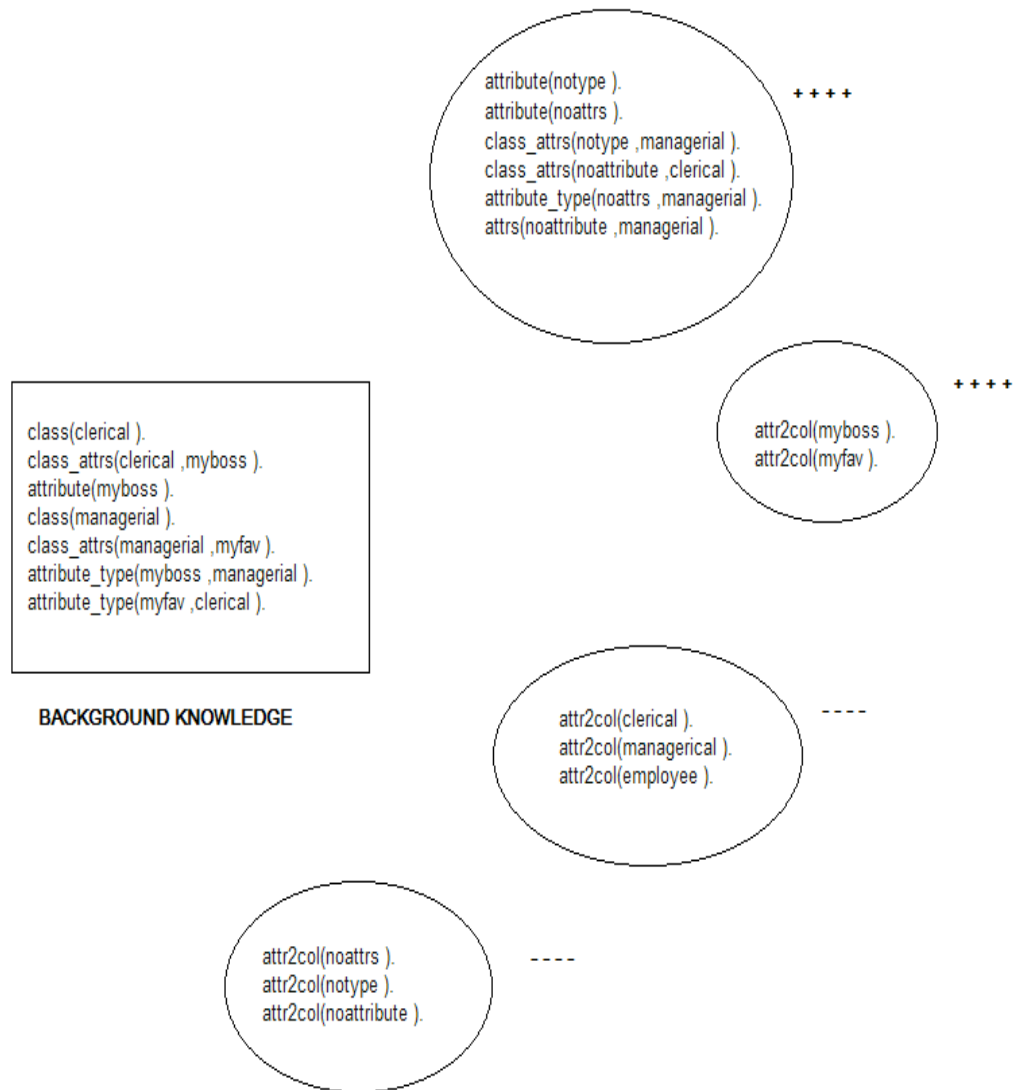
Figure 3.3: Mapping Metamodel



In many model transformation approaches, source and target metamodels are complemented with another metamodel, which specifies the interconnections allowed between elements of the source and target languages. This metamodel is referred to as mapping metamodel and the Instances of this metamodel are called (prototype) mapping models.



**Figure 3.4: Clauses in the model transformation**

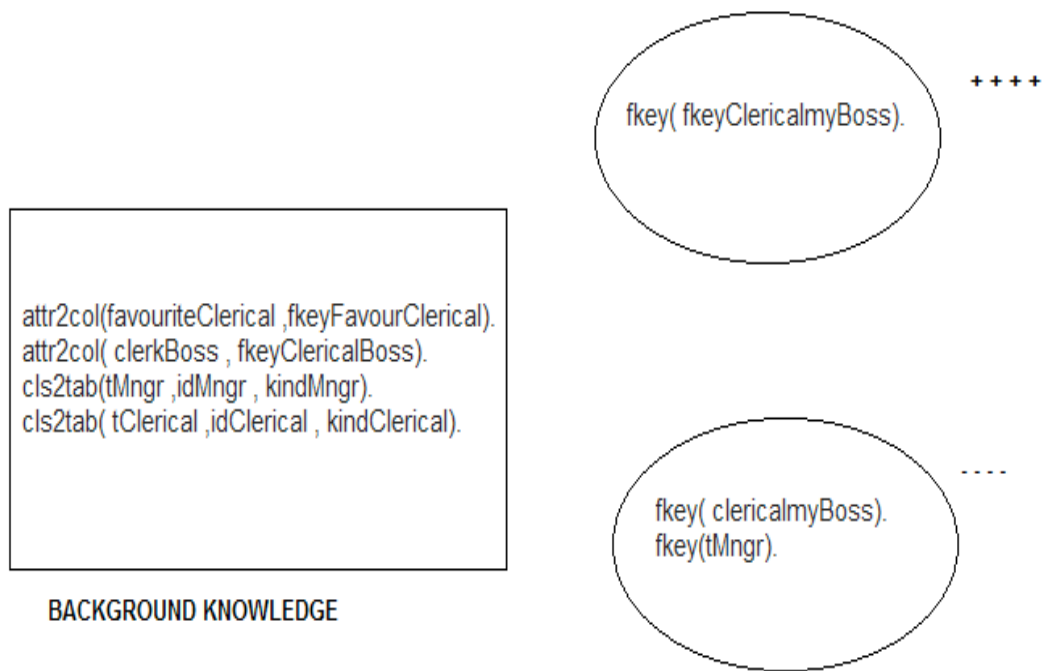


**Figure 3.5: Background Knowledge with the different facts**

Taking into account the clauses and the facts, the ILP engine derives the following rule:

```
attr2col(A) :- attribute(A), class_attrs(B,A),
attribute_type(A,C).
```

Upon counter analyzing the target model, the following can be derived:



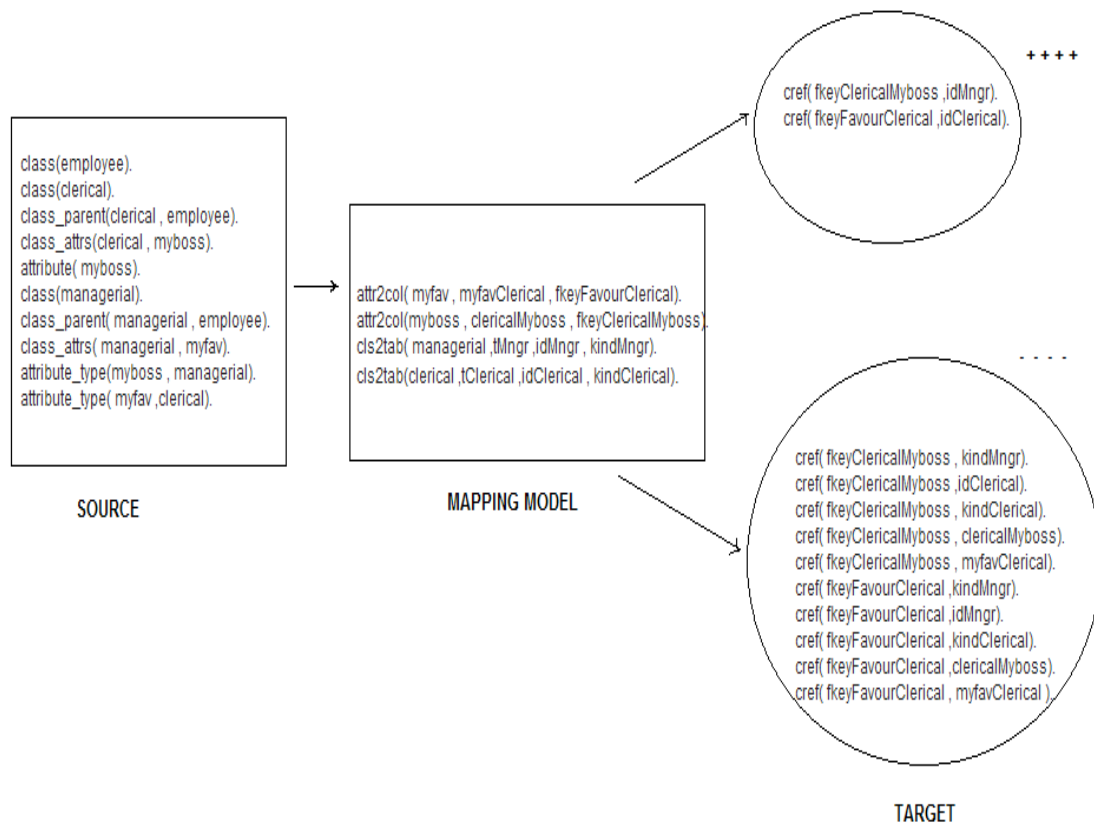
**Figure 3.6: Results after counter analyzing**

As a result, the ILP engine will derive the following hypothesis for the target model:

```
fkey(A) :- attr2col(B,A) .
```

## Analyzing Connectivity:

The final step in model transformation is analyzing the connectivity of the source and target models.



**Figure 3.7: Analyzing Connectivity**

## 3.2 Java

Java is an object-oriented language. Any software application written in Java helps in making use of all the concepts of object technology. Following are some of the other reasons why Java was selected as our programming language for the project : [19]

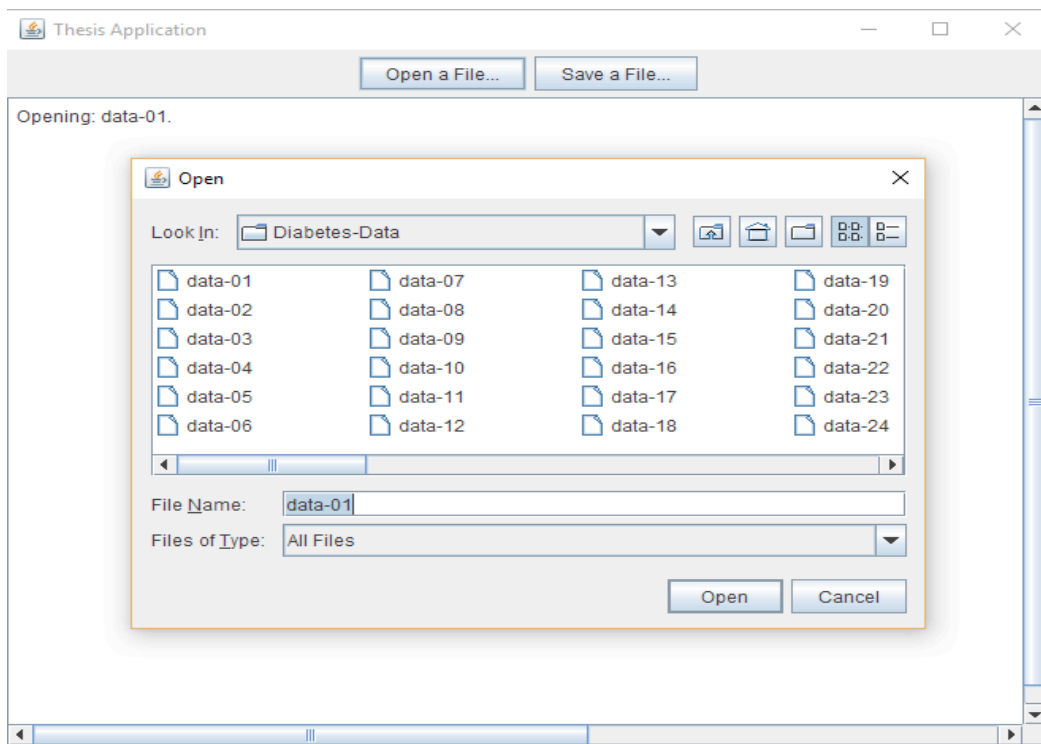
1. The source code of Java is compiled into bytecode, which is platform independent. So the Java programs can run on any platform using a Java virtual machine (incorporated with Java interpreter and run-time system).
2. Standard Java library contains the Swing package with a complete collection of GUI elements.
3. Java has the multi-threading feature, which enables parts of a program to be executed in a number of separate threads. Every thread has its independent code execution process, which does not depend on the other threads. This enhances the performance of the programs, by optimizing the resources of the computer specially while building a Graphical User Interface.

## 4. RESULTS

This section discusses the proof of concept for creating a Graphical User Interface which can read data from a database and send it to the ILP system. The system then processes the data and returns the applied rule. The system then displays the rule.

### 4.1 The GUI

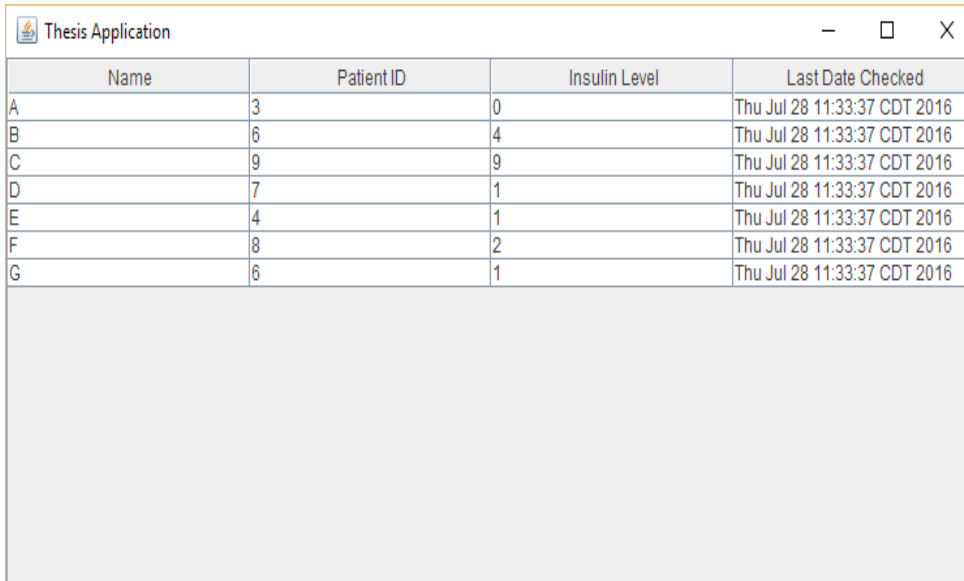
The given screenshot displays the GUI of the application. Here, the user can select a particular database and select the tables from it.



**Figure 4.1 Select a table from a database to select data**

## 4.2 Selecting Data

The data from the selected tables are sent to a text file. This text file contains all the data from the all the selected rows and columns.



The image shows a screenshot of a window titled "Thesis Application". Inside the window is a table with four columns: "Name", "Patient ID", "Insulin Level", and "Last Date Checked". The table contains seven rows of data, labeled A through G. Below the table is a large, empty rectangular area.

Name	Patient ID	Insulin Level	Last Date Checked
A	3	0	Thu Jul 28 11:33:37 CDT 2016
B	6	4	Thu Jul 28 11:33:37 CDT 2016
C	9	9	Thu Jul 28 11:33:37 CDT 2016
D	7	1	Thu Jul 28 11:33:37 CDT 2016
E	4	1	Thu Jul 28 11:33:37 CDT 2016
F	8	2	Thu Jul 28 11:33:37 CDT 2016
G	6	1	Thu Jul 28 11:33:37 CDT 2016

**Figure 4.2: The data from the table gets transferred into a text file**

### 4.3 Examples of Results

The input is received in the form of a text file and then the sequential algorithm is applied to it. This results in the output being generated in the form of rules. This can be observed from the figure below.

```

Final Rule --> act(column1) :-atm(column1,newVar1,newVar2,const_27,newVar4), bond(column1,newVar1,newVar555,const_7)
Final Scores --> 70 -11Accuracy ---> 86.41975308641975 Coverage ---> 56.0
Final Rule --> act(column1) :-atm(column1,newVar1,const_c,const_27,newVar4), bond(column1,newVar863,newVar1,const_7)
Final Scores --> 70 -11Accuracy ---> 86.41975308641975 Coverage ---> 56.0
Final Rule --> act(column1) :-atm(column1,newVar1,newVar2,const_27,newVar4), bond(column1,newVar552,newVar1,const_7)
Final Scores --> 70 -11Accuracy ---> 86.41975308641975 Coverage ---> 56.0
Final Rule --> act(column1) :-atm(column1,newVar1,newVar2,const_27,newVar4), bond(newVar580,newVar581,newVar1,const_7)
Final Scores --> 70 -11Accuracy ---> 86.41975308641975 Coverage ---> 56.0
Final Rule --> act(column1) :-atm(column1,newVar1,newVar2,const_27,newVar4), bond(column1,newVar1,newVar555,newVar556)
Final Scores --> 70 -11Accuracy ---> 86.41975308641975 Coverage ---> 56.0
Final Rule --> act(column1) :-atm(column1,newVar1,const_c,const_27,newVar4), bond(newVar891,newVar892,newVar1,const_7)
Final Scores --> 70 -11Accuracy ---> 86.41975308641975 Coverage ---> 56.0
Final Rule --> act(column1) :-atm(column1,newVar1,const_c,const_27,newVar4), bond(column1,newVar1,newVar866,newVar867)
Final Scores --> 70 -11Accuracy ---> 86.41975308641975 Coverage ---> 56.0
Final Rule --> act(column1) :-atm(column1,newVar1,const_c,const_27,newVar4), bond(column1,newVar863,newVar1,newVar864)
Final Scores --> 70 -11Accuracy ---> 86.41975308641975 Coverage ---> 56.0
Final Rule --> act(column1) :-atm(column1,newVar1,newVar2,const_27,newVar4), bond(newVar590,newVar1,newVar591,const_7)
Final Scores --> 70 -11Accuracy ---> 86.41975308641975 Coverage ---> 56.0
Final Rule --> act(column1) :-atm(column1,newVar1,const_c,const_27,newVar4), bond(column1,newVar858,newVar859,const_2)
Final Scores --> 70 -11Accuracy ---> 86.41975308641975 Coverage ---> 56.0

```

**Figure 4.3: The output in the form of rules learnt from the input data**



## 5. CONCLUSIONS AND FUTURE WORK

It is inevitable that the amount of data that is created every moment is huge and in order to work through them, we need to find patterns, anomalies and relationships between large data sets. Then applying certain rules on the data help us to figure out the relationships among them, in addition to providing knowledge about the dataset.

ILP involves using powerful learning methods to find a solution to a problem where the data can be visualized as multiple tables with relationships among them.

The thesis aims at creating a GUI, which would help novice users to define the ILP problems. They can connect to an existing database and select data from there to be transferred to the prolog code which implements sequential covering algorithm.

Future work for the thesis would be implementing other ILP algorithms such as Learn One Rule Algorithm or FOIL Algorithm on the basis of the already devised sequential algorithm. Also the GUI can be accordingly modified to select and use a specific algorithm. The GUI could also be modified to select data from more than one database. The output of the code could be produced in the GUI.

## 6. REFERENCES

1. Muggleton, Stephen, and Luc De Raedt. "Inductive logic programming: Theory and methods." *The Journal of Logic Programming* 19 (1994): 629-679.
2. De Raedt, Luc. "An inductive logic programming query language for database mining." *Artificial Intelligence and Symbolic Computation*. Springer Berlin Heidelberg, 1998.
3. Wrobel, Stefan. "Inductive logic programming for knowledge discovery in databases." *Relational data mining*. Springer Berlin Heidelberg, 2001. 74-101.
4. Page, David. "ILP: Just do it." *Computational Logic—CL 2000*. Springer Berlin Heidelberg, 2000. 25-40.
5. Paes, Aline, et al. "ILP through propositionalization and stochastic k-term DNF learning." *Inductive Logic Programming*. Springer Berlin Heidelberg, 2006. 379-393.
6. Muggleton, Stephen. "Inductive logic programming: issues, results and the challenge of learning language in logic." *Artificial Intelligence* 114.1 (1999): 283-296.
7. Srinivasan, Ashwin. "A learning engine for proposing hypotheses (Aleph)." (1999): 247262.
8. Knight, Kevin. "Automating knowledge acquisition for machine translation." *AI Magazine* 18.4 (1997): 81.

9. Hall, Mark, et al. "The WEKA data mining software: an update." *ACM SIGKDD explorations newsletter* 11.1 (2009): 10-18.
10. Varró, Dániel, and Zoltán Balogh. "Automating model transformation by example using inductive logic programming." *Proceedings of the 2007 ACM symposium on Applied computing*. ACM, 2007.
11. Plotkin, Gordon D. "A note on inductive generalization." *Machine intelligence* 5.1 (1970): 153-163.
12. Quinlan, J. Ross. "Learning logical definitions from relations." *Machine learning* 5.3 (1990): 239-266.
13. Muggleton, Stephen, and Wray Buntine. "Machine invention of first-order predicates by inverting resolution." *Proceedings of the fifth international conference on machine learning*. 1992.
14. Snellenburg, Joris, et al. "Glotaran: a Java-based graphical user interface for the R package TIMP." *Journal of Statistical Software* 49.3 (2012).
15. Bederson, Benjamin B., Jon Meyer, and Lance Good. "Jazz: an extensible zoomable user interface graphics toolkit in Java." *Proceedings of the 13th annual ACM symposium on User interface software and technology*. ACM, 2000.
16. Quinlan, J. Ross, and R. Mike Cameron-Jones. "FOIL: A midterm report." *European conference on machine learning*. Springer Berlin Heidelberg, 1993.
17. Quinlan, J. Ross. "Learning logical definitions from relations." *Machine learning* 5.3 (1990): 239-266.

18. Boytcheva, Svetla. "Overview of inductive logic programming (ILP) systems." *Cybernetics and Information Technologies* 1 (2002): 27-36.
19. Naressi, A., et al. "Java-based graphical user interface for the MRUI quantitation package." *Magnetic resonance materials in physics, biology and medicine* 12.2-3 (2001): 141-152.
20. Wikipedia – Inductive Logic Programming - Wikipedia contributors. "Inductive logic programming." *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 2 Feb. 2017. Web. 8 Mar. 2017.
21. Sequential Covering Algorithm. [http://www.d.umn.edu/~rmaclin/cs8751/Notes/L08\\_Rule\\_Learning.pdf](http://www.d.umn.edu/~rmaclin/cs8751/Notes/L08_Rule_Learning.pdf).

## 7. APPENDIX

Following are some of the code snippets for the developed Graphical User Interface:

### 7.1 Application Menu

```
/*
 * To change this license header, choose License
 * Headers in Project Properties.
 * To change this template file, choose Tools |
 * Templates
 * and open the template in the editor.
 */
package ILPGUI_Thesis;

/**
 *
 * @author Priyankana
 */

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.SwingUtilities;
import javax.swing.filechooser.*;

public class ThesisApp1 extends JPanel implements
ActionListener {

    JFileChooser fc;
    JMenuItem jmiOpen, jmiClose, jmiSave, jmiExit;
    JTextArea log;

    ThesisApp1 () {

        super(new BorderLayout ());
```

```
JFrame f = new JFrame("Thesis Application");
f.setSize(600, 600);

f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JMenuBar jmb = new JMenuBar();

log = new JTextArea(5, 20);
log.setMargin(new Insets(5, 5, 5, 5));
log.setEditable(false);
JScrollPane logScrollPane = new JScrollPane(log);

JMenu jmFile = new JMenu("File");
jmiOpen = new JMenuItem("Open");
jmiClose = new JMenuItem("Close");
jmiSave = new JMenuItem("Save");
jmiExit = new JMenuItem("Exit");
jmFile.add(jmiOpen);
jmFile.add(jmiClose);
jmFile.add(jmiSave);
jmFile.addSeparator();
jmFile.add(jmiExit);
jmb.add(jmFile);

JMenu jmOptions = new JMenu("Options");
JMenu a = new JMenu("A");
JMenuItem b = new JMenuItem("B");
JMenuItem c = new JMenuItem("C");
JMenuItem d = new JMenuItem("D");
a.add(b);
a.add(c);
a.add(d);
jmOptions.add(a);

JMenu e = new JMenu("E");
e.add(new JMenuItem("F"));
e.add(new JMenuItem("G"));
jmOptions.add(e);

jmb.add(jmOptions);
```

```

JMenu jmHelp = new JMenu("Help");
JMenuItem jmiAbout = new JMenuItem("About");
jmHelp.add(jmiAbout);
jmb.add(jmHelp);

jmiOpen.addActionListener(this);
jmiClose.addActionListener(this);
jmiSave.addActionListener(this);
jmiExit.addActionListener(this);
b.addActionListener(this);
c.addActionListener(this);
d.addActionListener(this);
jmiAbout.addActionListener(this);

f.setJMenuBar(jmb);
f.setVisible(true);
}

public void actionPerformed(ActionEvent e) {

    //Handle open button action.
    if (e.getSource() == jmiOpen) {
        int returnVal =
fc.showOpenDialog(ThesisApp1.this);

        if (returnVal ==
JFileChooser.APPROVE_OPTION) {
            File file = fc.getSelectedFile();

            log.append("Opening: " +
file.getName() + ".");
        } else {
            log.append("Open command cancelled by
user.");
        }

log.setCaretPosition(log.getDocument().getLength());

    //Handle save button action.
    } else if (e.getSource() == jmiSave) {
        int returnVal =

```

```

fc.showSaveDialog(ThesisApp1.this);
    if (returnVal ==
JFileChooser.APPROVE_OPTION) {
        File file = fc.getSelectedFile();
        //This is where a real application
would save the file.
        log.append("Saving: " + file.getName()
+ "." );
    } else {
        log.append("Save command cancelled by
user." );
    }

log.setCaretPosition(log.getDocument().getLength());
    }
}
public static void main(String args[]) {
    new ThesisApp1();
}
}

```

## 7.2 Input from the Database

```

/*
 * To change this license header, choose License
Headers in Project Properties.
 * To change this template file, choose Tools |
Templates
 * and open the template in the editor.
 */
package ILPGUI_Thesis;

/**
 *
 * @author Priyankana
 */

import java.awt.Dimension;

```



```
import java.util.Date;

import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.ListSelectionModel;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;
import javax.swing.table.TableModel;

public class JavaApplication2 implements
ListSelectionListener {

    String[] headings = { "Name", "Customer ID", "Order
#", "Status" };

    Object[][] data = { { "A", new Integer(3), "0",
new Date() },
        { "B", new Integer(6), "4", new Date() },
{ "C", new Integer(9), "9", new Date() },
        { "D", new Integer(7), "1", new Date() },
{ "E", new Integer(4), "1", new Date() },
        { "F", new Integer(8), "2", new Date() },
{ "G", new Integer(6), "1", new Date() } };

    JTable jtabOrders = new JTable(data, headings);

    TableModel tm;

    JavaApplication2() {
        JFrame jfrm = new JFrame("JTable Event Demo");
        jfrm.setSize(400, 200);

jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

jtabOrders.setPreferredScrollableViewportSize(new
Dimension(420, 62));
```

```

        ListSelectionModel rowSelMod =
jtabOrders.getSelectionModel();

        ListSelectionModel colSelMod =
jtabOrders.getColumnModel().getSelectionModel();

        rowSelMod.addListSelectionListener(this);
        colSelMod.addListSelectionListener(this);

        tm = jtabOrders.getModel();

        tm.addTableModelListener(new TableModelListener()
{
            public void tableChanged(TableModelEvent tme) {
                if (tme.getType() == TableModelEvent.UPDATE) {
                    System.out.println("Cell " +
tme.getFirstRow() + ", " + tme.getColumn() + "
changed."
                    + " The new value: " +
tm.getValueAt(tme.getFirstRow(), tme.getColumn()));
                }
            }
});
jfrm.add(new JScrollPane(jtabOrders));
jfrm.setVisible(true);
}

public void valueChanged(ListSelectionEvent le) {
    String str = "Selected Row(s): ";
    int[] rows = jtabOrders.getSelectedRows();
    for (int i = 0; i < rows.length; i++)
        str += rows[i] + " ";

    str += "Selected Column(s): ";
    int[] cols = jtabOrders.getSelectedColumns();

    for (int i = 0; i < cols.length; i++)
        str += cols[i] + " ";
}

```

```

        str += "Selected Cell: " +
jtabOrders.getSelectedRow() + ", "
        + jtabOrders.getSelectedColumn();
        System.out.println(str);
    }

    public static void main(String args[]) {
        JavaApplication2 j2 = new JavaApplication2();

    }
}

```

### 7.3 Conversion of data into a text file

```

/*
 * To change this license header, choose License
 * Headers in Project Properties.
 * To change this template file, choose Tools |
 * Templates
 * and open the template in the editor.
 */
package ILPGUI_Thesis;

/**
 *
 * @author Priyankana
 */

import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import javax.swing.table.*;
import java.io.*;
import java.util.*;

public class Admin extends JFrame {

```



```

String[height][width];
        model = new DefaultTableModel(matrix,
col);

        table = new JTable(model) {
public boolean isCellEditable(int row, int
column) {
        return false;
        }
};

}
}
catch (IOException ex) {
ex.printStackTrace();
}

DefaultTableCellRenderer cent = new
DefaultTableCellRenderer();
cent.setHorizontalAlignment( JLabel.CENTER );
table.getColumnModel().getColumn(0).
setCellRenderer(cent);
table.getColumnModel().getColumn(1).
setCellRenderer(cent);
table.getColumnModel().getColumn(2).
setCellRenderer(cent);
table.getColumnModel().getColumn(3).
setCellRenderer(cent);
table.getColumnModel().getColumn(4).
setCellRenderer(cent);
table.getColumnModel().getColumn(5).
setCellRenderer(cent);
table.getColumnModel().getColumn(6).
setCellRenderer(cent);
table.getColumnModel().getColumn(7).
setCellRenderer(cent);
table.getColumnModel().getColumn(0).
setPreferredWidth(100);
table.getColumnModel().getColumn(1).
setPreferredWidth(100);
table.getColumnModel().getColumn(2).
setPreferredWidth(100);
table.getColumnModel().getColumn(3).

```

```

setPreferredWidth(100);
    table.getColumnModel().getColumn(4).
setPreferredWidth(50);
    table.getColumnModel().getColumn(5).
setPreferredWidth(100);
    table.getColumnModel().getColumn(6).
setPreferredWidth(50);
    table.getColumnModel().getColumn(7).
setPreferredWidth(120);

    scroll = new JScrollPane(table);
    scroll.setLocation(20, 320);
    scroll.setSize(750,150);

    Container pane = getContentPane();
    pane.setLayout(null);
    pane.add(back);
    pane.add(scroll);

    /*setIconImage(new
ImageIcon("/images/icon.jpg").getImage());
    setLayout(new BorderLayout());
    JLabel www = new JLabel(new
ImageIcon(getClass().
getResource("/images/admin.jpg")));
    add(www);*/
    setDefaultCloseOperation(DISPOSE_ON_CLOSE);
    setTitle("Account Profile");
    setSize(800,550);
    setResizable(false);
    setVisible(true);
    setLocation(110, 30);
}
public static void main(String[] args) {
    new Admin ();
}
}

```

## 7.4 Read from the text file

```
/*
 * To change this license header, choose License
 * Headers in Project Properties.
 * To change this template file, choose Tools |
 * Templates
 * and open the template in the editor.
 */
package ILPGUI_Thesis;

/**
 *
 * @author Priyankana
 */
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
import java.io.*;
public class InsertFileDataToJTable extends
AbstractTableModel {
    Vector data;
    Vector columns;

    public InsertFileDataToJTable() {
        String line;
        data = new Vector();
        columns = new Vector();
        try {
            FileInputStream fis = new
FileInputStream("/file.txt");
            BufferedReader br = new
BufferedReader(new InputStreamReader(fis));
            StringTokenizer st1 = new
StringTokenizer(br.readLine(), " ");
            while (st1.hasMoreTokens())

columns.addElement(st1.nextToken());

while ((line = br.readLine()) != null) {
```

```
StringTokenizer st2 = new StringTokenizer(line, " ");
while (st2.hasMoreTokens())
    data.addElement(st2.nextToken());

        br.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public int getRowCount() {
    return data.size() / getColumnCount();
}

public int getColumnCount() {
    return columns.size();
}

public Object getValueAt(int rowIndex, int
columnIndex) {
    return (String) data.elementAt((rowIndex *
getColumnCount())
+ columnIndex);
}

public static void main(String s[]) {
    InsertFileDataToJTable model = new
    InsertFileDataToJTable();
    JTable table = new JTable();
    table.setModel(model);
    JScrollPane scrollpane = new JScrollPane(table);
        JPanel panel = new JPanel();
        panel.add(scrollpane);
        JFrame frame = new JFrame();
        frame.add(panel, "Center");
        frame.pack();
        frame.setVisible(true);
    }
}
```