The Aleph system made easy

By

João Paulo Duarte Conceição

INTEGRATED MASTER OF ELECTRIC ENGINEERING
AND COMPUTING 2008

in the

FACULTY OF ENGINEERING

of the

UNIVERSITY OF PORTO

Advisors

Rui Camacho
rcamacho@fe.up.pt
Faculty of Engineering, University of Porto, Portugal

Lubomir Popelinsky
popel@fi.muni.cz
Faculty of Informatics, Masaryk University, Czech Republic

ABSTRACT

The primary objective of this report is twofold: i) present the necessary background knowledge required to successfully carry out the dissertation work and; ii) propose a work plan. The dissertation work will be the development of a graphical interface for the Aleph system. The interface will hide the details of the Aleph system allowing non ILP[1] researchers to perform data analysis with Aleph. This report gives an introduction to Inductive Logical Programming with emphasis to the Aleph System. It describes the state of the art in ILP, and provides a description of the Aleph system. A work plan and final conclusions are presented at the end.

[1]Inductive Logic Programming

Contents

1 Introduction

1.1 Motivation

Nowadays we are surrounded by a huge amount of information. That quantity has the tendency to continue to increase. The hard disks of the computers have more capacity to store information and with the prices decreasing, it leads to a exponential store of information. Nowadays, electronic equipments store information raging from our decisions on supermarkets to our financial habits [1]. In general, all our choices are stored in databases. The amounts of information makes it impossible to be analised by human experts. Automatic data analysis is therefore required. This situation lead to the appearance of Data Mining [2] [2]. In Data Mining, the information is stored electronically and the searches are autonomously done by a computer based in some patterns with the objective of resolve a problem and simultaneously understand the content of the database [3]. In this context were developed techniques and algorithms of Artificial Intelligence that permit the computers to learn. The primary goal of these techniques is automatically extract information from the database based on computational and statistical methods. Simultaneously adopt these decisions to the problem we want to resolve in order to optimize his functional procedure and give the possibility to get conclusions based on founded patterns [4]. In the research area of Machine Learning and the developement of logical programs is Inductive Logic Programming (ILP) [5]. There are many ILP systems, although to use most of them it is necessary to know-how ILP programming works [6]. The ILP system called "A Learning Engine for Proposing Hypotheses" (Aleph) is one of these systems and is where this project takes place.

1.2 Goals

The primary objective of this dissertation is the study, project, development and test of an interface for the Aleph ILP system. This interface should be usable by any non ILP researcher and it should be possible to construct models interactively by tuning them without any knowledge about Aleph's parameters, so that anyone with basic knowledge about informatics can use this system. With the Aleph system we may construct several models automatically and show them to the user for him to choose. The models

[2] We use Data Mining even when referring to the whole process of Knowledge Discovery in Databases – KDD

should be presented in a language very close to English. We choosed this language, because it's the universal language and can be understood by a large proportion of population in the world.

1.3 Structure of the Report

First of all this report give us an introduction on ILP, describing basic concepts, its dimensions and a brief description of some ILP systems. After we understand the difference between various ILP systems and know how ILP works, the Aleph system is debriefed. In this section it's described the basic algorithm of the Aleph system, its basic functional procedures and parameters. It is also explained the wrapper concept and its functional model, giving an example to better understand it, due to the fact that it will be necessary to the development of the dissertation. In order to coordinate the dissertation development a working plan was set. At last, final conclusions are presented.

2 Inductive Logic Programming

2.1 Introducing ILP

ILP is a research area which intersects Machine Learning and Logic Programming. Its objective is to learn logical programs from examples and knowledge in the domain [5].

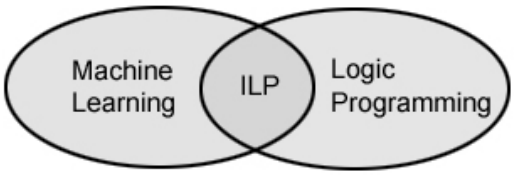

Figure 1: Intersection of Machine Learning and Logic Programming resulting ILP.

From Logic Programming ILP inherits its representation formalism, various techniques and a theoretical base. From Machine Learning inherits an experience and orientation approach for practical applications such as techniques and tools to induce hypothesis from examples and build intelligent learning machines [7]. These intelligent learning machines are learning programs which are able to change themselves in order to perform more efficiently and/or more accurately at a given task [5]. The ILP differs from the Machine Learning methods because the use of the representation language and its ability to use knowledge in the domain. This knowledge has a very important place to the learner, which task is, to find from examples an unknown relationship in terms of relations already known from that domain [5]. The knowledge of the domain is used in the construction of hypothesis it's a very important characteristic in ILP. In one hand, when this knowledge it's relevant it can substantially better the results of the learner in terms of precision, efficiency and its potential knowledge induced. On the other hand, some of this knowledge is irrelevant and will have opposite effects. The art of the ILP is to select and formulate the background knowledge to be used in the learner task [7].

2.2 Problems

In a general way ILP can be described by a knowledge theory from the initial background knowledge and some examples $E = E^+ \cup E^-$, where E^+

represents the positives examples and E^- represents the negatives examples of the concept to be learned. The objective of the ILP is to induce an hypothesis H that with the given background knowledge B, explains the examples E^+ and is consistent with E^- [5]. Although in most of the problems, the given background knowledge, the examples and the hypothesis should satisfy a joint of syntactic restrictions S, called bias of the language. That bias language defines the space of formulas used to represent hypothesis and can be considered as part of the background knowledge. The empiric learner of a single concept (predicate) in ILP can be formulated in this way:

Data:

- A set of examples E described in a language Le with: positive examples, E^+ and negative examples, E^-;
- An unknown predicate p, specifying the relationship to be learned;
- A language description of hypothesis, Lh, specifying the syntactic restrictions in the definition of the predicate p;
- The bias S, that define the space of hypothesis;
- The theory of the background knowledge B, described in a language Lb, defining predicates that can be used in the definition of p and can give additional information about arguments from the examples of the target relation.
- A operator between Le and Lh with relationship to Lb which determine if an example it's covered by a closure expressed in Lh.

Find:

- A definition H for p, expressed in Lh, so that $B \wedge H = E^+$ and $B \wedge H \neq E^-$;

2.3 Completeness and consistency of a hypothesis

After we choose the examples and concepts, it's necessary to verify if a given example belongs to that concept. When that condition it's satisfied we say that the description of the concept covers the description of the example, or that the description of the example it's covered by the description of the

concept. [5] The problem of the learner to a single concept C, described by examples, can be defined has:

Given a joint E, with positive and negative examples of a concept C, find a hypothesis H, described in a given language of description of concepts Lh, so that:

- All positive example $e \in E^+$ it's covered by H, and
- Neither negative example $e \in E^-$ it's covered by H.

To this test, a function $covers(H, e)$ can be defined. This function returns true if e it's covered by H, and false otherwise [7]. This function can redefined to joint of examples in this way:

$$covers(H, E) = \{e \in E \mid covers(H, e) = true\}$$

A hypothesis H it's complete in relation to the examples E if it covers all positive examples, $covers(H, E^+) = E^+$ [5]. A hypothesis H it's consistent in relation to the examples E if it don't covers neither negative example, $covers(H, E^-) = 0$ [5]. There are four situations that can occur depending how the hypothesis H covers the negative and positive examples as shown in the Figure 2:

(a) H complete and consistent, cover all positive examples and neither negative examples;
(b) H incomplete and consistent, don't cover all positive examples and don't cover neither negative examples;
(c) H incomplete and consistent, cover all positive examples and cover some negative examples;
(d) H incomplete and inconsistent, don't cover all positive examples and cover some negative examples.

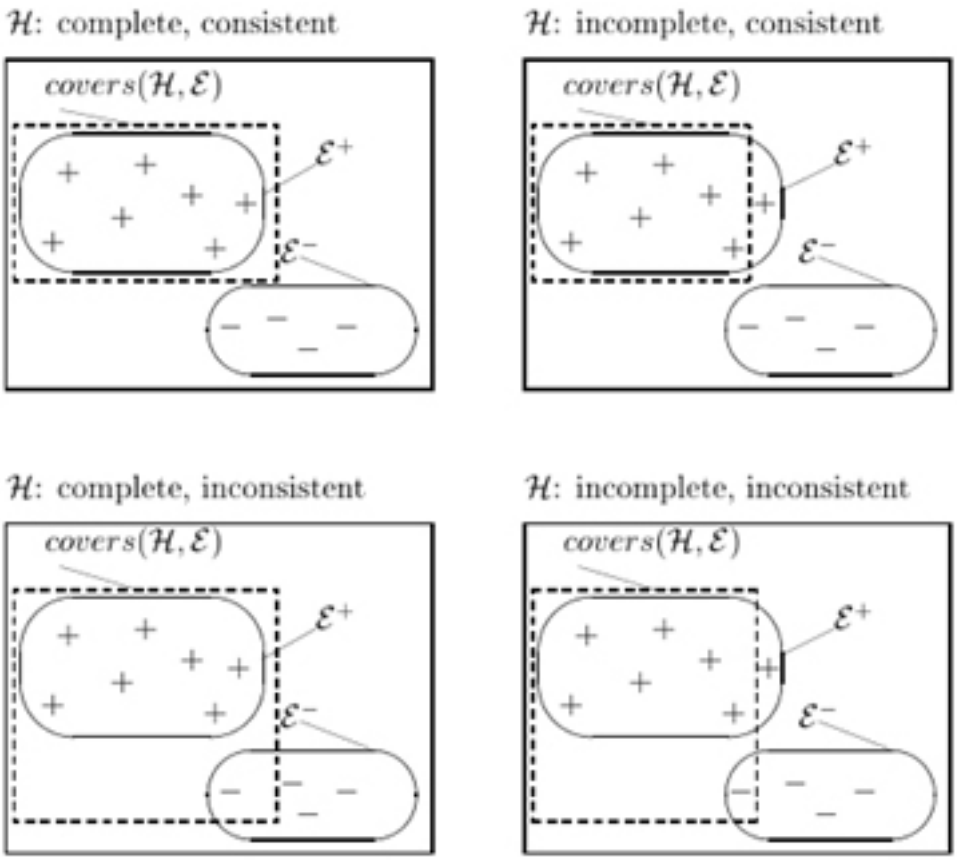

Figure 2: Completeness and consistency of a hypothesis. [5]

The cover function can be redefined to consider also the background knowledge B:

$$covers(B, H, E) = covers(H \cup B, E)$$

When we consider the background knowledge, the completeness and consistency also have to be redefined as shown below:

A hypothesis H it's complete, in relation to the background knowledge B and to the examples E, if all positive examples are covered:

$$covers(B, H, E^+) = E^+$$

A hypothesis H it's consistent in relation to the background knowledge B and to the examples, if neither negative example it's covered:

$covers(B, H, E^-) = 0$

2.4 Predictive and Descriptive ILP

Predictive ILP is one of the most common tasks in ILP and has the objective to define rules to the learner [7]. Typically that task confines E, H and B. The problem of the predictive ILP is defined by:

Given a background knowledge in the domain B, hypothesis H and a joint of examples E, one example $e \in E$ is covered by H if $B \cup H = e$. To extract an explanation in this kind of task it's necessary completeness and consistency of the hypothesis. To permit incomplete and inconsistence theories the problems should be extended to include learner rules for unperfected data, such as decision trees.

The primary goal of descriptive ILP is the learning of a theory clause [7]. Typically descriptive ILP restricts B to a joint of defined clauses and H to a joint of clauses and positive examples. To extract a rigorous explanation it is necessary that all clauses C in H be true in some predefined model. Relaxing the extraction of an explanation in a search clause and permitting theories that satisfies others acceptations like similarity and associability the descriptive ILP can be extended to incorporate associated learning rules.

2.5 Dimensions

The ILP systems can be divided, following some basic characteristics, in various dimensions [7]:

1. Can learn one single concept or multiples concepts (predicates). In case of learn one single concept, the observations are from that examples of that concept. In case of learn multiple concepts, its objective is to learn that definitions, and possibly its relations.

2. Can be necessary that all examples be given before the process of the learner takes place (batch learner or non-incremental) or can use especial individual training, given one by one, during the process of learning (incremental learner);

3. Can need a specialist or the user during the process of learning to verify the valid generalizations or classify new examples. In this case, the system is called interactive, otherwise, non-interactive.

4. Can make up new predicates. Doing that, amplifies the usable background knowledge and can be useful in the task of learn a concept. Those systems are called systems with inductive construction.

5. Can accept an empty hypothesis (theory), learning a concept from the beginning or can accept an initial hypothesis that is reviewed during the learning process.

6. Can use background knowledge in an intentional or extensional way. In one hand the extensional theory is represented only by facts (without variables), on the other hand the intentional theory have facts or variables leading to a reduced description of the concept.

Although the recently ILP systems are divided in two: in one extremity are situated the non-interactive systems with non-incremental learning. These ones learn a unique predicate from the beginning and are called empirical ILP systems. In the other extremity stay the interactive systems and theory reviewers that learn multiple predicates and are called interactive ILP systems. The empirical ILP systems tend to learn a single predicate using a large collection of examples [7]. The interactive ILP systems learn multiple predicates from a joint of examples and consults to the user [7].

2.6 Description of ILP systems

In this section will be described some ILP systems and their basic characteristics.

FOIL - it's an empirical system that learn multiple predicates from a non-interactive and non-incremental mode, realizing a top-down search in the hypothesis space

Progol - an empirical system that can learn multiple predicates in a non-interactive and non-incremental way. Realize searches from general to specific from a top-down approach.

GOLEM - empirical system that learn one unique predicate at a time from a non-interactive and non-incremental way using bottom-up search.

FORS - empirical system that realizes prediction from examples and background knowledge in problems from classes with real values. It learns a

unique predicate at a time from a non-interactive and non-incremental way doing a top-down search in the hypothesis space.

MIS - an interactive system and theory reviewer. It learns a definition of multiple predicates in a incremental way. Realize top-down search and it was the first ILP system that accept background knowledge from an intentional and extensional way.

Tilde - it's a learning system based on decision trees. These trees can be used to classify new examples or transformed in a logical program.

LINUS - an empirical ILP system, non-interactive and non-incremental. It transforms ILP systems to a attribute-value representation.

The next table shows the characteristics presents in the various ILP systems.

This table is composed by the next columns:

1. System: name of the system;
2. TD: if the system uses a top-down search;-1.5mm
3. BU: if the system uses a bottom-up search;-1.5mm
4. Predictive: if the finding task of knowledge is predictive. In this case, classification rules can be generated;-1.5mm
5. Descriptive: if the finding task of knowledge is descriptive. In this case, only true properties from the examples are observed;-1.5mm
6. Inc and N-Inc: if the system uses incremental or non-incremental learning, respectively;-1.5mm
7. Int and N-Int: if the system is the type interactive or non-interactive, respectively;-1.5mm
8. Mult-Pred: if the system can learn multiple predicates.

System	TD	BU	Predictive	Descriptive	Inc	N-Inc	Int	N-Int	Mult-Pred
Aleph		✓	✓			✓	✓		
Cigol		✓			✓		✓	✓	
Claudien	✓			✓		✓		✓	
Clint					✓		✓		✓
FOIL	✓		✓			✓		✓	
FORS	✓		✓			✓		✓	
GOLEM		✓	✓			✓		✓	
LINUS						✓	✓		
MARVIN		✓					✓	✓	
MIS	✓				✓		✓		✓
MOBAL	✓		✓			✓		✓	✓
Progol	✓		✓			✓		✓	✓
Tilde				✓					
WARMR				✓					

Figure 3: Characteristics of various ILP systems. [7]

3 A Learning Engine for Proposing Hypotheses

3.1 Aleph System

The Aleph system was developed to be a prototype to explore ideas in ILP and was written in Prolog. Nowadays Aleph use functionalities from various ILP systems like: Progol, FOIL, FORS, Indlog, MIDOS, SRT, Tilde e WARMR [8]. The Aleph has a powerful representation language that allows to represent complex expressions and simultaneously incorporate new background knowledge easily. Aleph also let choose the order of generation of the rules, change the evaluation function and the search order [7]. Allied to all this characteristics the Aleph system is open source making it a powerful resource to all ILP researchers.

3.2 The basic algorithm

The Aleph follows a very simple procedure that can be described in 4 steps [8]:

1. Select an initial example to be generalized. When there are not more examples, stop.

2. Construction of the more specific clause based on the restrictions language and the example selected in the last procedure. To this clause, we call bottom clause. To this step we call saturation.

3. Search for a more general clause than the bottom clause. These searches use the algorithm Branch-and-Bound. To this step, we call reduction.

4. Add the best clause to the theory, remove all redundant examples and return to step 1.

The Aleph uses three files to construct a theory:

- file.b: contains the background knowledge (intentional and extensional), the search and language restrictions and restrictions in the types and parameters;

- file.f: contains the positive examples (only facts without variables);

- file.n: contains the negative examples (only facts without variables). This file may not exist (Aleph can learn only by positive examples).

3.3 Mode Declarations

The mode declarations stored in the file.b describe the relations (predicates) between the objects and the type of data. That declarations allows to inform Aleph if the relation can be used in the head (modeh declarations) or in the body (modeb declarations) of the generated rules [7]. The declaration modes also describe the kind of arguments for each predicate, and have the follow format:

mode(call_numbers, Mode)

The call_numbers (also called recall), define the limit number of alternative instances for one predicate. A predicate instance it's a substitution of types for each variable or constant. The recall can be any positive number greater or equal to 1 or '*'. If it's known the limit of possible solutions for a particular instance, it's possible to define them by the recall. For example, if we want to declare the predicate parent_of(P,D) the recall should be 2, because the daughter D, has a maximum of two parents P. In the same way, if the predicate was grandparents(GP,GD) the recall should be 4, because the granddaughter GD has a maximum of four grandparents GP. The recall '*' is used when there are no limits for the number of solutions to one instance.

The Mode indicates the predicate format, and can be described has:

predicate(argument_type1, argument_type2, , argument_typen)

Example: for the learning relation uncle_of(U,N) with the background knowledge parent_of(P,D) and sister_of(S1,S2), the mode declarations could be:

- :- modeh(1,uncle_of(+person,+person))
- :- modeb(*,parent_of(-person,+person))
- :- modeb(*,parent_of(+person,-person))
- :- modeb(*,sister_of(+person,-person))

The declaration modeh indicate the predicate that will compose the head of the rules [7]. For this case, modeh inform us that the head of the rules should be uncle_of(U,N) where U and N are from the type person. The symbol '+' that appears before the type indicates us that the argument of the predicate is a variable. In this way, the head of the rules can be of the type uncle_of(U,N), and not, for example, uncle_of(john,ana). The symbol '-'indicates us it's an output variable. Instead of '-', if the symbol # appeared, indicates us that the argument could be a constant. The modeb declaration indicate that the generated rules can have, in the body of the rules, the predicate parent_of(P,D), where P and D are from the type person. The first modeb declaration in the example can be used to add parent_of in the body of the rules and add one or more parent(s) to a daughter (observe that call_numbers have the value '*'). Similarly, the second declaration modeb let the predicate parent_of be used in the body of the rules to find one or more daughters of a parent. At last, the third modeb declaration can be used to find one or more sister of a person.

3.4 Types

Types have to be specified for every argument of all predicates to be used in constructing a hypothesis [8]. To the Aleph, these types are names and these names means facts. For example, the description of objects for the type person could be:

- person(john)
- person(leihla)
- person(richard)
- ...

3.5 Determinations

Determination statements declare the predicated that can be used to construct a hypothesis [8]. This declaration take the follow format:

determination(Target_Pred/Arity_t, Body_Pred/Arity_b)

The first argument is the name and arity of the target predicate [8]. It's the predicate that will appear in the head of the induced rule. The second argument it's the name of the predicate that can appear in the body of

the rule. A possible determination for a relation called uncle_of(U,N) is:

determination(uncle_of/2, parent_of/2)

Typically, lots of declarations should be done for a target predicate. In case of non declared determinations, the Aleph don't construct any rule.

3.6 Positive and negative examples

The positive examples of the concept to be learned should be stored in the file with extension .f and the negative examples in the file with extension .n. For example, to learn the concept uncle_of(U,N), we could have the follow positive examples in the file with extension .f:

- uncle_of(Sam,Henry)
- uncle_of(Martha,Henry)
- ...

And the follow negative examples in the file with extension .n:

- uncle_of(Lucy,Charles)
- uncle_of(Lucy,Dominic)
- ...

3.7 Parameters

The Aleph let us define a variety of restrictions to be learned in the hypothesis space, such as a search of new values and available parameters in that space [7]. The predicate set allows the user to define a value of the parameter Parameter:

set(Parameter,Value)

We can also get the current value of a parameter:

setting(Parameter,Value)

And for last, the predicate noset, change the current value for its pattern value.

3.8 Other Characteristics

The Aleph has other important characteristics like:

- Instead of selecting one initial example to be generalized, it's possible to choose more than one. If we choose more than one initial example, it's created a bottom clause to each one of them. After the reduction step, the best of all reductions it's added to the theory;
- Let us construct the more specific clause, defining the place where the bottom clause it's constructed;
- The search clauses can be changed, using other strategies instead of using the Branch-and-Bound algorithm;
- It's possible to remove redundant examples to give a better perspective of the result clauses.

4 Wrappers

4.1 Adaptive patterns

The development of the graphical interface will be the primary goal of this work. Although it's also necessary to develop a wrapper. A wrapper, also known as adapter design pattern, is a type of software that is used to attach other software components [9]. In simple words, a wrapper encapsulates a single data source to make it usable in a more convenient way than the original unwrapped source. Wrappers can be used to present a simplified interface, encapsulating diverse sources so that they all present a common interface, adding functionalities to the data source, or exposing some of its internal interfaces [10]. In other words, the adapter design pattern is useful in situations where an already existing class provides some or all services you need but does not use the interface you need [11].

4.2 Functional model

All wrappers have the same basic logical mode [10]:

- The application operate in a language X;
- The application get responses in model Y;
- The wrapped data source is operated in a language Z;
- The wrapped data source responds with results expressed in model W.

The objective of the wrapper is convert the language X commands to language Z and the model W results to model Y. The differences between wrappers are on the models they support and in the sophistication of the functionality they provide.

4.3 Functionality levels and classes

There are four levels of wrappers [10]:

- Level 1: Queries in - Answers out;
- Level 2: Updates in - Yes/No out;

- Level 3: Subscription in - Notifications out;
- Level 4: Queries about - Capability lists out

The functionality of wrappers can be divided in five categories [10]. This subdivision hints at the internal architecture when the wrapper is constructed. The classes of functionality are:

- Generic functionality: refers to operations that take place because of the type of data source. For example, conversions of SQL3 queries to SQL queries against an Oracle DBMS will be done the same, regardless of the database schema;
- Source-specific functionality: refers to the operations that take place because of the data source. Assuming that the data source remained unchanged, these conversions would take place regardless of the program managing the data;
- Control structure: refers to the way which requests and responses are passed. For example, a synchronous data source may be wrapped by an asynchronous wrapper that buffers responses until requested by the caller;
- Error reporting: refers to errors reported back to the caller. These errors may be generated by the data source or by the recovery operation within the wrapper;
- Caller model: most wrappers assume that they are wrapping the source to be used by a particular kind of caller. Before the wrapper converts to the data source, the wrapper implementation can be reused;
- Calls to external services: refers to called services by wrappers to perform more complex services than the wrapper can perfom.

4.4 Example

Socket wrenches provide an example of the wrapper [12]. A socket wrench is a tool that uses separate, removable sockets to fit many different sizes. In this case, a socket attaches to a ratchet, providing that the size of the drive is the same. Typical drive sizes in the United States are 1/2" and 1/4". Obviously, a 1/2 drive ratchet will not fit into a 1/4 drive socket unless an adapter is used. A 1/2 to 1/4 adapter has a 1/2 female connection to fit on the 1/2 drive ratchet, and a 1/4 male connection to fit in the 1/4 drive

socket [12].

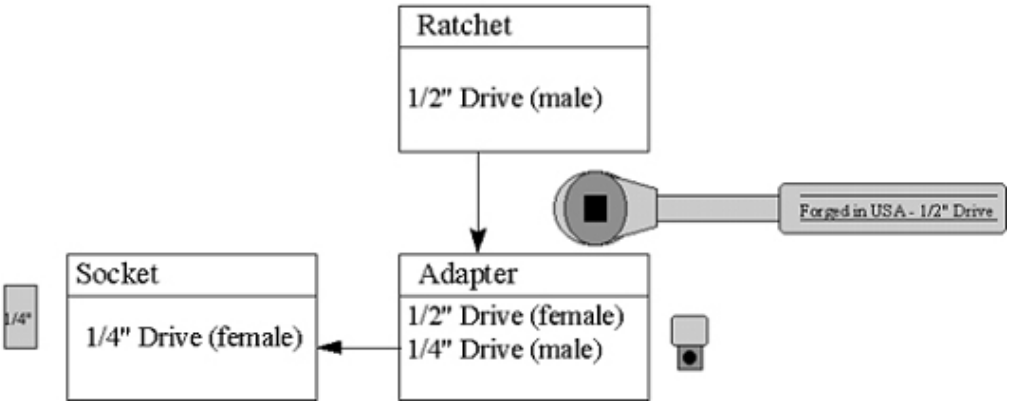

Figure 4: Wrapper example. [12]

5 Working Plan

As said in the Introduction the work consists in the development of a interface and a wrapper to make the use of the Aleph system easy. For that purpose we will design an interface with user interface techniques that facilitate the interaction between the user and the system. We will develop methods to translate data from different formats into a format suitable for Aleph. The interface will have facilities to assist the user in the development of the background knowledge required by Aleph. The Aleph's parameters will be automatically tuned using a wrapper. We will also provide the interface with procedures to make the presentation of the constructed models more appealing, translating them from logic to natural language (English).

The working plan gives us a perspective and orientation for the dissertation development. Starting in 18th February and finishing at 29th June, give a total of 18th weeks to work on the dissertation, excluding the Easters week. The predicted working plan for these weeks is:

- Weeks 1-2: Designing the interface architecture;
- Weeks 3-7: Implementation of the interface;
- Weeks 8-10: Implementation of the Aleph wrapper;
- Weeks 11-12: Deployment and test of the interface using test cases;
- Weeks 13-16: Writing of the thesis text.

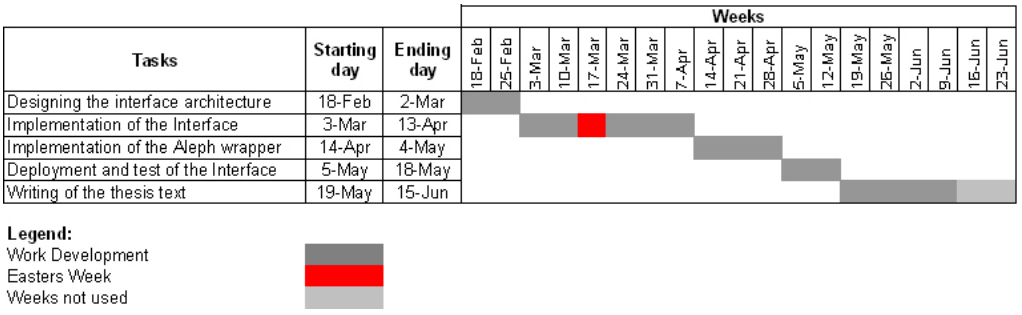

Figure 5: Gantt diagram demonstrating the working plan.

As we can see in the working plan, there are two weeks not used. These weeks were intentionally not used in order to use it in case of delays during the development work.

6 Conclusions

This report gave introductory concepts for ILP systems, in particular the Aleph system and wrappers. Now that the background knowledge for this thesis was debriefed, the dissertation work may be developed with the primary goals mentioned before. It was established a working plan in order to follow the necessary steps in the predicted dates which will lead to a successful project. The dissertation will be done at Faculty of Informatics, Masaryk University, Brno, Czech Republic in collaboration with the Professors Rui Camacho and Lubomir Popelinsky.

References

[1] Morgan Kaufman, Data Mining Practical Machine Learning Tools and Techniques, 2nd Edition, 2005

[2] Data Mining, Wikipedia (http://en.wikipedia.org/wiki/Data_mining)

[3] Jiawei Han, Micheline Kamber, Morgan Kaufman, Data Mining Concepts and Techniques, 2000

[4] Nils J. Nilsson, Introduction to Machine Learning, 1996

[5] Saso Dzeroski, Nada Lavrac, Inductive Logic Programming, Techniques and Applications, 1994

[6] Saso Dzeroski, Nada Lavrac, Relational Data Mining, Chapter 14 - Relational Data Mining Applications: An Overview

[7] Mariza Ferro, Aquisição de conhecimento de conjuntos de exemplos no formato atributo valor utilizando aprendizado de mquina relacional, Julho 2004 (http://www.teses.usp.br/teses/disponiveis/55/55134/tde-16112004-095938/)

[8] A. Srinivasan, The Aleph Manual (http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph_toc.html)

[9] Computer Science Department, (www.exciton.cs.rice.edu/JAvaResources/DesignPatterns/adapter.htm)

[10] David Wells, Object Services and Consulting, Inc., 1996 (http://www.objs.com/survey/wrap.htm)

[11] Adapter Pattern, Wikipedia (http://en.wikipedia.org/wiki/Wrapper_pattern)

[12] Vince Huston, OO design, Java, C++ (http://www.vincehuston.org/dp/adapter.html)