Dottorato di Ricerca in Informatica
Università di Bologna, Padova

Knowledge Management in Intelligent Tutoring Systems

Simone Riccucci

December 2007

Coordinatore:
Prof. Özalp Babaoğlu

Relatore:
Prof. Antonella Carbonaro

To Cristina

Abstract

In the last years, Intelligent Tutoring Systems have been a very successful way for improving learning experience. Many issues must be addressed until this technology can be defined mature. One of the main problems within the Intelligent Tutoring Systems is the process of contents authoring: knowledge acquisition and manipulation processes are difficult tasks because they require a specialised skills on computer programming and knowledge engineering. In this thesis we discuss a general framework for knowledge management in an Intelligent Tutoring System and propose a mechanism based on first order data mining to partially automate the process of knowledge acquisition that have to be used in the ITS during the tutoring process. Such a mechanism can be applied in Constraint Based Tutor and in the Pseudo-Cognitive Tutor.

We design and implement a part of the proposed architecture, mainly the module of knowledge acquisition from examples based on first order data mining. We then show that the algorithm can be applied at least two different domains: first order algebra equation and some topics of C programming language. Finally we discuss the limitation of current approach and the possible improvements of the whole framework.

Acknowledgements

PhD thesis work has been very challenging and hard but at the same time very interesting and exciting. The experience of Phd has certainly enriched me both from a scientific and human point of view. I would like here to thank all the people who directly or indirectly have allowed me to reach this aspired goal.

First of all I wish to thank Prof. Giorgio Casadei and prof. Antonella Carbonaro for their indispensable advice and essential support that they gave me during the time of PhD thesis writing.

Special thanks goes to prof. Simone Martini and prof. Andrea Sperduti for their valuable advice on the drafting of the thesis.

A warm thanks goes to the Department of Information Sciences at the University of Bologna that allowed me through facilities and funds to continue my work. I also thank all my colleagues PhD students with whom I have somehow closely "lived" for all the years of doctorate.

Many thanks go to Prof. Kinshuk and Prof. Stefano Cerri that have kindly agreed to review the PhD document and in particular Professor. Cerri for giving valuable advice on its revision.

I wish to thank my parents for their constant support and encouragement in all my professional endeavors. A special thanks goes to my brother-in-law Paolo that has helped in me to review the english language of my thesis.

Finally, a special thanks goes to my wife Cristina whose moral support during these years was more than necessary.

Contents

List of Tables

List of Figures

Introduction

In the last two decades there has been an incrementing demand for instructional needs both in quality and quantity. The use of computer to support the educational process has gained increased acceptance due to the ever more stringent needs of both private and public learning institutions. For this purpose, many systems supporting this kind of activities have been developed to streamline, to various degrees, the learning and assessment process.

Initially systems were very simple and they did not take into account any kind of information that could improve the learning process. Many of the applications used today are still simple although they have improved their graphical aspect and they begin to adopt standard for contents reuse and exchange such as SCORM.

Essentially the interaction between user and system was reduced to a sequential presentation of contents as in a text book and to a presentation of questions at the end of every module of instruction.

The lessons learned from years of development and experience, the advance of technology, and the development of authoring tools for systems supporting educational process has resulted in a sophisticated, computer based tutoring and assessment system. However, there is still a lot of room for further development.

A primary aim of tutoring and assessment task is to provide the necessary information to improve future educational experiences because it provides feedback on whether the course and learning objectives have been achieved to satisfactory level. Yet, it is important that the interaction is accurate and relevant to effectively make informed decisions about the learning process. Moreover, formative assessment can also be used to help bridge the gap between assessment and learning. This may be achieved particularly where assessment strategies are combined with useful feedback, and integrated within the learning process. The answers to the described objectives are enhanced if we could integrate intelligent adaptive techniques both in the material presentation and assessment process; accurate and fitted assessment data may improve both the curriculum and the student ability level.

The developing of this type of applications leads to several difficulties at various levels:

- e-learning is a relatively new field of research and there are not commonly accepted standards for the information encoding of these systems (even if much work has been done and is being made in this direction);
- realizing systems that are aware of both the domain in which they operate and the learner profile it is an extremely difficult task;
- there is not a common architecture that accurately describes the structure of the various parts the system is made of and mechanisms for their interaction, in such a way they are easy reusable;
- these systems are hard to setup and tune because requires the participation of many scientific discipline experts.

Some of the current ideas for intelligent tutoring system development are discussed in the remainder of this work. In particular we try to address the aspects of domain awareness and authoring process with particular attention for adding semantic knowledge to the system content.

In the conducted research we have analyzed some kind of intelligent tutoring systems and we have chosen the most significant for our purposes. We first propose a general framework in which we can place our activities and then we plan to develop some of the parts of the whole architecture.

We have focused our attention on knowledge acquisition task that is claimed to be the bottleneck for intelligent tutoring systems realization. We chose constraint in the form of conditional statement as the base for knowledge encoding and we propose a framework based on ontology, information extraction and inductive logic programming to accomplish the complicated task of knowledge acquisition. The knowledge is used by the system for guiding the learning process of the user.

The aim of this work is to ease the creation of an "intelligent system" meaning that the system can use some kind of information to enhance the learning process. In a successive step the system should be made "adaptive" in the sense that the system has to learn from experience for improving its performance. This topic is not covered by this thesis, but a future work in this direction is desirable.

Outline

Chapter 1 In Chapter 1 we give an overview of what an Intelligent Tutoring System is, show its main components that are student module, pedagogical module, domain knowledge, expert model and communication module. We then review the state-of-the-art ITSs which are Cognitive Tutors and Constraint-based Tutor and describe some instances of this type of tutor.

Chapter 2 In Chapter 2 we make some considerations about the current systems used in literature. We underline the main problems and propose a general architecture to overcome the issues posed in such a systems. We propose a system made up of various modules each of which should solve a problem. The aim is to separate the problems such that they can be resolved in various domain of research. In the following we concentrate on the knowledge acquisition module.

Chapter 3 In Chapter 3 we present the general architecture of knowledge acquisition module that is composed by various components. They process the solution examples given by the teacher and find the rules that holds in the examples. The found rules are then presented to the teacher for the validation phase and saved in the system as domain knowledge to be used by the *domain module* and *student module* of ITS. The ideas illustrated in this chapter was published in [40, 41, 42]

Chapter 4 In Chapter 4 we describe in detail an instance of the architecture described above based on first order data mining algorithms that is able to find disjunctive first order association rules starting from examples given by the user. We then analize some simple run to show the usefulness of the algorithm in authoring a knowledge base in ITSs. The ideas illustrated in this chapter was published in [43].

Chapter 5 In Chapter 5 we describe the process of building the knowledge base for a tutor on first order algebra equation and a simple C tutor. Furthermore we analize the output of the system using two different data-miners for the generation of rules.

Chapter 1

Intelligent Tutoring Systems

1.1 Introduction

Early systems that were developed for teaching purpose were known variously as *Computer Aided Instruction* or *Computer Aided Learning* systems. These traditional systems were developed to provide users with knowledge in a particular area and then assess the user retained knowledge by posing questions, usually multiple choice, to direct the course of study. The structure of such a system is shown in figure 1.1

Examinees are presented with information after which a series of question are asked. If the user answers correctly the next phase of instruction is entered. There may be a summary set of questions at the end of the instruction to gauge how well the student has performed.

If an examinee answers the questions incorrectly the instructional material is presented again, perhaps in a slightly different format. If, after the material has been repeated, the examinee again answers incorrectly then remedial instruction may be presented. In this kind of systems, the instruction was not individualized to the learner's needs. In general the decisions about how to move a student through the material were script-like, such as "if question 21 is answered correctly, proceed to material 54; otherwise go to material 33".

Such systems can appear to imitate intelligence by being able to adapt to student misconceptions. However, this appearance is a result of the system designer anticipating all possible errors that the student may make. These are built into the system at design time and encoded within the branching structure. If the designer of the system has not anticipated an incorrect interpretation of the instructional material then the system will not be able to provide feedback to help the examinee resolve the misunderstanding. These systems are incapable of dynamically generating a response to a particular situation as a human tutor would be able to do.

So, while CAI may be somewhat effective in helping learners, they do not provide the same kind of individualized attention that a student would receive from a human tutor. For a com-

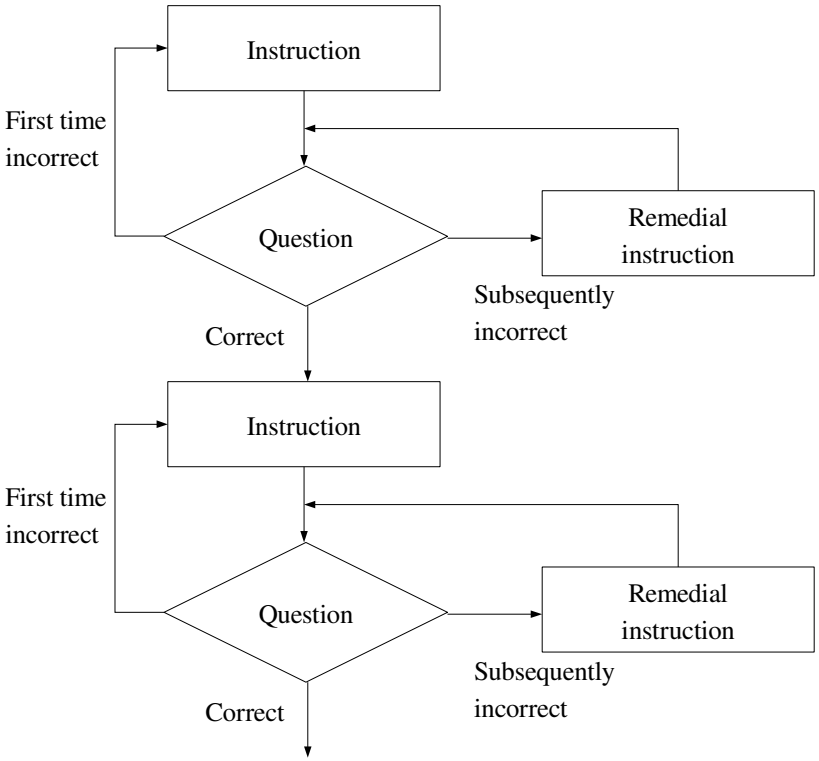

Figure 1.1: Routing within a Computer Aided Instruction System

puter based educational system to provide such attention, it must implement more advanced algorithms to parse the knowledge domain and related learner feedback.

During last few years there have been many systems developed for this purpose each having its own characteristics. Until now the majority of such ITSs is specialized in a specific knowledge domain. In literature we can find many examples of specialized ITS operating in various knowledge domain.

Over the years, various tutoring systems have been built to teach programming (e.g., PROUST [20], MENO-II [46], and ELM-PE [56].) They analyze the user's solutions to exercises and provide feedback to identified misconceptions or missing skills based on the analysis. Other approaches have also appeared in intelligent tutoring paradigm. Kumar's model-based tutor asks students to predict C++ programs' output and identify semantic and run-time errors [8]. It provides explanations of program execution line by line to help students understand code behavior. Adaptive navigation based on student modeling is used in a web-based system called ELM-ART II [57] to provide individualized annotated hyperlinks and curriculum sequencing.

Expert critiquing systems provide useful feedback to users' work in many domains. Such feedback includes alerts to problematic situations, relevant information to the task at hand, directions for improvement, and prompts for reflection [44]. One of these systems is "Java Critiquer" that uses an incremental authoring approach to amortize the high development cost [39].

Other systems like "CTutor" [47], "Prolog Tutor" [18] and "Java Intelligent Tutoring System" [54] adopts advanced code analysis to see what is the intention of student and give feedback based on this analysis .

There are also tutoring system based on natural language like CIRCSIM-Tutor [2] that is a language-based intelligent tutoring system for first-year medical students to learn about the reflex control of blood pressure. Students solve small problems and are tutored by dialog with the computer.

Despite of the great variety of ITS in the literature we chose to focus our attention on tutoring system that define a generic architecture. It is possible, in principle, to inject knowledge base of every kind in this type of system so that the process of learning can start in whatever domain of knowledge. This choice will enable us to develop a system for knowledge acquisition that tries to be independent from the specific domain in which an ITS can be developed.

In the next sections, after a brief introduction to the ITS, we show the two major systems that are state-of-the-art for these generalist ITS architectures: Constraint Based Tutor and Cognitive Tutor.

1.2 Intelligent Tutoring Systems: overview

Recognizing the deficiencies of traditional Computer Aided Instruction systems Intelligent Tutoring Systems were subsequently developed which attempted to adapt the speed and level of presentation to that required by a student.

Typically this kind of systems can be seen as a number of independent components which can communicate between them:

- the *student module* forms a framework for identifying a student's current state of understanding of the subject domain. The knowledge that describes the student's current state of mind is stored in a student model. In order to make any learning environment adaptable to individual learners, it is essential to implement a student model within the system. The student module should permit the system to store relevant knowledge about the student and to use this stored knowledge to adapt the instructional content of the system to the student's needs;

- the *pedagogical module* contains the knowledge of how to teach, that is, a teaching or tutoring strategy. It orchestrates the whole tutoring process and deals with issues like which topic to present, when to present a new topic, when to present a problem, when to review, and when to offer remedial help;

- the *domain knowledge* module contains the knowledge of what to teach. It represents an area of syllabus and usually requires knowledge engineering in its construction. Domain knowledge is usually represented as skills, concepts, procedures and problems of the subject domain under study;

- the *expert model* is strictly related to domain knowledge. The expert model uses the domain knowledge to advise other parts of the system. It may indicate the relative difficulty of curriculum sections or problems, such that the pedagogical module can select the next task. Furthermore, this module aims to provide expert like solutions to problems in the domain to be taught;

- the *communication module* controls interactions with the learner, including the dialogue and the screen layouts.

The interaction scheme between the five modules is shown in figure 1.2. Knowledge representation and tutoring methodologies are areas suitable for the application of intelligence.

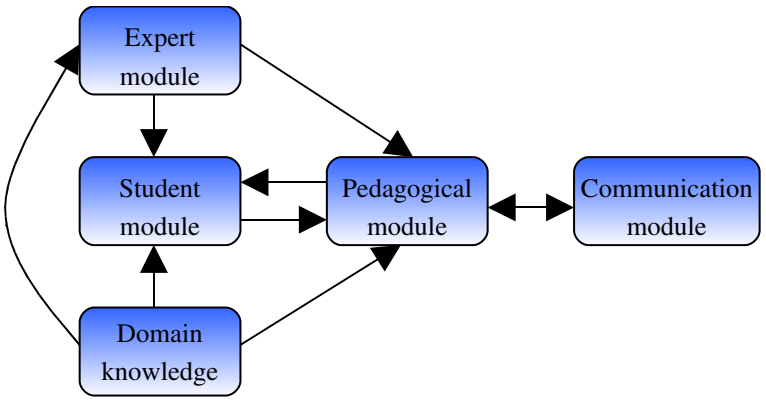

Figure 1.2: Interaction of component in an Intelligent Tutoring System

1.2.1 Student Model

The student diagnosis module has the function of capturing the level of a student's understanding of the domain knowledge. It keeps track of information that is specific to each individual student, such as his mastery or competence of the material being taught, and his misconceptions. In effect, it stores the computer tutor's beliefs about the student. These information are used by the pedagogical module to tailor its teaching to the individual needs of the student.

This module concerns what is commonly known as *student modelling*, that is how to store the current student knowledge. This information is very often related to the domain knowledge as its subset.

Student models can be classified according to the functions they can perform. In [45] are described six main functions of student models:

- **Corrective:** Feedback intended at repairing a misunderstanding of the student. In this case, the model must identify a difference between the student's understanding and the correct knowledge, and provide this information to other parts of the system.

- **Elaborative:** Extending the knowledge of the student. In this case, the model should identify areas where the student can be introduced to new material or a refinement of her current understanding.

- **Strategic:** Changing the approach to teaching at a higher level than local tactics. This requires the student model to provide more general information about the student, such as her success rate with the current teaching strategy as opposed to a previous teaching strategy.

- **Diagnostic:** Analysis of the state of the student. In some sense, all aspects of student modelling are diagnostic. What is meant here is the explicit use of the student model to refine information about the student in order to make a decision. If, for example, the tutor wishes to introduce a new topic, but the student model is unable to indicate whether the current level of understanding of the student is adequate, the model can be requested to generate diagnostic examples which can be presented to the student.

- **Predictive:** Using the model to anticipate the effect of an action upon the student. This requires the student model to act as a 'simulator' to simulate the behaviour of the student, given a particular action.

- **Evaluative:** Providing an assessment of the level of achievement of the student. This requires the system to make some aggregation across the information that it has.

Student models can also be classified by their modes of interpretation: process or state models. Process models are capable of simulating the process by which the learner solves a problem and can therefore perform the predictive function of student modelling. They are also called executable or runnable models. A student model is executable if its present state can be utilised by a certain interpreter to simulate the behaviour of the modelled student when he is solving a problem. *Cognitive tutors* [22], which will be presented in next chapter, use this kind of student modelling. On the other hand, state models do not have the capability of simulating and contain only state information. Examples are *Constraint-based tutors* [33].

1.2.2 Domain knowledge and expert module

A crucial aspect in the development of an ITS is how related knowledge is represented and how reasoning for problem solving is accomplished. The domain model contains a representation of the information to be taught. It provides input into the expert module, and ultimately is used to produce detailed feedback, guide problem selection/generation, and as a basis for the student model.

The domain model may take many forms, depending on the knowledge representation used, the domain it represents, and the granularity of the information being presented. For example domain knowledge can be stored at the page and topic level, and provides basic information about the content of the page, which aids in problem selection and course sequencing. In Cognitive tutors, the domain model consists of low-level production rules that completely describe the expected student behaviour down to atomic thought components. Simulation-based systems [34], use the domain model to describe how each component of the simulation should behave (i.e. what actions are possible with this object, and what the consequences of each action should

be), and how components are interrelated. Constraint-based systems describe the possible valid states that an answer may occupy.

The expert module uses the domain knowledge to advise other parts of the system. It may indicate the relative difficulty of curriculum sections or problems, such that the pedagogical module can select the next task. In Cognitive tutors it identifies whether or not the student's current solution is on track and, if not, assess what has gone wrong. It may also be able to run the domain model to solve the problem from a given state. In constraint-based systems it evaluates the student solution against the constraints to determine what concepts have been misunderstood.

1.2.3 Pedagogical module

The pedagogical module uses information from the student model to determine what aspects of the domain knowledge should be presented to the learner. This information, for example, may be new material, a review of previous topics, or feedback on the current topic. One pedagogical concern for an ITS is the selection of a meta-strategy for teaching the domain. Once the meta-strategy is selected, low level issues, such as the exact example to use, must be decided.

The tutor must decide the content of the material to be presented to the student. This involves decisions about the topic, the problem, and the feedback.

To select a topic to present, the tutor must examine the student model to determine the topics on which the student needs to focus. Many possibilities exist for the most appropriate topic on which a student should work. For example, if the meta-strategy indicates that review is in order, the tutor will select a topic the student has already learned. On the other hand, if new information is to be presented, the tutor will choose a topic that the student does not yet know.

Once the topic has been selected, a problem must be generated for the student to solve. The grain size of the problem is determined by the domain knowledge representation. For example the student can be asked to deduce some facts from a simulation of the world concerning the domain, or the student can be given a simple problem, such as adding two fractions. Whatever the granularity of the problem generated, it is important that the difficulty be appropriate for the student's level of ability, which can be determined from the student model.

Most tutors work smoothly as long as students get everything right. Problems arise when the student has difficulties and needs help from the tutor. In these situations, the tutor must determine the kind of feedback to provide. The issue of how much help to provide the student is also a very complex issue as too little feedback can lead to frustration and floundering while too much feedback can interfere with learning.

Once the system decides how much feedback to give, it must determine the content of the advice. The feedback should contain enough information so that the student can proceed to the next step in solving the problem. Furthermore, the advice given to the learner should be appropriate

for her ability level. For example the more proficient the student is at a particular skill, the more subtle the hint is. On the other hand, a student with low proficiency in a skill would be presented with a more obvious hint.

High level strategy selection in ITSs has proven a formidable problem. Most ITSs do not explicitly identify the strategies they are using for teaching and implicitly implement an hardcoded strategy. A better method is to use the student model to select an appropriate strategy from those maintained by the system. Ideally a student's model could track the instructional strategies that are most effective for teaching him.

1.3 Cognitive tutor

Cognitive tutors are based on the ACT-R theory of mind [1] . The central principle of this theory is that the processes of thought can be modelled using declarative and procedural knowledge. Declarative knowledge corresponds to things we are aware we know and can usually describe to others. Examples of declarative knowledge include "Luigi Einaudi was the first president of the Italian republic" and "The halting problem is not computable". Procedural knowledge is knowledge which we show in our behaviour but which we are not conscious of. For example, usually no one can describe the rules by which we speak a language but we can do it. In ACT-R declarative knowledge is represented in structures called *chunks* whereas procedural knowledge is represented in *productions*. Thus chunks and productions are the basic building blocks of an ACT-R model.

Tutoring is achieved using a method known as *model tracing*. As the student works at the problem, the system traces her progress along valid paths in the model. If she makes a recognisable off-path action she is given an error message indicating what she has done wrong, or perhaps an indication of what she should do. If the action is identified as being off-path but cannot be recognised, she may only be told that it is incorrect, but not why.

Knowledge tracing is used to monitor the knowledge that students have acquired from problems. A Bayesian procedure is used to estimate the probability that a production rule has been learned after each attempt. This assessment information is used to individualize problem selection and optimally route students through the curriculum.

1.3.1 PAT: Algebra Tutor

PUMP (Pittsburgh Urban Mathematics Project) Algebra Tutor (PAT) [23] was developed for use in High-School setting following the ACT theory. The main purpose of the system is teaching to apply mathematics learned at school to real world problem.

Students work through PAT problem situations by reading its textual description and a number of questions about it. They investigate the situation by representing it in tables, graphs, and symbols and using these representations to answer the questions. The major focus of the tutor is helping students to understand and use multiple representations of information.

Students are presented with a screen divided in four windows each containing a tool for solving the problem (in addition to the window showing the current text of problem). While students work on a problem, tutor traces their activities and if something goes wrong (student makes error), the system shows the relative feedback in various form, depending on the type of error.

An experiment was conducted on 470 students using this system and the experimental classes outperformed students in comparison classes by 15% on standardized tests and 100% on tests targeting the PUMP objectives (real life problems).

1.3.2 LISP Tutor

The LISP tutor [9] was an early attempt at a Cognitive tutor. The student is given a description of a small program to encode in LISP, which she then writes with the system's help.

In the LISP Tutor the expert model was created as a series of correct production rules for creating LISP programs and a learner model was built as a subset of these correct production rules along with common incorrect ones. The tutor acts as a problem solving guide but never states the productions to be learned.

Students interact with a language editor that shows the structure of program to be coded. As the user builds her solution, the system inserts tags that describe the general form of the program, for example (user input in bold):

```
(defun fact (n)
 (cond ((equalp) <ACTION>)
  <RECURSIVE-CASE>))
```

In the above case the student was asked to code the program that compute the factorial of a given number. At this point the system interrupts the student and asks for the use of ZEROP function, that tests a number against zero, instead of the general EQUALP. Then the tutor interacts with the student in similar manner until the final solution is reached.

Note that, even if the use of EQUALP is valid, this would lead to a solution that is off the path specified in the LISP tutor, because the authors have deemed it undesirable. This is a design decision, rather than a characteristic of the modelling method.

The LISP tutor performs very well. In an initial mini-course at CMU, students using it solved a series of exercises in 30% less time than those in a standard LISP environment, and performed one standard deviation better on their final test. However, later evaluations failed to conclusively

prove that the style of teaching used (notably comprehensive, immediate help) improved performance *per se*.

Anderson et al. concluded that the main reason for improved performance in post tests previously was probably because the LISP tutor enabled students to cover more exercises in the same amount of time, which subsequently gave them an advantage. However, this is, in itself, considered a worthwhile outcome, since enabling students to achieve their learning in less time gives them more time to learn additional material, or to do other things.

1.4 Constraint-based tutor

This kind of ITS is based on Ohlsson's theory of learning from performance errors. Constraint Based Modelling (CBM) focuses on faulty knowledge, realizing that it is not sufficient to describe what the student knows correctly. The basic assumption is that diagnostic information is not hidden in the sequence of student's actions, but in the problem state the student arrived at. This assumption is supported by the fact that there can be no correct solution of a problem that traverses a problem state, which violates fundamental ideas, or concepts of the domain.

The constraint-based model proposed by Ohlsson represents both domain and student knowledge in the form of constraints, where the constraints represent the basic principles underlying the domain. A constraint is characterised by a *relevance* clause, and a *satisfaction* clause. The relevance clause is a condition that must be true before the constraint is relevant to the current solution. Once the relevance clause has been met, the satisfaction clause must be true for the solution to be correct.

For example we can consider a student learning to code with a programming language. When she has to manipulate array inside an iterative statement one of such rules can be that the index used in the array must not exceed the array dimension. A constraint for the situation might be:

```
IF array dimension is N
AND array index is I
THEN the guard is I < N
```

1.4.1 SQL-Tutor

SQL-Tutor is a knowledge-based teaching system which supports students learning SQL. The intention was to provide an easy-to-use system that will adapt to the needs and learning abilities of individual students. The tailoring of instruction is done in two ways: by adapting the level of complexity of problems and by generating informative feedback messages.

It consists of definitions of several databases, and a set of problems and the ideal solutions to them. The student are presented a question asking to write an SQL statement, a description of

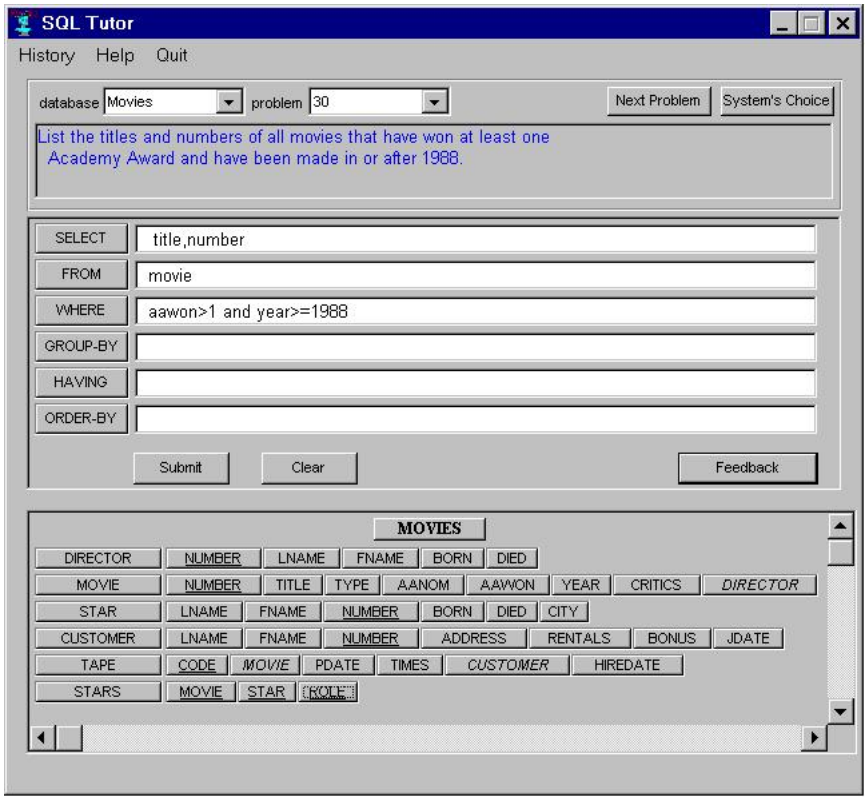

Figure 1.3: GUI of SQL-Tutor

the database schema and the structure of the query (currently only SELECT query are addressed) consisting of all clause SELECT, FROM, WHERE, ORDER BY, GROUP BY, HAVING. The graphic interface can be seen in figure 1.3 (from the author site). When the student submits the solution, it is passed to the system that corrects errors, updates the student model and selects the next action (i.e. feedback or another question). In the early version the system had several limitations:

- the domain admits more than one solution to a given problem but the system was unable to recognize a different solution. This translates in misleading feedback that can confuse the student. This problem has been resolved changing the representation of constraints reducing it to simple pattern matching and developing a problem solving procedure able to produce a piece of solution from a constraint [30, 31]. The partial solution replaces a piece of the student solution that violates the constraint and the procedure is iterated until no more errors exist.

- system had a limited problem set. Authoring a set of questions that cover all the defined constraint (over 500) is a very difficult task. In [31] have been developed a method, based on problem solving capability described in the previous point that generates a set of problem

starting from the constraint set defined in the system. The author has to manually transform the generated solutions into natural language for presentation to the student.

Authoring a constraint based tutor is a task reserved to ITS engineers or expert with a necessary knowledge to represent constraints. In [32] a system called WETAS was developed to help the process of system authoring. In [51] WETAS was augmented with the capabilities of extracting constraints from ontology.

Chapter 2

An enhanced architecture for teaching

2.1 Motivation

Many intelligent tutoring systems have been proposed, some of which have been developed and experimented (also on a wide scale like PAT tutor), on many types of knowledge domains mainly in scientific ones. In many of the experiences with these systems has been found a benefit for customers that have used them. It is therefore well proved that the use of this kind of systems is, in principle, effective and improves the process of learning in numerous varieties of domains.

The research conducted in the field of ITS is relatively young and there are several problems inherent to the development of this kind of software. In the following we will make some basic hypotheses to reason about and we try to define a system architecture that can solve some of the issues:

- There exist a certain amount of documents that can be retrieved in an easy and fast way and they come from trusted sources. These documents contain non structured information, which is data that cannot be used directly from the system for tutoring purpose (Learning Object, HTML documents, pdf documents, etc...). These documents deal with a specific domain of knowledge;
- There is an expert that is the depositary of the structured domain knowledge of interest and another expert that is the depositary of the instruction strategies to be used for teaching the domain (often these two figures are the same person);
- There is a system capable of transferring the structured knowledge from the expert to the computer and it can deduce the same knowledge, or part of it, from the documents that contain non structured information;
- The system delivers the acquired knowledge following a predefined pedagogical plan and improve its performances by means of users feedbacks and interactions.

Having in mind these considerations the main problems for the creation of an ITS are:

- Defining the representation of the semantic and/or syntactic content of documents that contain the non structured knowledge. This kind of representation should be used from the system in order to guide the learning process during the didactic material delivering. The extraction of this information must be carried out in automatic or semi-automatic way;
- Defining and realizing authoring tools that through the interaction with the depositary of the structured knowledge (the domain expert) can generate, in an easy and fast way, the necessary information needed to the system in order to both deliver the contents and guide the problem solving learning for the subject domain;
- Defining the system structure in such way to be highly modular and in such way that it can improve its performance through the several interactions with the users, that is it should learn from experience.

In the following section we propose a general architecture that take into account the previous considerations and that wants to be a contribution to the maturation of these systems with a look to the future developments in several of the research fields regarding every module of the system. In particular we pose the attention on the generality of the architecture that, in principle, could be used in order to construct ITS in whichever domain of knowledge. In the next chapter we will analyze the part of the system that is able to acquire the domain knowledge by means of interaction with the domain expert. In ideal conditions the domain expert will not need for additional competences, beyond to those possessed on the domain, in order to construct an instrument for supporting the learning process.

2.2 Architecture proposal: an overview

The proposed architecture is shown in figure 2.1. We suppose a scenario where the domain expert is a teacher that has access to this hypothetical system and to a number of electronic documents that she wants to use for creation of didactic material. The final objective will be to enable the teacher to easily author the system without any specific skills on knowledge engineering. The fundamental idea behind this architecture is to automate as much as possible the phase of domain knowledge acquisition and subsequently refinements to both the teaching strategies and domain knowledge. The human intervention is required in the case the system is not able to accomplish one of the task involved in knowledge acquisition. We chose the constraint-based model as the core structure of the system. The domain knowledge is composed by a number of component:

- The set of documents containing unstructured domain information;

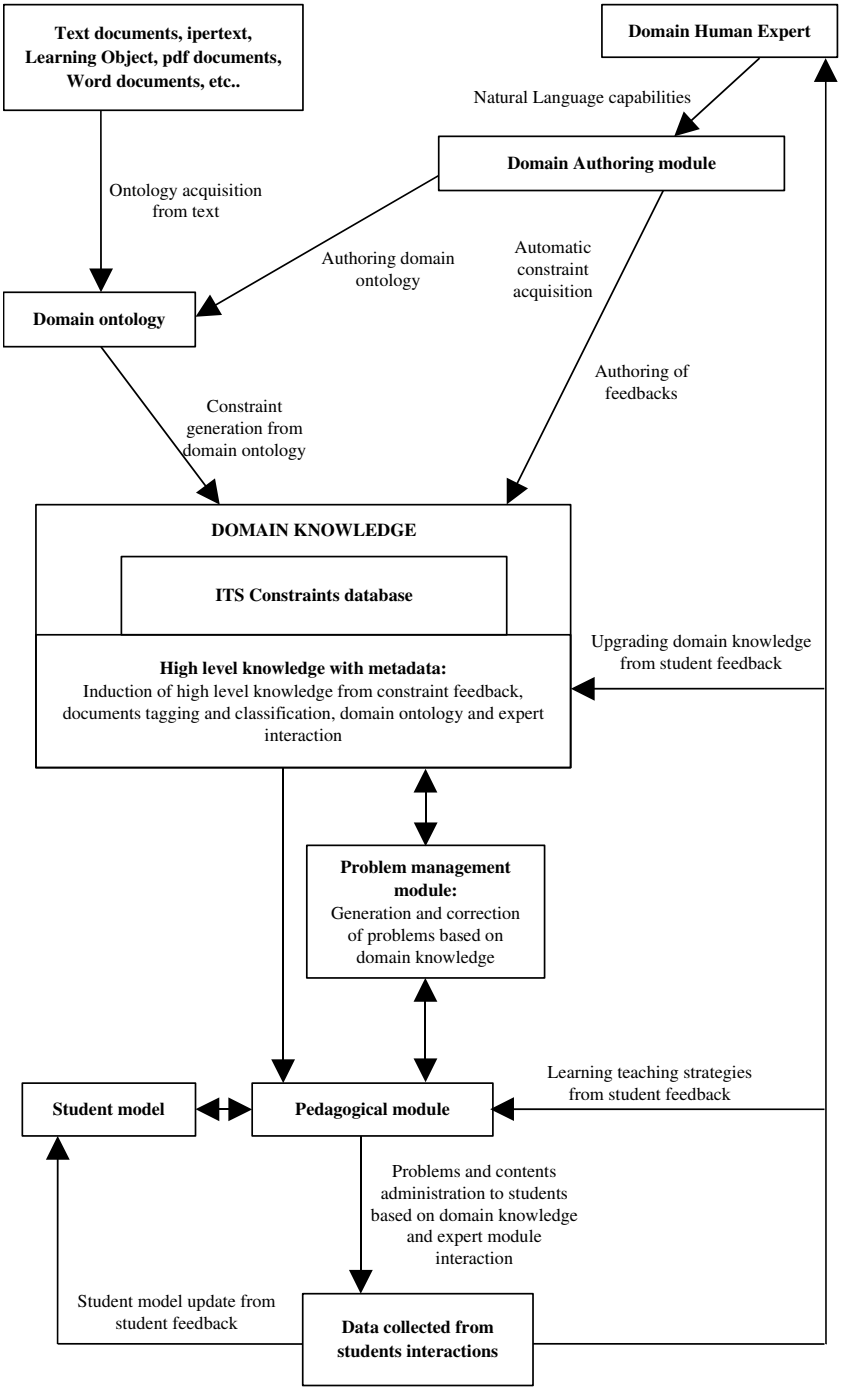

Figure 2.1: Architecture of an advanced Intelligent Tutoring System: proposal

- The ontology describing the concepts and relations between them;
- The set of constraints that must be satisfied by a solution state in order to be correct;
- An high level knowledge that groups the constraints in a more coarse-grained fashion.

Ideally the domain expert has to do some tasks that should not require additional competence apart the domain one. These actions are:

- Validate the content of documents, that is the expert has to instruct the system on the validity of the documents in the knowledge base;
- Provide a set of problems along with the related set of solutions. We suppose that solutions are supplied in form of text.

The system has a module that can create an ontology starting from the documents that deal with a determined argument. A number of systems have been proposed for semi-automatic ontology acquisition from text sources such as ASIUM [16], TextToOnto [27], Ontolearn [35] and OntoLT [11] even if there much work to do in order to obtain a system suitable for practical use. A possible integration of these systems should be reasonable.

In a more realistic scenario domain knowledge can be acquired by interacting with human expert that inserts her knowledge by means of an authoring tool capable of both graphic editing and natural language dialog. Both ontology and constraint base of the system can be acquired with this modality.

Once domain knowledge is acquired in the form of constraints, an induction of the structure of knowledge can be done based on ontology and interaction with human expert. In [29] Martin et al. use machine learning to induce the knowledge structure from feedback messages associated with constraints but the mechanisms should be improved considering also the domain ontology. This type of knowledge can be updated, hopefully improving it, by means of learning algorithms that based on student feedback can change or add some kind of relation (like precedence relation between topics).

The expert module of the system uses the set of constraint to check if a partial or total solution, given to certain problem, is in a correct state. An example of this methodology is given in [31] where an algorithm for problem solving and generation starting from constraint is presented.

The pedagogical module selects the most suitable action for current student based on student model and domain knowledge. A number of techniques were used to deal with this problem. An overview of student modelling and related issues can be found in [19] and [45]. The pedagogical module can be seen has a black box that takes in input information on student and domain and select the next action based on these information. The policy can be improved applying machine learning algorithms in order to refine teaching strategies.

In the following we will focus on the domain knowledge authoring system component, in particular on the acquisition of constraint base from domain ontology and examples given by the expert.

Chapter 3

Domain knowledge acquisition system

From the proposed architecture we now relax some assumption in order to realize two sub modules:

- the tutoring module that contains the knowledge base, a set of problem on the domain to be taught and the a problem solver/solution checker;
- the knowledge acquisition module that is made of various component that starting from a domain ontology and a set of examples learns new constraints.

The first relaxation is that we do not consider the documents in the knowledge base of the system. We suppose that the ontology is generated manually or automatically from another component. We do not consider the high level knowledge that will be useful in a successive phase of development when also the documents will be considered in the knowledge base. For this reason we simplify the pedagogical module that, in this context, selects the material based on very simple policies. Furthermore the refinement of both the teaching strategies and high level domain knowledge are not taken into account. In figure 3.1 the overall schema of tutoring module is shown. We suppose a scenario where the student is asked to answer a question requiring a solution in a form of text. The system supports the student problem solving by providing feedback that relates to student errors. In order to accomplish this task the system needs for a knowledge base which enables the capability of information extraction from solution (e.g. recognizing of concepts and relations between them) and error detecting on the provided answers.

3.1 The domain knowledge

Ontology is substantially a formal specification of a set of concepts surrounding a domain and a set of relationships that hold between the concepts. The role of ontology in domain knowledge ac-

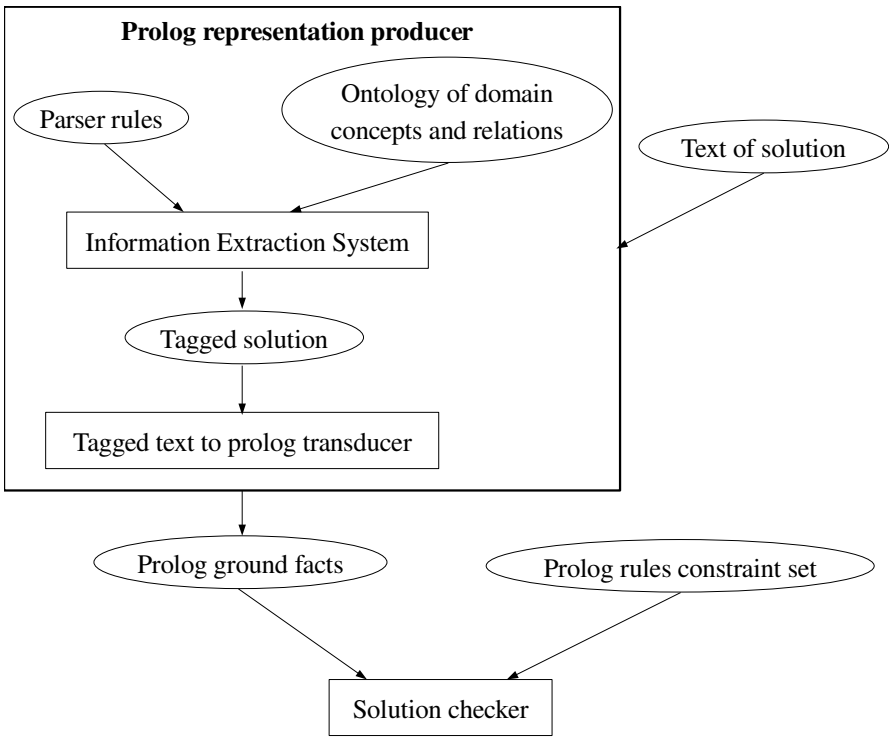

Figure 3.1: Schema of tutoring module

quisition and management has been demonstrated very important. In [52] and [53] a mechanism for automatic constraint acquisition based on domain ontology was proposed.

The Constraint Acquisition System (CAS) proposed by Suraweera et al. [53] (a tool supporting the authoring process of constraint-based tutor), allows the expert to create the constraint base, following these steps:

- The expert creates the domain ontology specifying concepts and relations. During this phase system engages a dialog with the expert for ontology validation;
- The expert decides the structure of the solution that has to strictly adhere to the domain ontology structure (for example the solution is divided in n pieces each one representing an instance of a concept);
- The expert gives to the system a set of problems and for each problem a set of possible solutions that solve the problem;
- The constraint base is divided in two subset: syntax constraints that impose restrictions on the syntax of solution and semantic constraints that are used to check the semantic requirement in the domain and problem to be solved;

- The system extracts syntax constraints directly from the ontology structure analysing the restriction on concepts attributes and relationships;
- The system learns semantic constraints from the examples given by the expert based on domain ontology with an ad-hoc algorithm;
- The system interacts with the domain expert showing the learned constraints that must be validated before they can be added to the knowledge base.

CAS uses the LISP language to store the constraints based essentially on pattern matching [28].

This method is a notable step ahead in knowledge acquisition and in our opinion this is the right direction for further development. We now analyses possible improvements to this methodology:

- The system requires a strongly structured solution. In many cases this is an unacceptable restriction. Furthermore this fact imposes an additional effort to the domain expert in order to specify the solution structure;
- The semantic constraints learning algorithm is too strictly coupled to the constraint representation.

We suppose to have a solution in a form of a free text to overcome the first problem. So, the solution can be a fragment of programming code or an essay in natural language or a text derived from another component acquiring the solution. A new problem is faced out with this solution representation: the system has to recognize the concepts specified in the domain ontology from a text. In this way we have moved the task of specifying the solution structure from the domain expert to the system. The second issue is resolved adopting the Prolog language for constraints representation instead of Lisp pattern matching. As we will show this type of encoding allows to formulate the problem of constrains learning as an Inductive Logic Programming (ILP) problem. This approach allows for a variety of consolidated techniques based on theoretical approach of first order logic. In the following we analize some systems that can be used for integration with proposed framework and then we choose the most appropriate for our purpose.

3.2 Information Extraction

From the previous observations we have to distinguish at least two cases in recognizing the entities that correspond to concepts in domain ontology:

- The solution is expressed in natural language resulting in a highly unstructured text;

- The solution is expressed in a programming language or a more restrictive formalism and can therefore be analyzed in different and (hopefully) efficient way.

In the former case information extraction task is defined as the operation of identifying a subset of information within a document. This subset of information can corresponds to predefined generic types of information of interest and represents specific instances found in the text. It can be assumed that such generic types of information correspond to the concepts and relation between them in the domain ontology. There are some kind of tools that allows to do this task even if in a not very reliable way. Example of such a tools is ANNIE which is an information extraction system embedded in the framework for language processing called GATE [3]. In the latter case the task can be accomplished by a parser that can detect the concepts and some kind of relation in a reliable way.

In both cases the system needs for a set of parser rules in order to extract useful information that can be used in successive phase of knowledge acquisition process. We must have in mind that the user should be able to define some kind of information that can be easily transformed in parser rule used for information extraction. The information we need to detect are the concepts involved in the domain and relation between such a concepts. To the best of our knowledge it does not exist yet a system capable of learning to accomplish this task in an acceptable way for text in natural language. A comprehensive description of the state of the art systems that use machine learning for information extraction can be found in [55]. It can be seen that does not exist a system reliable enough to be used in our task.

So, while for natural language text the process of information extraction is quite difficult and requires sophisticated tools and programming artifacts, the same process for structured text, like source code of programming languages, is more easily implementable. For this reasons we restrict ourselves to detect information in structured language.

In particular the user will define an ontology with given constructs that can be directly transformed into a parser. The parser will generate a parser tree that with some modification can be mapped to a knowledge base for the process of rule learning.

3.3 Inducing the constraints

The main problem of ITSs, is the complexity of authoring the domain model knowledge base. The effort needed to build even a system that tutor on simple domain, can be prohibitive especially for a user without any programming knowledge. For a cognitive tutor it has been estimated that an hour of instruction corresponds to 200 hours of system development. We propose to use Inductive Logic Programming as the learning mechanism for the knowledge base acquisition. The schema of constraint acquisition can be seen in figure 3.2

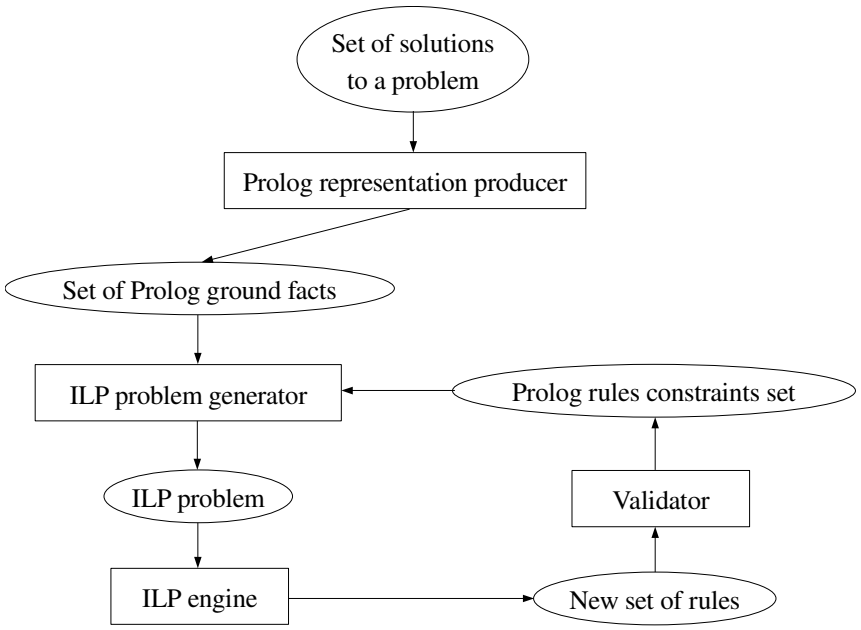

Figure 3.2: Schema of knowledge acquisition module

3.3.1 Inductive Logic Programming

Inductive logic programming is situated in the intersection of machine learning on the one hand, and logic programming on the other hand. It shares with the former fields the goal of finding patterns in data, patterns that can be used to build predictive models or to gain insight in the data. Furthermore it has inherited an experimental approach and orientation toward practical applications. With logic programming it shares the use of clausal first order logic as a representation language for both data and hypotheses in addition to its sounds theoretical basis.

The general problem of ILP is learning general rules or theory underlying a set of given examples. This problem can be formulated as task of learning concepts from examples, referred as *inductive concepts learning*. A definition of the problem is the following:

> *given a universal set of objects* U*, a concept* $C \subseteq U$*, learning a concept* C *means to learn to recognize objects in* C*, i.e. to be able to tell whether* $x \in C$ *for each* $x \in U$*.*

We briefly review some concept of logic programming which is a popular choice for language in order to describe objects and concepts. A first order alphabet is a set of predicate symbols, constant symbols and functor symbols. A clause is a formula of the form $A_1, ..., A_m \leftarrow B_1, ..., B_n$ where the A_i and B_i are logical atoms. An atom $p(t_1, ..., t_n)$ is a predicate symbol p followed by a bracketed n-tuple of terms t_i. A term t is a variable V or a functor symbol $f(t_1, ..., t_k)$ immediately

followed by a bracketed k-tuple of terms t_i. Constants are functor symbols of arity 0. Functor-free clauses are clauses that contain only variables as terms.

The above clause can be read as A_1 or ... or A_m if B_1 and ... and B_n . All variables in clauses are universally quantied, although this is not explicitly written. Extending the usual convention for denite clauses (where m = 1), we call $A_1, ..., A_m$ the head of the clause and $B_1, ..., B_n$ the body of the clause. A fact is a denite clause with an empty body, (m = 1, n = 0).

A Herbrand interpretation over a rst order alphabet is a set of ground atoms constructed with the predicate, constant and functor symbols in the alphabet. A Herbrand interpretation I is a model for a clause c if and only if for all grounding substitutions θ of c : $body(c)\theta \subset I \rightarrow head(c)\theta \cap I \neq \emptyset$. We also say c is true in I. A clausal theory T is a set of clauses. A Herbrand interpretation I is a model for a clausal theory T if and only if it is a model for all clauses in T. Roughly speaking, the truth of a clause c in an interpretation I can be determined by running the query `? body(c), not head(c)` on a database containing I using a theorem prover (such as Prolog). If the query succeeds, the clause is false in I. If it nitely fails, the clause is true. In the following we adopt the notation $A_1; ...; A_k$:-$B_1, ..., B_n$ to indicate a clause $A_1, ..., A_m \leftarrow B_1, ..., B_n$

In this representation both *objects* and *concept* corresponds to a set of clauses.

Suppose that we have to represent the concept of *daughter*. We can say that X is daughter of Y if X is female and Y is parent of X. In the chosen language

```
daughter(X,Y):-female(X),parent(Y,X).
```

where `daughter(X,Y)` is the consequent and `female(X),parent(Y,X)` is the antecedent. A set of fact for this concept is

```
parent(mario,lucia).     female(lucia).
parent(gianni,mara).     female(mara).
daughter(lucia,mario).   daughter(mara,gianni).
```

The general statement of inductive logic programming problem is the following:

Given is:

A background knowledge B, as a set of Horn clauses
A set of positive examples P, as a set of Horn clauses
A set of negative examples N, as a set of Horn clauses

Find an hypothesis H as a set of Horn clauses such that:
$\forall p \in P$: H $\cup$ B cover p
$\forall n \in N$: H $\cup$ B does not cover n

where the *cover* concept is different depending on the problem settings. The more frequently used settings are "*learning from entailment*" and "*learning from interpretation*" where the cover concept is respectively logic deduction and the Herbrand model mechanisms.

3.3.2 Representation of data

In order to apply the ILP paradigm for constraint learning, we need a suitable representation both for constraint and the annotated text solution. Following the definition of inductive concept learning:

- The set of positive and negative examples are supplied by the domain expert. The set of problems solutions represents our positive examples while it is possible to give also erroneous solutions that will be used as negative examples;
- The background knowledge, that can be initially empty, is the set of constraints in our domain. We suppose that once we have found a constraint, this is valid;
- The hypothesis to be found is our new constraints set that must be validated in a successive phase.

Two new components are needed to formulate a complete ILP problem: the *Prolog representation transducer* and the *ILP problem generator*. The former component takes as input an annotated text and the domain ontology and produces as output a set of Prolog ground fact that represents the instances of the concepts in the domain together with the relationship within the instances. As an example concepts can be predicates name and relationship can be a predicate that contains two concepts as parameters.

The latter component is responsible of creating the final ILP problem from the Prolog representation of examples provided by the expert. This module produces the input files needed by the ILP engine used by the system. As an example depending on the chosen ILP engine the module must produce a suitable set of language bias, background knowledge and settings files. We give more details on next section. Once the problem is completed in all its parts, it can be processed by an ILP engine.

We consider some systems and then we choose the more appropriate. In next section we give more details on chosen system.

The system ClassicCL [49] is able to find rules that can relate each predicate in the example with each other. In this setting, called "learning from interpretation", the provided examples are considered as Herbrand interpretations that are models for the searched rules. In this way the found rules that are true in the positive examples. ClassicCL is able to find rules from both only positive and positive and negative examples. The produced rules are in this form: "if B then H"

where B is a conjunction of atoms and H can be a disjunction of atoms where all the variables are universally quantified. WARMR [14] and FARMER [36] systems for mining frequent query in a deductive database. A deductive database is substantially a set of Prolog ground facts and a set of Prolog rules with some restriction (in particular only DATALOG patterns are considered). We give more details in chapter 4. These kinds of systesms can be used to find rules in the form of existentially quantified first order formula like the following "if $A_1, ..., A_i$ holds in data then $A_1, ..., A_i, ...A_n$ holds in data" where A_i is a logical atom. ALEPH [48] is a highly flexible ILP system that is able to reproduce many algorithms of other ILP engine like PROGOL [] and FOIL []. Mainly this kind of system is able to learn single concepts about a given set example describing this concept. Its mainly purpose is to learn rules for classifying object in different classes.

After some investigation we find that the settings of ClassicCL, WARMR and FARMER

3.4 Hypothetical scenarios

We describe now a hypothetical scenario where the teacher interacts with the depicted system for authoring the intelligent tutor. We can contextualize the process inside a typical Learning Content Management System (LCMS) which the teacher uses to insert material such as html pages and questions for auto assessment. As an example we suppose to speak about the World Wide Web and related arguments.

The teacher inserts the following statement inside a page that tells about hypertext:

> *A hypertext is a document containing a number of link (connection or pointer) to other documents or sections of the same document. Following the links, the user can jump from one point to another inside a document or to other documents*

Now the teacher wants to formulate a question to test the comprehension of this statement. For example "*What does a hypertext contain?*", than she gives a number of plausible answers to the system:

- *"A hypertext is a document that contains links to some portions of itself or links to some other document. The user can follow these links to jump to other documents or inside the same document"*
- *"A hypertext is constituted by a set of link that can point inside the same document or to other documents. Links can be used to jump to other hypertext or inside the same one"*

The teacher now identifies the main concepts involved in the statement like "*hypertext*", "*link*", "*document*", "*document portion*" and the corresponding relationship such as "*contains*" that holds between document and link, "*pointTo*" that holds between link and document. A possible class hierarchy is depicted in figure 3.3 The teacher than trains the AIE module to recognize the concepts

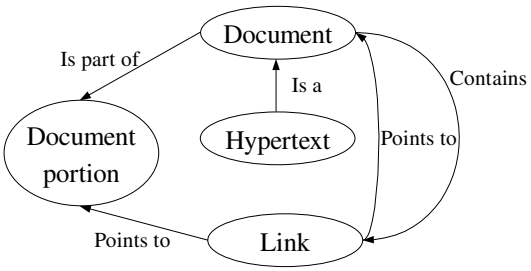

Figure 3.3: A representation of some concepts related to hypertext

she has specified by highlighting some portion of text and selecting the appropriate concept. In this case the text corpus is too small to train the system but we suppose that at end of training the system is able to recognize these concepts.

Now the system has to learn the related constraints involved in solving the proposed problem. After the learning process is applied, the system should be able to induce constraints like the following:

- *If the teacher solution contains the concept of hypertext and the concept of link related by a "contains" relation and the student solution has a concept of hypertext then the student solution must contain the concepts of link*
- *If both the teacher and student solution contains the concept of hypertext and link and in teacher solution they are related by a "contains" relation then the student solution must contain the concepts of link related by the a "contains" relation*

At this point the teacher validates the constraints and attaches a feedback to each of them. When the problem will be proposed to the student, any sentence that contains these two concepts and relates them with the "contains" relation, will be recognized as valid because it does not violate the constraint. Otherwise an appropriate feedback is shown to the student.

If we consider teaching a programming language the situation is similar. In the ideal case the teacher gives to system a grammar for the language and the system automatically generates the corresponding ontology that can be also modified by the teacher to improve the structure or simply to rename some concepts or relations.

If there is a material that explains some of the concepts derived from the grammar the teacher trains the AIE system to recognize these concepts inside the explanations. The representation of material can be used also to produce automatically the feedback when a solution given by a student violates some constraints.

Now suppose that teacher wants to check the comprehension of the iterative constructs of the C language by giving the following problem: "*Compute the sum of an array of integer*". Two possible

Solution 1	Solution 2

`int v[10];`
`int sum;`
`sum=0;`
`for(int i=0;i<10;i++) {`
`  sum+=v[i];`
`}`
	`int v[10];`
	`int sum;`
	`sum=0;`
	`int i=0;`
	`while(i<=9) {`
	`  sum=sum+v[i];`
	`  i++;`
	`}`

Table 3.1: Two possible solutions to the problem of computing the sum of an array

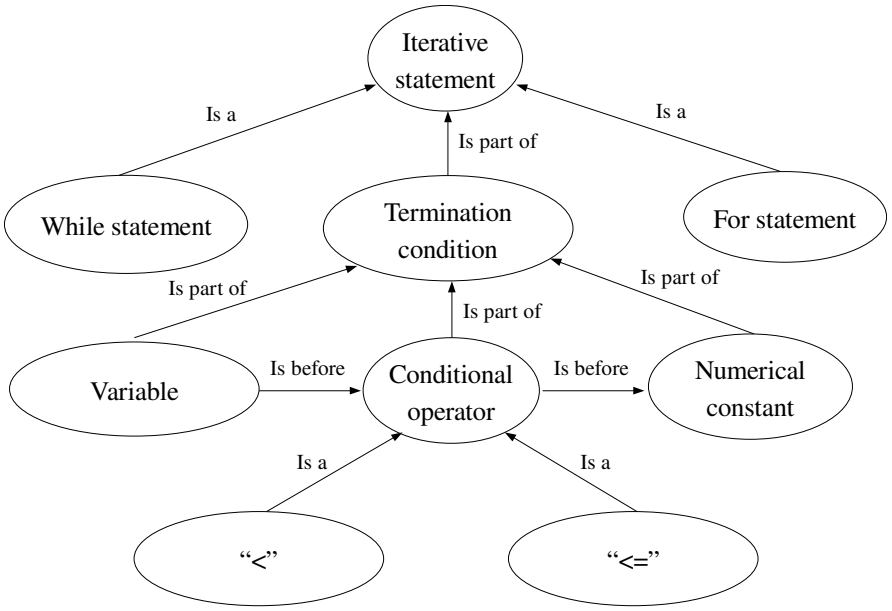

Figure 3.4: Ontology for some concepts and relation of C language

solutions are shown in table 3.1 The system recognizes all the possible concepts and we suppose to analyze the concepts of "iterative construct", "while", "for", "termination condition" simplifying the termination condition as composed by a "variable", "operator" and "numerical constant". A possible corresponding ontology is shown in figure 3.4. This ontology is not complete because there are not some other concepts that can be found in the above pieces of code such that "array" or "arithmetic expression" etc...

The system now should be able to induce some constraints that specify the presence of an iterative construct and that the termination condition can be either "i<10" or "i< =9". In this way if the student writes the solution:

```
int v[10];
int sum;
sum=0;
for(int i=0;i<=9;i++) {
sum+=v[i];
}
```

that has not been specified by the teacher, the system can check it against the constraints and recognize it is a right solution even if it does not exist in the set of teacher answers.

3.5 Summary

In this section we have presented a general architecture for knowledge acquisition in an Intelligent Tutoring System based on Inductive Logic Programming and reviewed some previous works. Our system is based on the idea of Constraint-based Tutor where the system knowledge is made up of rules that have to be satisfied by a solution in order to be correct.

In general the system knowledge base is made up of three components:

- a **domain ontology** describing the concepts and relations surrounding the domain concepts;
- a **set of parser rules** for identifying the entity in the provided solution;
- a **constraints base**, expressed in Prolog language, that a solution must satisfy in order to be correct.

The ontology and the parser rules are used by an Information Extraction System (IES) to tag the various parts of solution in order to recognize the entities corresponding to the concepts in the domain ontology. This process can be accomplished using ANNIE, a set of tools for Natural

Language processing that is a part of the GATE framework [3] in the case of solution expressed in natural language or by a specialized parser in the case of programming language.

The component called "*Tagged text to Prolog transducer*" is used to obtain a version of the tagged solution in the form of Prolog ground facts. This set of facts must represent the information corresponding to the solution text and its structure must agree with the domain ontology.

In this form the solution can be checked against the constraint base. Following the work of [31, 51, 53] each rule in the constraint base has attached a feedback that explains the error concerning the constraint violation. Error checking is accomplished by the "*Solution Checker component*".

The phase of knowledge acquisition can be divided in three major tasks: ontology building, creation of the set of parser rules and construction of constraint base. We will focus on automating the construction of constraint base even if there are some methods that can be used to automate the other two tasks. The problem of finding a general rule that must be satisfied in one domain can be formulated as an Inductive Logic Programming (ILP) problem. In a general setting an ILP problem is formulated as follow:

> given a set of training positive and negative examples E and background knowledge B, where B is a set of rules, find a hypothesis H, expressed in some concept description language, such that H is complete and consistent with respect to the background knowledge B and the examples E.

In our case the examples, background knowledge and hypothesis are expressed in Horn clauses (Prolog language). In our setting the positive, and optionally the negative, examples are given as a set of solutions to the same problem. Such solutions are then processed by the Prolog representation producer module in order to obtain a suitable representation to be processed in an ILP setting. The background knowledge is represented by the constraint set that initially is empty. This information is processed by another component called ILP problem generator that produces a set of data that can be directly processed by an ILP engine like ALEPH [48]. The ILP engine will produce a new set of rules representing the constraints related to the provided examples. Such set must be validated before it can be added to the constraint database. The whole process can be repeated until all necessary constraints are found.

Chapter 4

RuleMinator: disjunctive first order association rules in ITSs

In this chapter we describe the module for knowledge acquisition in more detail and give some simple run example. The process is based on algorithms for data mining in particular the ones that find association rules in given set of patterns. The process is articulated in two macro steps: first we find frequent patterns in the provided examples then we find disjunctive association rules from frequent pattern but in a new space of examples defined by a cartesian product like operator between examples. More detail in following sections.

From now on we will consider DATALOG (a restriction of Prolog to function-free definite clauses) for representing data and patterns. We also consider a further restriction for data representation to graph to obtain an improved performance in pattern mining.

4.1 Preliminary Concepts

Data mining algorithms are designed to find interesting patterns from a large dataset [7]. Many of existing algorithms are based on propositional logic and the mined data are stored in a single database table composed of many fields. The interest for mining multi-relational data (data stored in more than one table) algorithms have been increasingly grown. This type of algorithms is often based on "Inductive Logic Programming" and aim to find patterns in which more than one relation is involved.

Propositional pattern that are mined by "traditional" data mining algorithms are extended by relational algorithms so that we can have relational classification rules, relational classification trees, relational association rules, etc...

In table 4.1 some data of example are shown. A comparison between propositional and relational rule, with respect to the data of the table, can be:

Costumer Table

ID	Gender	Age	Income	TotalSpent	BigSpender
c1	male	30	214000	18800	yes
c2	female	19	139000	15100	yes
c3	male	55	50000	12400	no
c4	female	48	26000	8600	no
c5	male	63	191000	28100	yes
c6	male	63	114000	20400	yes
c7	male	58	38000	11800	no
c8	male	22	39000	5700	no
...	...	...	...	...	...

MarriedTo Table

Spouse1	Spouse2
c1	c2
c2	c1
c3	c4
c4	c3
c5	c12
c6	c14
...	...

Table 4.1: Customer table and MarriedTo relation. Example from [15]

Propositional rule

IF Income $>$ 108000 THEN BigSpender = yes

Relational rule

BigSpender(C1,Age1,Income1,TotalSpent1) $\rightarrow$

MarriedTo(C1,C2) $\vee$

Customer(C2,Age2,Income2,TotalSpent2,BigSpender2) $\vee$

Income2 $>=$ 108000.

If we take into consideration more than one relations to represent patterns, we have the advantage of a greater expressivity respect to propositional setting. Multi relational representation allows to express concepts otherwise impossible to express in the propositional formulation. The relational patterns, in fact, are expressed with the first order logic language or, more often, with a subset of it.

Even if the problem can be "proposizionalized" e.g. making a join on the interested tables, and resolved with traditional algorithms, there are some problems as the loss of significance in the found rules, a relevant grow of the data to deal with and a loss of the information in the case of grouping some data after the join operation.

Suppose we are given the relations *customer(CustID,Name,Age,SpendsALot)* and *purchase(CustID, ProductID,Date,Value,PaymentMode)*, where each customer can make multiple purchases, and we are interested in characterizing customers that spend a lot. If we natural join the relations we obtain a single relation whit a single purchase in each row so we lost significance for customer characterization. On the other hand if we aggregate the purchase for each client we can lost some useful information. If we consider the two tables together we dont have this type of problems. However we pay in term of computational complexity the greater expressivity of the pattern.

WARMR [13] system is an upgrade of APRIORI [7] algorithm to the relational framework. In APRIORI we want to find all patterns in a table satisfying a support frequency criterion. The patterns are subset of the table attributes and are called *itemsets*. In WARMR the *itemset* concept is extended with the *query* concept while the *association rule* concept is extended with *query extension* concept.

All data in WARMR are expressed in DATALOG language, a subset of the clausal logic where the clauses are definite (with an atom in the head) and do not contain function symbols. In Datalog a *term* is defined as a constant symbol or a variable. To distinguish between them, we write variables with an initial upper case letter, while using names beginning with lower case letters for constants. An *atom* is an *m*-ary *predicate* symbol followed by a bracketed *m*-tuple of terms. A *definite clause* is a universally quantified formula of the form $B \leftarrow A_1, \ldots, A_n (n \geq 0)$, where B and the A_i are atoms. This formula can be read as "B $\mathtt{if}$ A_1 $\mathtt{and}$ $\ldots$ $\mathtt{and}$ A_n". If $n = 0$ a definite clause

is also called a *fact*. *Ground* clauses are clauses that contain only constants as terms, no variables. A substitution θ is a set of bindings $\{X_1/a_1, \ldots, X_m/a_m\}$ of variables X_i to terms a_i. If we substitute a_i for each occurrence of X_i in a clause C, we obtain $C\theta$, the instance of C by substitution θ. If $C\theta$ is ground, it is called a ground instance of formula C, and θ is called a grounding substitution. The search space for first order data mining algorithms is defined by mean of θ-subsumption relation. Clause C θ-subsumes D iff there exists a substitution $\theta = \{X_i/V_{i_j}\}$, where X_i ranges over the variables in C and V_{i_j} ranges over the variables and constants in D, such that $C\theta \subseteq D$.

A deductive Datalog database is a set of definite clauses. Often a distinction is made between the extensional database, which contains the predicates defined by means of ground facts only, and the remaining intensional part.

A Datalog (and Prolog) query is a logical expression of the form ?-$A_1, \ldots, A_n$. Submitting such a query to a Datalog database corresponds to asking the question "does a grounding substitution exist such that conjunction A_1 and ... and A_n holds within the database". The (resolution based derivation of the) answer to a given query with variables $\{X_1, \ldots, X_m\}$ binds these variables to terms $\{a_1, \ldots, a_m\}$, such that the query succeeds if a_i is substituted for each X_i. This so-called *answering substitution* is denoted by $\{X_1/a_1, \ldots, X_m/a_m\}$. Due to the nondeterministic nature of the computation of answers, a single query Q may result in many answering substitutions. We will refer by $answerset(Q, \mathbf{r})$ to the set of all answering substitutions obtained by submitting query Q to a Datalog database $\mathbf{r}$.

The query mining task is defined as follow:

> Assume $\mathbf{r}$ is a Datalog database, $\mathcal{L}$ is a set of Datalog queries Q that all contain an atom *key*, and $q(\mathbf{r}, Q)$ is true if and only if the frequency of query $Q \in \mathcal{L}$ with respect to $\mathbf{r}$ given key is at least equal to the frequency threshold specified by the user. The *frequent query discovery task* is to find the set $Th(\mathcal{L}, \mathbf{r}, q, key)$ of frequent queries.

The parameter key in the problem definition is used to tell the algorithm what have to be counted during the search process to compute the frequency of the query. In propositional setting the things to count are implicitly defined in the problem definition. The frequency of a query is the number of substitutions of variables in *key* in which query *Q* holds divided by the number of all substitutions where *key* holds. Formally the frequency of a query $Q \in \mathcal{L}$ that contain an atom key w.r.t database $\mathbf{r}$ is defined as follow:

$$freq(Q, \mathbf{r}, key) = \frac{|\{\theta_k \in answerset(?\text{-}key, \mathbf{r}) \mid Q\theta_k \text{ succeeds w.r.t. } \mathbf{r}\}|}{|\theta_k \in answerset(?\text{-}key, \mathbf{r})|}$$

Substantially a DATALOG (PROLOG) query ?- key(...) counts the number of solutions which corresponds to all the objects in the database while the DATALOG query ?- key(...),Q counts the solutions the query Q holds.

A frequent query Q is *closed* if for each query Q_1 such that Q θ-subsumes Q_1 (i.e. Q_1 is more specific than Q) the frequency of Q_1 is less than the frequency of Q.

WARMR algorithm, as in APRIORI [7], searches in the lattice induced by an order relation. WARMR uses θ-subsumption to build the lattice while APRIORI uses the classical set subsumption. In the WARMR algorithm the search space can be pruned based on monotonicity of θ-subsumption with respect to frequency of query.

WARMR can be considered a proof-of-concept system that does not care enough about the efficiency issues of the mining process. Often the system can not mine queries of length greater than six or seven atoms in reasonable time. Our system has to do this process in few seconds in order to be usable by a teacher that is authoring its own intelligent tutoring system. In the following section we briefly review two state of the systems that is more efficient than WARMR at the cost of making some assumptions that restrict the flexibility on the represention of patterns.

4.2 State-of-the-art of first order data mining

4.2.1 Frequent query discovery: FARMER

FARMER [36] is a system for mining frequent DATALOG queries from a dataset that uses a formalism close to WARMR and at the same time is implemented to reach an efficiency that is two order of magnitude better than those of WARMR. The main problems of the WARMR algorithm are the following:

- WARMR uses basically a generate and test algorithm to accomplish its tasks so many equivalent queries can be generated during the mining process;
- WARMR uses θ-subsumption based test to count the frequency of the candidate query that is known to be NP-complete.

FARMER changes the classical query discovery task in to query discovery under *Object Identity* [17, 26]. This choice closely matches that of subgraph mining [37, 59] that is considerably more efficient than the general query mining, is very natural and does not pose restrictions in many data mining situations. Based on Object Identity framework, FARMER uses a trie data structure and define an order on queries that allows for more efficient search space traversals than the approach used by WARMR. This partially solves the redundant query generation problem that is present in WARMR algorithm. Furthermore all the computed evaluation of query against the dataset are kept in memory to avoid redudant computation of query matching. In this way FARMER can completely avoid the need for θ-subsumption test for query counting reducing the computation time of two order of magnitude respect to WARMR.

We briefly review the main concepts of Object Identity framework and the consequence on the represention of mined patterns. The Object Identity bias [26] is defined as follow:

> Under *Object Identity* within a Datalog clause, terms (even variables) denoted with different symbols must be distinct (i.e., they must refer to different objects).

For example under OI assumption the Datalog clause $C = p(X) : -q(X,X), q(Y,a).$ is an abbreviation for the Datalog^{OI} (a logic language resulting from the application of OI to Datalog) clause $C_{OI} = p(X) : -q(X,X), q(Y,a), X \neq Y], [X \neq a], [Y \neq a]$. Such a restriction allows for a more efficient search in the space of solution because computing the query frequency and queries equivalence is more easy than using normal θ-subsumption.

The consequence on the patterns mined under OI assumpuntion is that they are a proper superset of patterns mined with normal θ-subsumption. This can be easly seen through the following example: suppose we have a pettern representing a left hand side of equation having two integer and one integer respectively: $lhs(X1), has(X1,X2), integer(X2), has(X1,X3), integer(X3)$
$lhs(X1), has(X1,X2), integer(X2)$
Under normal θ-subsumption these two patterns are equivalent applying the substitution $S = \{X2/X3\}$ but are two distinct patterns under OI assumption because the variables X2 and X3 must be distinct objects (as normally humans are likely to think). The algorithm outline is as follow:

4.2.2 Frequent graph mining

The graph mining problem can be considered a specialization of the query mining problem where further restriction is that the data is represented as labeled graphs. We describe some fundamental concepts regarding this problem because some of these concepts will be used for implementing the system that infers the rules for ITS.

A labeled graph G is defined as a tuple $G = \{V, E, \Sigma_V, \Sigma_E, l\}$ where V is a set of vertices and $E \subseteq V \times V$ is a set of undirected edges. Σ_V and Σ_E are the sets of vertex labels and edge labels respectively. The labeling function l defines the mappings $V \rightarrow \Sigma_V$ and $E \rightarrow \Sigma_E$.

Given a pair of labeled graphs $G = \{V, E, \Sigma_V, \Sigma_E, l\}$ and $G' = \{V', E', \Sigma'_V, \Sigma'_E, l'\}$ G is a ***subgraph*** of G' iff:

- $V \subseteq V'$,
- $\forall u \in V, (l(u) = l'(u))$,
- $E \subseteq E'$,
- $\forall (u,v) \in E, (l(u,v) = l'(u,v))$.

Given the previous definition G is an ***induced subgraph*** of G' if G is a ***subgraph*** of G' and

- $\forall u, v \in V', ((u, v) \in E' \Leftrightarrow (u, v) \in E)$.

That is, G is an induced subgraph of G' if it has the most edges that appear in G' over the same vertex set.

Given the previous definition G is ***isomorphic*** to G' iff there exist a bijection $f : V \rightarrow V'$ such that

- $\forall u \in V, (l(u) = l'(f(u)))$,
- $\forall u, v \in V, ((u, v) \in E \Leftrightarrow (f(u), f(v)) \in E$,
- $\forall (u, v) \in E, (l(u, v) = l'(f(u), f(v)))$.

The bijection f is an *isomorphism* between G and G'.we also say that G is isomorphic to G' and vice versa.

A labeled graph G is ***(induced)subgraph isomorphic*** to a labeled graph G', denoted by $G \subseteq G'$, iff there exists a (induced)subgraph G'' of G' such that G is isomorphic to G''.

Given a set of graphs GD (referred as a graph database) and a threshold σ $(0 < \sigma \leqslant 1)$, the frequence of a graph G, denoted by $freq_G$ is defined as the fraction of graphs in GD to which G is (induced)subgraph isomorphic.

$$freq_G = \frac{|\{G' \in GD | G \subseteq G'\}|}{|GD|}$$

G is ***frequent*** iff $supG \geq \sigma$. The frequent (induced)subgraph mining problem is: "given a threshold σ and a graph database GD, finding all frequent (induced)subgraphs in GD".

The problem of query mining under object identity without support for the intensional knowledge can be reduced to the problem of frequent induced subgraph mining. It is easy to find a transformation of data representation from a DATALOG database to a graph database without changing the initial problem.

Several algorithms have been developed to discover arbitrary connected subgraphs. One research direction has been to explore ways of computing canonical labels for candidate graphs to speed up duplicate candidate detection [24, 59]. Various schemes have been proposed to reduce the number of generated duplicates graphs during search [21, 59]. In particular, candidates that are not in canonical form need not be developed and investigated further. Gaston [37] approach leverages the fact that many graph patterns are in fact not full cyclic graphs but trees or even paths (which have simpler canonical forms) has been shown to be fastest on a number of datasets [38, 58].

But almost all of these systems do not support the mining of induced subgraph and unless you specify the knowledge base natively as a graph it is not possible to use them directly instead of a query miner as WARMR or FARMER.

4.3 RuleMinator: Disjunctive rules mining from frequent query

We now define a mechanism for extraction of rules from examples based on the query mining process that can be adapted smoothly to the constraint based tutor that we called RuleMinator. During the description of this mechanism we illustrate the example of the process on the simple domain of algebra equation such that the explanation is more clear.

The context is described in section 3.4. The teacher must supply to the system a set of N problems along with their possible solutions. The system, at this point, applies an algorithm for extraction of rules that fit all the given examples and then shows them to the teacher for the validation phase and attribution of the relative feedback to each rule. The feedback is shown in tutoring phase when a rule is violated.

As we have shown in chapter 3 the general architecture for knowledge base acquisition is made of three phases: ontology building, creation of a set of parser rules and construction of constraint base. We will focus on automating the construction of constraint base even if there are some methods that can be used to automate the other two tasks.

Remember that the problem of finding a general rule that must be satisfied in a domain, can be formulated as an Inductive Logic Programming (ILP) [25] problem. In our setting teacher gives only positive examples as a set of solutions to the same problem. Such solutions are then processed by the *Prolog representation producer* module in order to obtain a suitable representation to be processed in an ILP setting. The background knowledge is represented by the ontology opportunely transformed in Prolog terms and rules and it is used to map the solution in Prolog terms. It can be used in the second phase of rules generation to select the most interesting rules but this feature is not implemented yet.

These information are processed by another component called "ILP problem generator" that produces a set of data that can be directly processed by an external ILP engine that in our case is FARMER augmented with an algorithm to extract disjunctive association rules that we call RuleMinator. RuleMinator will produce a new set of rules representing the constraints related to the provided examples. Such set of rules must be validated before it can be added to the constraints database. The whole process can be repeated until all necessary constraints are found. Currently, constraints base has to be generated from scratch each time the teacher adds new problem but in the future we plan to change the mechanism to modify the added constraint.

The algorithm outline is shown in figure 4.1

In the following subsections we give the implementation details along with the example an algebra problem description.

1. Transform given example in input files for FARMER
2. Find all frequent query in given examples
3. Build the closed query lattice
4. Generate the disjunctive association rules
5. Transform the association rules in the new space of queries
6. Order the rules based on predefined heuristics

Figure 4.1: Algorithm to mine disjunctive association rules in the ITS settings

4.3.1 Producing input for FARMER

As in all ILP based systems also FARMER uses a language bias declaration in order to constrain the search space to valid patterns described by such a formalism. Therefore in order to supply a valid input to the system of query mining we have to produce two files starting from a solution given by the user: one file containing the description of language bias and one file containing knowledge base on which carrying out the search of interesting patterns.

We briefly describe the specification of FARMER language bias and then we illustrate the model adopted for its automatic generation. A *mode declaration* $p(c_1, \ldots, c_n)$ consists of a predicate with arguments c_i, each of which is either '+' (input), - (output) or # (constant). The bias B of a search space consists of: 1) the type definitions of predicates, 2) a set of modes M, 3) an operator const(T) which defines for each type T a set of constants, 4) a function max which assigns an integer to each predicate in M, and 5) one atom $k(X)$ (this atom is called the key of the search).

The search space is defined as follow: Given a bias B, a query Q and an atom $A = p(t_1, \ldots, t_n)$, atom A is a (mode) refinement of Q iff there is a mode $M = p(c_1, \ldots, c_n) \in M$ such that for every $1 \leq i \leq n$ either:

- t_i is a variable in $var(Q, T_i)$ and c_i ='+', or
- t_i is a variable not in $\bigcup_j var(Q, T_j)$ and c_i ='-', or
- t_i is a constant in const(Ti) and ci ='#'

Inoltre FARMER aggiunge un meccanismo basato su *mode trees* in cui possibile definire mode declaration in forma di albero. A "mode" or "key" declaration "M" may be followed by a "b." fact on the next line. The "b." fact denotes the start of a set of modes; one mode in this set is applied during query construction immediately after mode "M" has been applied. The modes are never

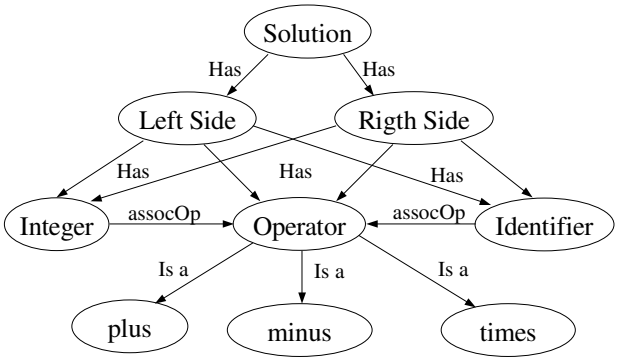

Figure 4.2: Partial ontology for equation of first order

applied in any other situation. "M" is said to be the parent of the modes within the block. "b." ..."e." blocks may be nested. A mode within a nested "b."..."e." block can therefore have several ancestors. Within a "b."..."e." block integers may be used as mode arguments. Such an integer marker refers to the argument of an ancestor mode (to this purpose, in an ancestor mode an argument can be suffixed with an integer to mark that mode argument). When a mode in a "b."..."e." block is applied to introduce an atom, for arguments marked with an integer the term is used that occurs in the atom introduced by the ancestor mode. Within a "b."..."e." block no two modes may occur that can possibly generate the same atom.

With this language bias mechanism it is possible to define mode describing trees, undirected graph beyond more general structures such as Datalog queries. FARMER does not support intensional knowledge base so during the phase of pattern mining, we supposes not to have further information on domain such as hierarchy of concepts or other rules that define particular relations. This issue can be resolved expanding the intensional knowledge using the extensional knowledge but this would require a number of additional operations that could degrade the total performances of the system.

The language bias for this problem is shown in figure 4.3 and represents a language that generates only connected graphs. There are three kind of predicate: `vertex` that corresponds to a unary predicate, `edge` corresponds to a binary predicate that introduce a new concept, `edge1` corresponds to a binary predicate that does not introduce a new concept and `edge2` corresponds to a binary predicate that introduce a property (a constant) related to an existing concept.

We use the example of the domain of the algebraic equations in order to better specify the transformation process of the solution in input for system FARMER. The figure 4.2 shows a partial ontology of the domain associated to this type of problems. The solution will consist in a instance of some concepts related by the relative present relations if they are in the solution. Corresponding representation in Datalog terms can be easy obtained making in correspondence the concepts

```
key(simkey(-)).
mode(solution(+1,#,-2)).
b.
  mode(edge(+1,#,+2,-3)).
  b.
    mode(vertex(1,#,3)).
  e.
e.
mode(edge(+1,#,+,-3)).
b.
  mode(vertex(1,#,3)).
e.
mode(edge2(+,#,+,#)).
mode(edge1(+,#,+,+)).
```

Figure 4.3: Language bias for mining frequent pattern in algebra domain

to unary predicates and the relations to binary predicates. For the explanation we transform the predicates in the language bias in the following way: `vertex(Key,Pred,Var)` $\rightarrow$ `Pred(Var)` and various type of `edgex(Key,Pred,Var1,Var2)` $\rightarrow$ `Pred(Var1,Var2)`.

As an example consider the equation $3x - 4 = 2$. Following the ontology we obtain the representation depicted in figure 4.4 that represents the solution in terms of concepts and relation. With the convention described before it is easy to obtain the representation in datalog terms as follow:

```
solution(sol),has(sol,lhs1),lhs(lhs1),has(sol,rhs1),rhs(rhs1),
has(lhs1,times1),times(times1),has(lhs1,id1),identifier(id1),
has(lhs1,int1),integer(int1),has(rhs1,int2),integer(int2),
hasValue(int1,3),assocOp(times1,id1),assocOp(int1,times1),
........
```

4.3.2 Closed query lattice generation

For a better visualization of the rules it was adopted the strategy described in [50] where the queries are arranged in a closed query lattice from which it is possible to extract non redundant

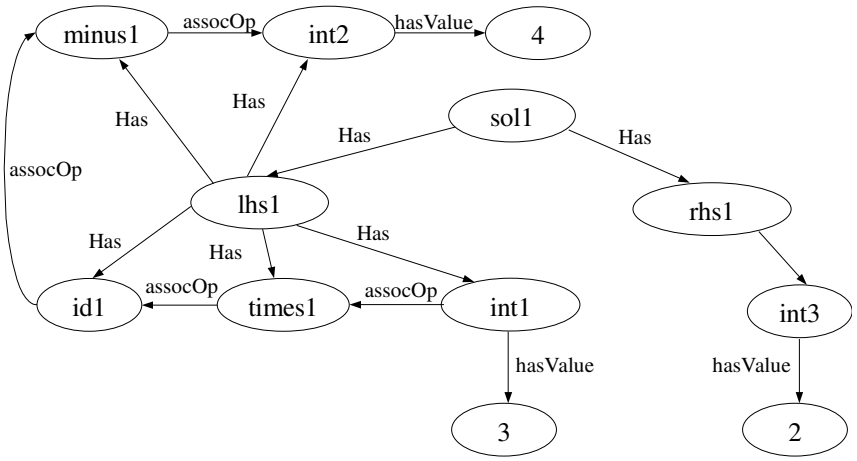

Figure 4.4: Graph representation of the equation $3x - 4 = 2$

association rules. Farmer outputs a trie data structure containing all possible queries having a support greater than a given value so we have to adopt a mechanism that generates the closed query lattice. We have chosen to post process FARMER output in spite of directly modify the original algorithm because it does not exist yet an efficient mechanism for closed query mining and because of the realization difficulty. This rices a problem in terms of performance due to the fact we have to do numerous subsumption tests. In order to obviate to this problem we transform the original theta-subsumption under object identity test to one equivalent subgraph isomorphism test for labeled graphs. It is in fact easy to see that adopting the following transformation the problem of theta^{OI}-subsumption can be reduced to a subgraph matching problem:

- For each variable and constant in the query there is a corresponding node on the graph;
- For each atom in the query there is a corresponding node on the graph: the node are labeled with the type of atom, formed by the name of atom and the constants in the predicate, and the arity of atom;
- For each variable and constant in an atom there is a labeled edge with label identifying the atom type and the position of the variable in the atom.

As an example consider the following two clauses under OI-subsumption hypothesis $C1 = t1(X), a(X, Y), t2(Y), b(Y, Z), t3(Z)$ and $C2 = t3(B), b(A, B), t2(A)$. With the following substitution $\{A/Y, B/Z\}$ clause C2 subsumes clause C1. If we apply the trasnformation described above we obtain the labeled graph in figure 4.5.

We have chosen VF2LIB [12] library for the subgraph matching. In comparison to a normal Prolog engine as tuProlog [6] the speed up observed it is around three orders of magnitude. After post processing the number of queries turning out from nearly 2000 to 10.

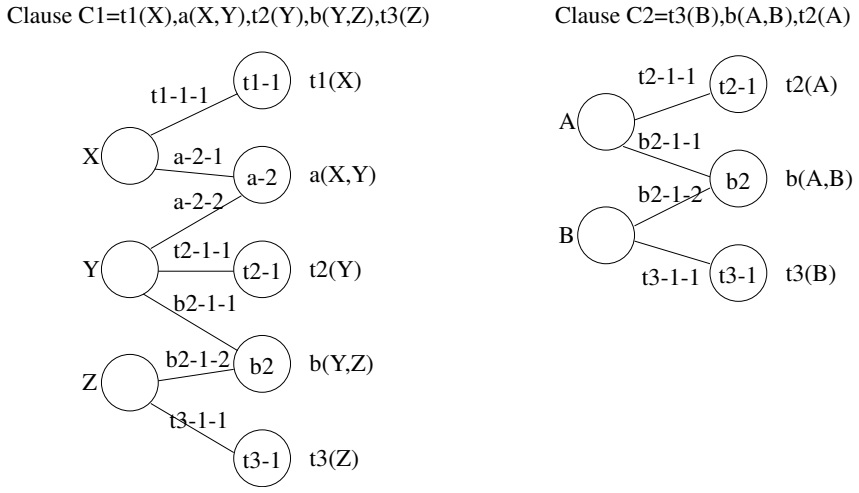

Figure 4.5: Graph representation of the clauses C1 = $t1(X), a(X,Y), t2(Y), b(Y,Z), t3(Z)$ and C2 = $t3(B), b(A,B), t2(A)$

4.3.3 Find disjunctive association rules for the ITS

The objective in the context of an ITS is finding all exact rules that is 100% confident rules. This is needed because we must be sure that the rule can be applied in all the cases of resolution of the problems presented to the system. For this purpose one want to find also association rules that have also disjunctions on the consequent part. As an example for expressing the fact that "if in the left hand side of an ideal solution of equation there is a plus operator than either in the student solution left hand side there must be a plus operator or in right hand side there must be a minus operator" the traditional definition of association rule is not sufficient Given a set of frequent query $\mathcal{S} = \{Q_p Q_1, \ldots, Q_p Q_n\}$ where Q_p is a common prefix containing an atom key and a Datalog database $\mathbf{r}$ a *disjunctive multi relational association rule* (D-rule) $\mathcal{R}_{p_n}$ is an existential quantified formula in this form: $Q_p \Rightarrow Q_p Q_1 \vee \cdots \vee Q_p Q_n$. This formula should be read as "if query ?-Q_p succeeds then either the query ?-$Q_p Q_1$ succeeds also or the query ?-$Q_p Q_2$ succeeds also ... or the query ?-$Q_p Q_n$ succeeds also" The relative frequency of a D-rule is defined as follow:

$$freq(\mathcal{R}_{p_n}) = \frac{|answerset(?\text{-}Q_p, \mathbf{r}) \bigcap (\bigcup_i answerset(?\text{-}Q_p Q_i, \mathbf{r}))|}{|answerset(?\text{-}key, \mathbf{r})|}$$

and its confidence is defined as:

$$conf(\mathcal{R}_{p_n}) = \frac{|answerset(?\text{-}Q_p, \mathbf{r}) \bigcap (\bigcup_i answerset(?\text{-}Q_p Q_i, \mathbf{r}))|}{|answerset(?\text{-}Q_p, \mathbf{r})|}$$

Once the frequent association rules are found they must be transformed in a suitable form for being used in the ITS. Until now the produced rules have been generated starting from the examples given from the teacher but for being usable by the system they have to be generated taking into

account that the given examples have to be considered in a new space defined by the cartesian product of the answers to the same problem. Formally given n problems P_i having n_i solutions S_{i_j} with $0 < j \leq n_i$ we consider the new data set $\mathcal{D}$ defined as follow:

$$\mathcal{D} = \left\{ S \mid \forall P_i \quad S = S_{i_j} \times S_{i_k}, \quad 0 \leq j, k \leq n_i \right\}$$

In this way each couple of solutions are treated as if they were respectively ideal solution and student solution obtaining consequently rules that express in the premise characteristic pertaining to the teacher solution and in the consequent the facts that must hold in order for the rule to be satisfied.

Once the rules have been generated they have to be listed to the user in comprehensible way and order by their importance. Since the obtained rules are all exact rules (i.e. 100% confident rules) heuristics, other than confidence, are needed in order to decide how to order rules. Currently those used are based on the length of the rule defined on the number of atoms (the more the rule are simple the more are probable it is interesting), number of disjunctions in the body of the rule (the few is the number of disjunctions the more the rule is interesting), intersection of the support of the disjunctions.

4.3.4 Test for a simple algebra equation tutor

We test the algorithm with a problem on simple domain to see if the rules founded by the system are plausible. The ontology in figure 4.2 shows a partial representation of domain concepts and relations in the domain of first grade albegra equation

Suppose we have a problem which allows the following solutions:

```
3*x-4=2;
3*x-2=4;
3*x=4+2;
```

We suppose to write the solution without simplification for now because of the difficult of FARMER to include intensional knowledge in the mining process. The resulting representation in Prolog fact for the first equation is:

```
solution(V2N0),has(V2N0,V2N1),lhs(V2N1),has(V2N0,V2N2),rhs(V2N2),
has(V2N1,V2N3),times(V2N3),has(V2N1,V2N4),identifier(V2N4),
has(V2N1,V2N5),integer(V2N5),has(V2N2,V2N6),integer(V2N6),
hasValue(V2N5,3),assocOp(V2N3,V2N4),assocOp(V2N5,V2N3),
........
```

After running the system on this simple set of solution it presents the following rules as the most important (we report only a few to show that system found a set of plausible rules):

```
RULE 1
IF IS solution(V2N0),has(V2N0,V2N1),lhs(V2N1)
THAN SS solution(V2N0),has(V2N0,V2N1),lhs(V2N1)

RULE 2
IF IS solution(V2N0),has(V2N0,V2N1),integer(V2N1)
THAN SS solution(V2N0),has(V2N0,V2N1),integer(V2N1)

RULE 3
IF IS solution(V2N0),has(V2N0,V2N1),lhs(V2N1),has(V2N1,V2N2),
times(V2N2),has(V2N1,V2N3),identifier(V2N3),assocOp(V2N2,V2N3)
AND SS solution(V2N0),has(V2N0,V2N1),lhs(V2N1),has(V2N1,V2N2),
times(V2N2),has(V2N1,V2N3),identifier(V2N3),
THAN SS assocOp(V2N2,V2N3)

RULE 4
IF IS solution(V2N0),has(V2N0,V2N1),lhs(V2N1),
      has(V2N1,V2N2),minus(V2N2)
THAN SS solution(V2N0),has(V2N0,V2N1),lhs(V2N1),
        has(V2N1,V2N2),minus(V2N2)
OR SS solution(V2N0),has(V2N0,V2N1),rhs(V2N1),
      has(V2N1,V2N2),plus(V2N2)

RULE 5
IF IS solution(V2N0),has(V2N0,V2N1),rhs(V2N1),
      has(V2N1,V2N2),integer(V2N2),hasValue(V2N2,X)
THAN SS solution(V2N0),has(V2N0,V2N1),rhs(V2N1),
        has(V2N1,V2N2),integer(V2N2),hasValue(V2N2,X)
OR SS solution(V2N0),has(V2N0,V2N1),lhs(V2N1),
      has(V2N1,V2N2),integer(V2N2),hasValue(V2N2,X)
```

Rules 1 and 2 state that if an ideal solution has a left hand side or an integer than the student solution must have respectively a left hand side or an integer. The second rule is plausible, even if not always true, because it holds in all the given examples. Rule 3 states that if there are an integer and a times operator related by an “assocOp” relation in the ideal solution and there are also an integer and an operator in the student solution they have to be related by “assocOp” relation too.

Chapter 5

Building the tutor

In this chapter we describe a process for the construction of a tutor in certain domains of knowledge starting from the definition of an ontology describing the domain and we propose a semi-automatic procedure allowing the acquisition of rules used during tutoring process, exploiting the component Ruleminator described in previous chapter.

5.1 Introduction

5.2 Building the tutor

The construction of tutors occurs mainly defining ontology of the domain knowledge and providing examples of solving some problems that will be submitted to students during tutoring. In this process we make simplifications according to the tools that it has been decided to use but that does not constitute a restriction on the general framework: they can be replaced by developing tools that provide more complete and advanced functionality to generate rules allowing for a more accurate and complete tutor.

5.2.1 Defining the ontology and generating syntactich constraint

The ontology must be defined in order to derive parsing rules automatically that would recognize the different concepts and relations within the text given as a response. This fits in all domains in which the answers are structured or semi-structured like a programming language.

In defining ontology teacher must focus more on teaching than on completeness or correctness of the definition of the domain because generally is very difficult to define a domain exhaustively and a partial definition is sufficient if the teacher want to focus on particular aspect of domain.

That is why ontologies defined below are not exhaustive but tend to focus attention on the main concepts without pretending to model for the entire domain.

To create rules for parsing we have chosen to use the tool JavaCC [4] that is a software for generating LL(k) parser from a pseudo grammar defined in Java. Its functioning, like other similar tools, requires defining TOKEN for pattern matching and a BNF grammar. Since these types of parsers do not support the left recursion, we suppose that the definition of the ontology does not give rise to a grammar containing left recursion.

To map the ontology we have chosen to take a sub-set of possible constructs and have some predefined standard relation that the user can use to build ontology. The generated parser rapresents the syntactic constraints that the tutor uses during tutoring. Error messages generated by the parser can be customized to get better feedback than default.

The constructs available to the user are the following:

- Concepts;
- Relation “Is A” defining a taxonomy between concepts;
- Relation “Part Of” and “Unordered Part Of” defining a partonomy between concepts.

The relation “Part Of” here is intended as a relation defining an ordered belonging between components as opposed to “Unordered Part Of” that defines only membership without order. For each concept should be defined: its name, if it is a normal concepts, if it is a property (eg. An integer) or string (eg. does not take part to semantic constraint), when a concept is “terminal” (eg. with no concepts involved in relations “Part Of”) you must define a regular expression which serves to recognize the concept; for this purpose a library of common regular expressions such as INTEGER, IDENTIFIER etc.. is made available to the user. If the concept participates in a relation “Part Of” you can define a name for relationship between siblings. For each relation “Part of” you need to define the minimum and the maximum cardinality. Both are 1 by default. Each concept in the ontology represents a terminal or non-terminal symbol of grammar. The relation “Is A” results in a disjunctive productions while relations “Part Of” result in a conjunctive productions. The cardinality of relations “Part Of” is used to generate various type of operator: “[]”, “*”, “+”

5.2.2 Generating semantic constraint with Ruleminator

For the generation of rules with Ruleminator the text of the various solutions is processed by the parser generated with the ontology and through the tool JJTree is generated the corresponding abstract syntax tree (AST). Starting from AST and property associated with nodes the knowledge base files and language bias file are generated for the mining system (in this case FARMER).

Each node of AST can be decorated with the following properties:

- Name: is the name of the concept recognized in the text. For each concept is generated a new instance in the knowledge base
- SiblingRelation: if the concept is part of a "Ordered Part Of" relation this is the name of the relation that joins his next sibling node
- Property: if the concept is identified by a string in the text (such as a numeric value or an identifier) this variable contains the string. In this case the instance of the concept is the value of Property

Depending on the complexity of the representation of a solution, the system can generate a compact or expanded representation of the knowledge base. If the first option is selected every generated predicate contains all the children of corresponding nodes of the AST while in the second case for every child a different predicate is generated. In the first representation is easier to create clear rules for simple domains like first order equation algebra while for more complex domains such as C programming the second representation is best suited. This also involves a better system performance for mining task (having less predicate there will be dealt with shorter query).

The language bias is generated as a result of the chosen representation and the AST. Once the FARMER files are generated the mining procedure is launched and than the algorithm for extracting the rules is executed.

5.3 Example of Tutor

5.3.1 X-AlgebraTutor

With the ontology described in Figure 5.1 and applying the rules described in the previous section the following JavaCC grammar is obtained:

```
TOKEN : /* LITERALS */
{
  < INTEGER_LITERAL:
        <DECIMAL_LITERAL> (["l","L"])?
      | <HEX_LITERAL> (["l","L"])?
      | <OCTAL_LITERAL> (["l","L"])?
  >
|  < #DECIMAL_LITERAL: ["1"-"9"] (["0"-"9"])* >
|  < #HEX_LITERAL: "0" ["x","X"] (["0"-"9","a"-"f","A"-"F"])+ >
|  < #OCTAL_LITERAL: "0" (["0"-"7"])* >
```

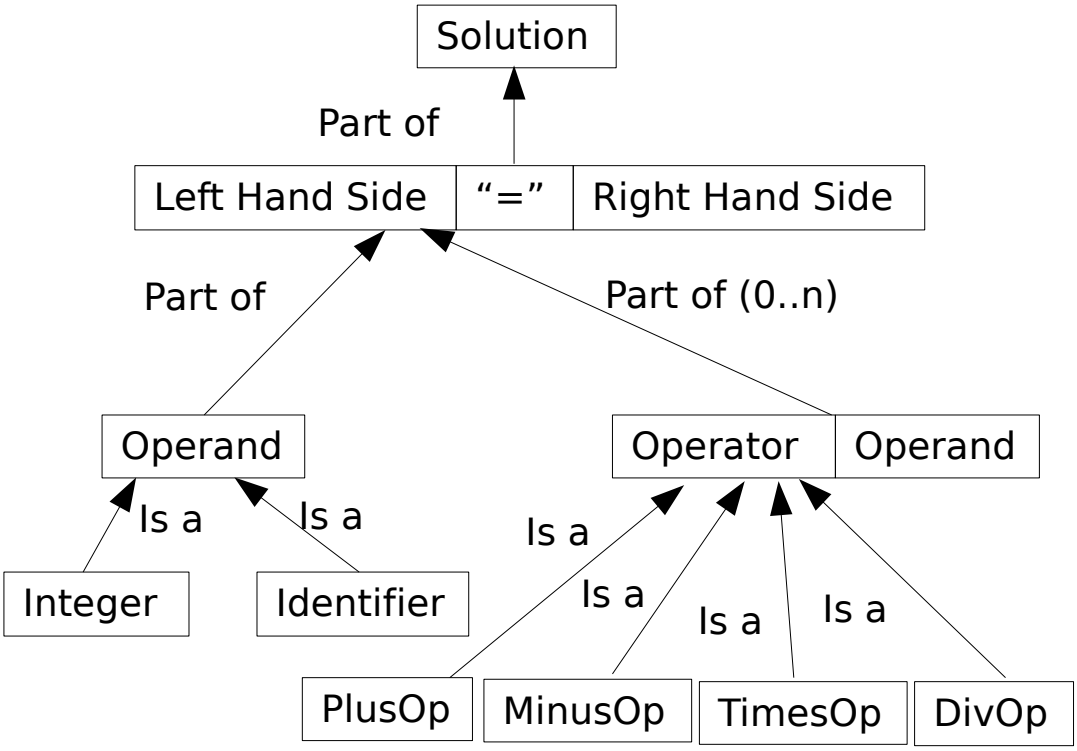

Figure 5.1: Ontology for domain of algebra equation

```
}
TOKEN : /* IDENTIFIERS */
{
  < IDENTIFIER: <LETTER> (<LETTER>|<DIGIT>)* >
|  < #LETTER: ["_","a"-"z","A"-"Z"] >
|  < #DIGIT: ["0"-"9"] >
}

SimpleNode Solution():{}
{
  LHS() "=" RHS() ";"
  { jjtThis.setName("solution");
   return jjtThis; }
}
void LHS(): {}
{
Operand() (Operator() Operand())*
{ jjtThis.setName("LHS"); }
}

void RHS(): {}
{
Operand() (Operator() Operand())*
{ jjtThis.setName("RHS"); }
}

void Operand() #void : {}
{
Integer() | Identifier()
}

void Integer(): {Token t;}
{
t = <INTEGER_LITERAL>
{  jjtThis.setProperty(t.image);
jjtThis.setName("Integer");
```

```
jjtThis.setSiblingRelation("assocOp"); }
}

void Identifier(): {Token t;}
{
t = <IDENTIFIER>
{  jjtThis.setProperty(t.image);
jjtThis.setName("Identifier");
jjtThis.setSiblingRelation("assocOp");}
}

void Operator() #void : {}
{
PlusOp() | MinusOp() | TimesOp() | DivOp()
}

void PlusOp(): {}
{
"+"
{
  jjtThis.setName("PlusOp");
  jjtThis.setSiblingRelation("assocOp");}
}

void MinusOp(): {}
{
"-"
{
  jjtThis.setName("MinusOp");
  jjtThis.setSiblingRelation("assocOp");}
}

void TimesOp(): {}
{
"*"
{
```

```
  jjtThis.setName("TimesOp");
  jjtThis.setSiblingRelation("assocOp");}
}

void DivOp(): {}
{
"/"
{
  jjtThis.setName("DivOp");
  jjtThis.setSiblingRelation("assocOp");}
}
```

If you give the following examples:

```
2*x-5=1;
2*x-1=5;
2*x=5+1;
```

The language bias created for Farmer is as follows:

```
showstyle(trie).
showconstantnames(true).
predicate(solution(Chiave,SOLUTION)).
predicate(haschildsolutionLHS(Chiave,SOLUTION,LHS)).
predicate(haschildLHSInteger(Chiave,LHS,INTEGER)).
predicate(haschildLHSTimesOp(Chiave,LHS,TIMESOP)).
predicate(assocOpIntegerTimesOp(Chiave,INTEGER,TIMESOP)).
predicate(haschildLHSIdentifier(Chiave,LHS,IDENTIFIER)).
predicate(assocOpTimesOpIdentifier(Chiave,TIMESOP,IDENTIFIER)).
predicate(haschildLHSMinusOp(Chiave,LHS,MINUSOP)).
predicate(assocOpIdentifierMinusOp(Chiave,IDENTIFIER,MINUSOP)).
predicate(assocOpMinusOpInteger(Chiave,MINUSOP,INTEGER)).
predicate(haschildsolutionRHS(Chiave,SOLUTION,RHS)).
predicate(haschildRHSInteger(Chiave,RHS,INTEGER)).
predicate(haschildRHSPlusOp(Chiave,RHS,PLUSOP)).
predicate(assocOpIntegerPlusOp(Chiave,INTEGER,PLUSOP)).
predicate(assocOpPlusOpInteger(Chiave,PLUSOP,INTEGER)).
predicate(simkey(Chiave)).
key(simkey(-)).
```

```
pkey(solution(Chiave,Solution)).
pkey(haschildsolutionLHS(Chiave,SOLUTION,LHS)).
pkey(haschildLHSInteger(Chiave,LHS,INTEGER)).
pkey(haschildLHSTimesOp(Chiave,LHS,TIMESOP)).
pkey(assocOpIntegerTimesOp(Chiave,_,TIMESOP)).
pkey(haschildLHSIdentifier(Chiave,LHS,IDENTIFIER)).
pkey(assocOpTimesOpIdentifier(Chiave,TIMESOP,_)).
pkey(haschildLHSMinusOp(Chiave,LHS,MINUSOP)).
pkey(assocOpIdentifierMinusOp(Chiave,_,MINUSOP)).
pkey(assocOpMinusOpInteger(Chiave,MINUSOP,_)).
pkey(haschildsolutionRHS(Chiave,SOLUTION,RHS)).
pkey(haschildRHSInteger(Chiave,RHS,INTEGER)).
pkey(haschildRHSPlusOp(Chiave,RHS,PLUSOP)).
pkey(assocOpIntegerPlusOp(Chiave,_,PLUSOP)).
pkey(assocOpPlusOpInteger(Chiave,PLUSOP,_)).
mode(solution(+,-)).
mode(haschildsolutionLHS(+1,+2,-3)).
mode(haschildLHSInteger(+1,+2,#)).
mode(haschildLHSTimesOp(+1,+2,-3)).
mode(assocOpIntegerTimesOp(+1,#,+)).
mode(haschildLHSIdentifier(+1,+2,#)).
mode(assocOpTimesOpIdentifier(+1,+,#)).
mode(haschildLHSMinusOp(+1,+2,-3)).
mode(assocOpIdentifierMinusOp(+1,#,+)).
mode(assocOpMinusOpInteger(+1,+,#)).
mode(haschildsolutionRHS(+1,+2,-3)).
mode(haschildRHSInteger(+1,+2,#)).
mode(haschildRHSPlusOp(+1,+2,-3)).
mode(assocOpIntegerPlusOp(+1,#,+)).
mode(assocOpPlusOpInteger(+1,+,#)).
minsup(1).
```

The knowledge base for Farmer is as follows:

```
solution(k0,k0solution1).
haschildsolutionLHS(k0,k0solution1,k0LHS1).
 haschildLHSInteger(k0,k0LHS1,c2).
 haschildLHSTimesOp(k0,k0LHS1,k0TimesOp1).
```

```
  assocOpIntegerTimesOp(k0,c2,k0TimesOp1).
  haschildLHSIdentifier(k0,k0LHS1,cx).
  assocOpTimesOpIdentifier(k0,k0TimesOp1,cx).
  haschildLHSMinusOp(k0,k0LHS1,k0MinusOp1).
  assocOpIdentifierMinusOp(k0,cx,k0MinusOp1).
  haschildLHSInteger(k0,k0LHS1,c5).
  assocOpMinusOpInteger(k0,k0MinusOp1,c5).
 haschildsolutionRHS(k0,k0solution1,k0RHS1).
  haschildRHSInteger(k0,k0RHS1,c1).

solution(k1,k1solution1).
 haschildsolutionLHS(k1,k1solution1,k1LHS2).
  haschildLHSInteger(k1,k1LHS2,c2).
  haschildLHSTimesOp(k1,k1LHS2,k1TimesOp2).
  assocOpIntegerTimesOp(k1,c2,k1TimesOp2).
  haschildLHSIdentifier(k1,k1LHS2,cx).
  assocOpTimesOpIdentifier(k1,k1TimesOp2,cx).
  haschildLHSMinusOp(k1,k1LHS2,k1MinusOp2).
  assocOpIdentifierMinusOp(k1,cx,k1MinusOp2).
  haschildLHSInteger(k1,k1LHS2,c1).
  assocOpMinusOpInteger(k1,k1MinusOp2,c1).
 haschildsolutionRHS(k1,k1solution1,k1RHS2).
  haschildRHSInteger(k1,k1RHS2,c5).

solution(k2,k2solution1).
 haschildsolutionLHS(k2,k2solution1,k2LHS3).
  haschildLHSInteger(k2,k2LHS3,c2).
  haschildLHSTimesOp(k2,k2LHS3,k2TimesOp3).
  assocOpIntegerTimesOp(k2,c2,k2TimesOp3).
  haschildLHSIdentifier(k2,k2LHS3,cx).
  assocOpTimesOpIdentifier(k2,k2TimesOp3,cx).
 haschildsolutionRHS(k2,k2solution1,k2RHS3).
  haschildRHSInteger(k2,k2RHS3,c5).
  haschildRHSPlusOp(k2,k2RHS3,k2PlusOp1).
  assocOpIntegerPlusOp(k2,c5,k2PlusOp1).
  haschildRHSInteger(k2,k2RHS3,c1).
```

```
  assocOpPlusOpInteger(k2,k2PlusOp1,c1).
```

By applying the algorithm Ruleminator the rules obtained are those of the previous chapter. While here it has taken another representation to automate the process for creating the parser, rules are substantially equivalent.

5.3.2 X-CTutor

In this section we want to develop a tutor on the C language that given some problems drives the student to their resolution. In figure 5.2 is shown part of an ontology that is the domain of language C. As mentioned previously, there are no pretensions to build a complete ontology that would be, for the current state of art, too complex for generating useful rules, but to take a more simple as possible the key concepts in the domain.

In this case, for example it is assumed that the student should write a program that makes the sum of an array of integers. In any program of this kind we suppose that there is a part dedicated to declaration of variables, a part for their eventual initialization and body of a solution that contains the rest of instructions.

The assignment statements are simplified for not taking a too complicated form for representation. In addition to the generation of syntactic constraints we specify to generate the compact representation for the knowledge base of Farmer.

The examples data as input to the system are as follows:

```
 int v[10];
int somma;
int i;
somma=0;
for(i=0;i<10;i++) {
  somma+=v[i];
}
% NEW EXAMPLE
int v[10];
int somma;
int i;
i=0;
somma=0;
while(i<10) {
  somma=somma+v[i];
  i++;
```

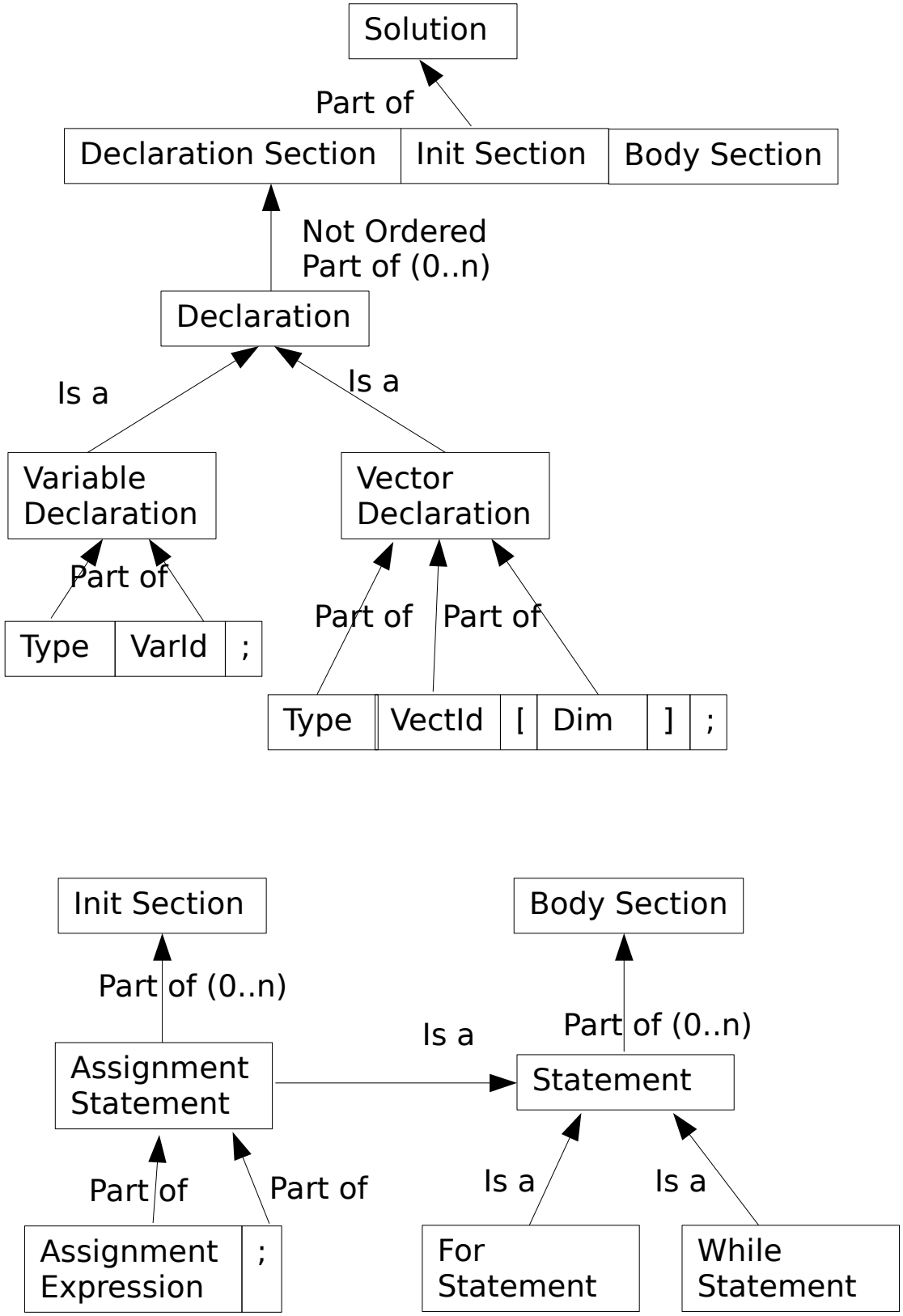

Figure 5.2: Part of Ontology for domain of C programming language

```
}
% NEW EXAMPLE
int v[10];
int somma;
int i;
i=9;
somma=0;
while(i>=0) {
  somma=somma+v[i];
  i--;
}
```

After running Ruleminator on these examples some of found rules are as follows (we transform the output for better readability):

```
IF IS has Init Section
THEN SS has Init Section

IF IS Declaration Section has "int i" variable
THEN SS Declaration Section has "int i" variable

IF IS Body Section has "while" statement
THEN SS Body Section has "while" statement
OR SS Body Section has "for" statement

IF IS stop condition of while is "i>=0"
THEN SS stop condition of while is "i>=0"
OR SS stop condition of while "i<=9"
```

One observation is that many rules that we expect are not generated. This is because Farmer does not support input as intentional knowledge base and therefore is not possible, for example, to add in the process of rules mining the necessary knowledge describing the hierarchy of concepts arising from the relations “Is A” defined in the ontology. For example, a possible missed rule can be “If there is an iterative statement in the solution then stop condition of the statement is either “i ¡10” or “i ¡= 9”. This rule can not be generated unless you explicitly specify the condition of arrest in the same type of construct. The absence of this type of declaration

```
IterativeStatement(X):-WhileStatement(X).
IterativeStatement(X):-ForStatement(X).
```

does not allow to generalize the rule. Other systems like WARMR allow to define intentional knowledge but the performance gap is such that it can not be used for a real application.

5.4 Quantitave and qualitative analisys of Ruleminator output

In this section will be made a comparative analysis of systems WARMR and FARMER integrated within the framework used to generate the rules. Also will be made quantitative and qualitative analysis to evaluate the system in the context of a real use.

The tests were made on the examples shown in the previous subsection: algebra equations domain and C programming domain restricted to some constructs. Tests carried out in two run in which the user provides ontologies shown above and three examples of possible resolution to problems in the domain. Factors considered were run time, quantity of generated rules and position of the most important rules in the system output. The computer where we performed the tests has a Pentium IV 2.8 GHz CPU and 512 Mbytes of RAM running Ubuntu Linux operating system.

An important observation is that exploiting in the best way the output provided by the systems is necessary to make changes to the source code in order to have all the data calculated during the process of mining without having to recalculate it later (eg support for every query). This was only possible with Farmer because distributed by the author as open source software while WARMR system is provided with data mining tools ACE [10] with no sources.

WARMR natively supports the inclusion of intensional knowledge in the mining process and it is therefore possible to elegantly specify rules in a separated file. Instead FARMER has no such support but with a trick you can simulate a simple intensional knowledge querying the extensional one and inserting the results in the input data without changing the original problem. A simple example of this process is shown below:

Estensional knowledge:

```
male(martin). female(lara). female(carol).
sister(lara,carol). son(martin,lara).
```

Intensional knowledge:

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

Explicitated intensional knowledge:

	WARMR	FARMER
Mining phase	7s	0.03s
Reading phase	5.6/6.3s	0.5/0.7s
Rule Generation phase	0.3	0.3

Table 5.1: Algebra data set without intensional knowledge

	WARMR	FARMER
Mining phase	19s	0.1s
Reading phase	19s	1.5s
Rule Generation phase	0.4	0.4

Table 5.2: Algebra data set with intensional knowledge for operator concept

```
male(martin). female(lara). female(carol).
sister(lara,carol). son(martin,lara).
person(martin). person(lara). person(carol).
```

The results are summarized in tables 5.1,5.2,5.3,5.4. The tables show the time taken from each phase of process for the two systems and for different initial data sets that are a problem on an algebra equation and a problem on computing the sum of an array of integer:

- Mining phase: this is the time needed for the two systems to complete the first phase of data mining
- Reading phase: time needed for reading output and the construction of the closed query lattice
- Rule Generation phase: time needed for creating rules

Results show the difference in performance of about two/three orders of magnitude between the two systems. In addition for the output provided by WARMR is necessary to recalculate the support for each query (although this is already calculated in the mining phase) because there is no option that will give it in output. This makes the process of reading phase extremely more

	WARMR	FARMER
Mining phase	61s	0.24s
Reading phase	18s/19s	0.8/2.5s
Rule Generation phase	0.1s	0.1s

Table 5.3: SumC data set without intensional knowledge

	WARMR	FARMER
Mining phase	1370s	2s
Reading phase	N/D	9s
Rule Generation phase	N/D	0.7s

Table 5.4: SumC data set with intensional knowledge for iterative statement concept

	Algebra DataSet	SumC DataSet
Number of rules	2852	764
Position within the most interesting rules	30	50

Table 5.5: Summary of number of rules generated and position within we found the most interesting rules

expensive than in FARMER case. With the source code of WARMR this problem would be easily solvable and is not a parameter to be considered in performance. The generation of rules takes the same time because the final output is essentially the same (both systems solve the same problem).

Table 5.5 summarize the quality of the Ruleminator. The rules are the same for both WARMR that FARMER and as can be seen from the tables many rules are generated even though the most interesting are in the top positions. The rules showed in previous subsection can be founded in the first 30 rules in the case of Algebra dataset and in the first 50 rules in the case of SumC dataset.

We report some of the generated output of Ruleminator and analyze in more detail the generated rules. For the Algebra data set in the very first positions we find the following rules:

```
IF IS solution(V0N0,V1N0),haschildsolutionRHS(V0N0,V1N0,V7N0)
THEN SS solution(V0N0,V1N0),haschildsolutionRHS(V0N0,V1N0,V7N0)

IF IS solution(V0N0,V1N0),haschildsolutionLHS(V0N0,V1N0,V2N0)
THEN SS solution(V0N0,V1N0),haschildsolutionLHS(V0N0,V1N0,V2N0)

IF IS solution(V0N0,V1N0),haschildsolutionLHS(V0N0,V1N0,V2N0),
haschildsolutionRHS(V0N0,V1N0,V7N0)
THEN SS solution(V0N0,V1N0),haschildsolutionLHS(V0N0,V1N0,V2N0),
haschildsolutionRHS(V0N0,V1N0,V7N0)

IF IS solution(V0N0,V1N0),haschildsolutionLHS(V0N0,V1N0,V2N0),
haschildLHSInteger(V0N0,V2N0,c2)
THEN SS solution(V0N0,V1N0),haschildsolutionLHS(V0N0,V1N0,V2N0),
```

```
haschildLHSInteger(V0N0,V2N0,c2)

IF IS solution(V0N0,V1N0),haschildsolutionLHS(V0N0,V1N0,V2N0),
haschildLHSIdentifier(V0N0,V2N0,cx)
THEN SS solution(V0N0,V1N0),haschildsolutionLHS(V0N0,V1N0,V2N0),
haschildLHSIdentifier(V0N0,V2N0,cx)

IF IS solution(V0N0,V1N0),haschildsolutionLHS(V0N0,V1N0,V2N0),
haschildLHSTimesOp(V0N0,V2N0,V4N0)
THEN SS solution(V0N0,V1N0),haschildsolutionLHS(V0N0,V1N0,V2N0),
haschildLHSTimesOp(V0N0,V2N0,V4N0)
```

Only the third rule is not relevant because can be deduced from the first and second. Ruleminator does not implement a reduction of produced rules so some of them can be unnecessary. The user can choose to keep the third rule instead of the first two. The other rules is correct only for the provided example set because the expression "2*x" is always present in the left hand side of equation. An important observation is that the first and second rules can be derived from the definition of ontology and is therefore redundant with respect to the syntactic constraints represented by the parser. In fact the definition of the ontology forces the presence of these two entities.

After that there are many rules that is correct only in the given example. For example:

```
IF IS solution(V0N0,V1N0),haschildsolutionLHS(V0N0,V1N0,V2N0),
haschildLHSTimesOp(V0N0,V2N0,V4N0),assocOpIntegerTimesOp(V0N0,c2,V4N0)
AND SS solution(V0N0,V1N0),haschildsolutionLHS(V0N0,V1N0,V2N0),
haschildLHSTimesOp(V0N0,V2N0,V4N0)
THEN SS solution(V0N0,V1N0),haschildsolutionLHS(V0N0,V1N0,V2N0),
haschildLHSTimesOp(V0N0,V2N0,V4N0),assocOpIntegerTimesOp(V0N0,c2,V4N0)

IF IS solution(V0N0,V1N0),haschildsolutionLHS(V0N0,V1N0,V2N0),
haschildLHSTimesOp(V0N0,V2N0,V4N0),assocOpTimesOpIdentifier(V0N0,V4N0,cx)
AND SS solution(V0N0,V1N0),haschildsolutionLHS(V0N0,V1N0,V2N0),
haschildLHSTimesOp(V0N0,V2N0,V4N0)
THEN SS solution(V0N0,V1N0),haschildsolutionLHS(V0N0,V1N0,V2N0),
haschildLHSTimesOp(V0N0,V2N0,V4N0),assocOpTimesOpIdentifier(V0N0,V4N0,cx)
```

They constrain the presence of the expression "2*x-" to be present in the left hand size.

At position 26 and 27 there are the following rules that are more significant:

```
IF IS solution(V0N0,V1N0),haschildsolutionLHS(V0N0,V1N0,V2N0),
haschildLHSMinusOp(V0N0,V2N0,V6N0)
THEN SS solution(V0N0,V1N0),haschildsolutionLHS(V0N0,V1N0,V2N0),
haschildLHSMinusOp(V0N0,V2N0,V6N0)
OR SS solution(V0N0,V1N0),haschildsolutionRHS(V0N0,V1N0,V7N0),
haschildRHSPlusOp(V0N0,V7N0,V8N0)

IF IS solution(V0N0,V1N0),haschildsolutionRHS(V0N0,V1N0,V7N0),
haschildRHSInteger(V0N0,V7N0,c5)
THEN SS solution(V0N0,V1N0),haschildsolutionRHS(V0N0,V1N0,V7N0),
haschildRHSInteger(V0N0,V7N0,c5)
OR SS solution(V0N0,V1N0),haschildsolutionLHS(V0N0,V1N0,V2N0),
haschildLHSInteger(V0N0,V2N0,c5)
```

The former says that if the left hand side of equation specified by the teacher has a minus operator then the solution given by the student must contain a minus operator in the left hand side or a plus operator in the right hand side. The latter rule can be generalized by substitute the constant "c5" with a variable because is valid in each of the given example. It says that if the right hand side of equation specified by the teacher has an integer X than the solution given by the student must contain an integer X in the right hand side or in the left hand side.

For the SumC data set the first generated rules are as following:

```
IF IS solution(V0N0,V1N0),haschildsolutionbodySection(V0N0,V1N0,V13N0)
THEN SS solution(V0N0,V1N0),haschildsolutionbodySection(V0N0,V1N0,V13N0)

IF IS solution(V0N0,V1N0),haschildsolutiondeclarationSection(V0N0,V1N0,V2N0)
THEN SS solution(V0N0,V1N0),haschildsolutiondeclarationSection(V0N0,V1N0,V2N0)

IF IS solution(V0N0,V1N0),haschildsolutioninitSection(V0N0,V1N0,V9N0)
THEN SS solution(V0N0,V1N0),haschildsolutioninitSection(V0N0,V1N0,V9N0)
```

These rules say that a solution must contain both "Declaration Section", "Init Section" and "Body Section". In this case the definition of ontology, unlike for algebra dataset, does not impose the presence of the three entities "Declaration Section", "Init Section" and "Body Section". In fact these rules are valid for the given examples beacause all contains the specified sections.

At position 7,8,9 there are rules that express the constraint of declaration for the variables used in the three examples (we report the rules in reduced form for better readability):

```
IF IS has int somma variable
THEN SS has int somma variable

IF IS has int v[10] array
THEN SS has int v[10] array

SE IS has int i variable
THEN SS has int i variable
```

At position 19 we can find the rules stating that if teacher solution has a while statement then student solution must have while statement or for statement

```
IF IS solution(V0N0,V1N0),haschildsolutionbodySection(V0N0,V1N0,V13N0),
haschildbodySectionwhileStatement(V0N0,V13N0,V14N0)
THEN SS solution(V0N0,V1N0),haschildsolutionbodySection(V0N0,V1N0,V13N0),
haschildbodySectionwhileStatement(V0N0,V13N0,V14N0)
OR SS solution(V0N0,V1N0),haschildsolutionbodySection(V0N0,V1N0,V13N0),
haschildbodySectionforStatement(V0N0,V13N0,V14N0)
```

Between the 9th and 19th position there are most redundant rules generated because of the lack of a reducer not yet implemented. As an example we report one of these rules. The other are similar.

```
IF IS solution(V0N0,V1N0),haschildsolutiondeclarationSection(V0N0,V1N0,V2N0),
haschilddeclarationSectionvariableDeclarationtypevarID(V0N0,V2N0,V7N0),
variableDeclarationtypevarID(V0N0,V7N0,cint,csomma),
haschildsolutionbodySection(V0N0,V1N0,V13N0)
THEN SS solution(V0N0,V1N0),haschildsolutiondeclarationSection(V0N0,V1N0,V2N0),
haschilddeclarationSectionvariableDeclarationtypevarID(V0N0,V2N0,V7N0),
variableDeclarationtypevarID(V0N0,V7N0,cint,csomma),
haschildsolutionbodySection(V0N0,V1N0,V13N0)
```

At position 32 we find the rule stating that the stop condition of while can be either “i>=0” or “i<=9”:

```
IF IS solution(V0N0,V1N0),haschildsolutionbodySection(V0N0,V1N0,V13N0),
haschildbodySectionwhileStatement(V0N0,V13N0,V14N0),
haschildwhileStatementconditionalExpressionvarIDconditionalOperatorconstant(V0N0,V14N0,V
conditionalExpressionvarIDconditionalOperatorconstant(V0N0,V15N0,ci,cget,c0)
THEN SS solution(V0N0,V1N0),haschildsolutionbodySection(V0N0,V1N0,V13N0),
```

```
haschildbodySectionwhileStatement(V0N0,V13N0,V14N0),
haschildwhileStatementconditionalExpressionvarIDconditionalOperatorconstant(V0N0,V14N0,V
conditionalExpressionvarIDconditionalOperatorconstant(V0N0,V15N0,ci,cget,c0)
OR SS solution(V0N0,V1N0),haschildsolutionbodySection(V0N0,V1N0,V13N0),
haschildbodySectionwhileStatement(V0N0,V13N0,V14N0),
haschildwhileStatementconditionalExpressionvarIDconditionalOperatorconstant(V0N0,V14N0,V
conditionalExpressionvarIDconditionalOperatorconstant(V0N0,V15N0,ci,clet,c9)
```

Between 19th and 32th position most rules are not significant.

Immediately after the position 50 we can find some other interesting rules. For example:

```
IF IS solution(V0N0,V1N0),haschildsolutionbodySection(V0N0,V1N0,V13N0),
haschildbodySectionwhileStatement(V0N0,V13N0,V14N0),
haschildwhileStatementcompoundStatement(V0N0,V14N0,V19N0),
haschildcompoundStatementassignmentExpressionvarIDunaryOperator(V0N0,V19N0,V17N0),
assignmentExpressionvarIDunaryOperator(V0N0,V17N0,ci,cdecOp)
THEN SS solution(V0N0,V1N0),haschildsolutionbodySection(V0N0,V1N0,V13N0),
haschildbodySectionwhileStatement(V0N0,V13N0,V14N0),
haschildwhileStatementcompoundStatement(V0N0,V14N0,V19N0),
haschildcompoundStatementassignmentExpressionvarIDunaryOperator(V0N0,V19N0,V17N0),
assignmentExpressionvarIDunaryOperator(V0N0,V17N0,ci,cdecOp)
OR SS solution(V0N0,V1N0),haschildsolutionbodySection(V0N0,V1N0,V13N0),
haschildbodySectionwhileStatement(V0N0,V13N0,V14N0),
haschildwhileStatementcompoundStatement(V0N0,V14N0,V19N0),
haschildcompoundStatementassignmentExpressionvarIDunaryOperator(V0N0,V19N0,V17N0),
assignmentExpressionvarIDunaryOperator(V0N0,V17N0,ci,cincOp)
```

stating that if the teacher solution has a while statement with an “i–” instruction then the student solution must have a while statement with “i–” or “i++” instruction.

From the tests illustrated above it can be seen that many of the interesting rules can be found in top positions. One improvement to add is control over redundant rules which constitute a significant number of total. The heuristics used to sort the rules is based on the length of the rule so that the shorter rules (that in the case of restriction to object identity the more general) are at head of the list. This makes it difficult to access the semantically more complicated rules which require a higher number of predicates to be enunciated. It is therefore necessary to introduce a heuristic that is somehow able to recognize the longer rules which have a greater importance than others who, despite being more general (shorter), are not relevant to the description of the domain. The algorithm, however can also be used in complex domains like the C programming

after adding a component that transforms the outputted rules in natural language (adding sugar syntax is quite simple because the name of concepts and relationships can be derived from the predicate name of the rules). By adding a filter to eliminate redundant rules should lead to a substantial reduction in output.

Conclusions and future work

The use of Intelligent Tutoring System in recent years has proved effective in improving the learning experience in various domains of knowledge. The technologies for distance learning spread ever faster, but giving little weight to the system "intelligence" that allows to develop a real interoperability with user. The ITS can fill the GAP but they are extremely difficult to create and maintain because many skills are required for their development.

In this work we have presented an architecture that is proposed as a basis for the development of advanced tools for learning and we implemented the part that represents the heart of the creation of an ITS. The acquisition of knowledge must be done in a transparent way for the user who builds these systems and the procedure explained partly solves the problem. In addition, the system is based on theoretical results and robust tools so it can offer very promising perspectives for future development. Every part of architecture can be developed as a separated field of research giving room for improvements that will enhance the system.

The system is now limited to work on texts in free form that can be recognized with a parser generated automatically from the ontology described by the user. This involves the use of this system on structured or semi-structured domains preventing the use on domains in which problems can be solved using natural language. In particular it would be very interesting to develop an adaptive information extraction component that can be trained to extract entity and relations between them.

As an example it could be adopted the annotation style of GATE [3], a framework for natural language processing which comes with a built-in information extraction system called ANNIE. In GATE an annotation is substantially equivalent to an XML tag with a name and a set of attributes. For example if we have the following text:

```
A piece of text
```

a possible annotation can be:

```
<sentence numberOfWord=4 language=''English''> A piece of
text</sentence>
```

The tag name corresponds to the annotation type and the list of attributes corresponds to the so called feature map in GATE. Each annotation is represented as an object containing an ID, the type of annotation, its feature map and two pointers to the document representing the start and end offset of annotation. By means of ANNIE tools is possible to annotate some kind of document specifying many parameters.

Building an annotation mechanism from scratch is a very onerous task. An interesting opportunity of GATE is the possible integration with a various machine learning framework that can learn annotating documents starting from a set of annotated ones.

Adaptive information extraction is the ability of adapt the Information Extraction (IE) system depending on the domain that system is analyzing. The MELITA system [5] is an annotation interface, which is based on adaptive IE system AMILCARE, that allows training an IE system specifying the annotation schema as an ontology and interacting with the system. The user first defines ontology, specifically a hierarchy of classes that corresponds to the annotation types, a set of documents and than starts to manually annotate the documents. The learning engine runs in background and support the user annotation task. This type of mechanism can be used in the architecture depicted in figure 3.1 to train the Information Extraction module.

The system for the knowledge acquisition is limited by the capabilities and performance of systems for data mining and possible future developments should cover the improvement of these facilities. In particular, the use of FARMER does not allow an elegant use of intensional knowledge and does not output the complete DAG of queries. The first improvements would lead to greater expressiveness of the found rules while the second would improve the overall performance of the system.

Finally an interface that allows to integrate and use all the tools developed should have priority over possible developments of the instrument with particular emphasis in the management of the system knowledge base.

References

[1] Act-r a cognitive theory about human cognition. http://act-r.psy.cmu.edu/.

[2] Circsim-tutor project. http://www.cs.iit.edu/~ circsim/.

[3] Gate: General architecture for text engineering. http://gate.ac.uk/.

[4] Javacc: Java compiler compiler. https://javacc.dev.java.net/.

[5] Melita: An annotation interface that uses adaptive information extraction from texts. http://nlp.shef.ac.uk/melita/.

[6] tuprolog: a light-weight prolog engine. http://www.alice.unibo.it/tuProlog.

[7] AGRAWAL, R., MANNILA, H., SRIKANT, R., TOIVONEN, H., AND VERKAMO, A. I. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996, pp. 307–328.

[8] A.N., K. Model-based reasoning for domain modeling in a web-based intelligent tutoring system to help students learn to debug c++ programs. In *Intelligent Tutoring Systems* (Biarritz, France, June 2002), vol. 2363 of *LNCS*, pp. 792–801.

[9] ANDERSON, J. R., AND REISER, B. The lisp tutor. *Byte 10*, 4 (1985), 159–175.

[10] BLOCKEEL, H., DEHASPE, L., AND RAMON, J. The ace data mining system. http://www.cs.kuleuven.ac.be/ dtai/ACE/.

[11] BUITELAAR, P., OLEJNIK, D., AND SINTEK, M. Ontolt: A protégé plug-in for ontology extraction from text. In *2nd International Semantic Web Conference: Demo Session* (Sanibel Island, Florida, USA, 2003).

[12] CORDELLA, L. P., FOGGIA, P., SANSONE, C., AND VENTO, M. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans. Pattern Anal. Mach. Intell. 26*, 10 (2004), 1367–1372.

[13] DEHASPE, L., AND RAEDT, L. D. Mining association rules in multiple relations. In *ILP* (Prague, Czech Republic, 1997), pp. 125–132.

[14] DEHASPE, L., AND TOIVONEN, H. Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery 3*, 1 (199), 7–36.

[15] DZEROSKI, S. Multi-relational data mining: an introduction. *SIGKDD Explorations 5*, 1 (2003), 1–16.

[16] FAURE, D., NDELLEC, C., AND ROUVEIROL, C. Acquisition of semantic knowledge using machine learning methods: The system asium. Tech. Rep. ICS-TR-88-16, Laboratoire de Recherche en Informatique,Inference and Learning Group, Universite Paris, Sud, 1998.

[17] FERILLI, S., FANIZZI, N., MAURO, N. D., AND BASILE, T. M. A. Efficient θ-subsumption under object identity. In *AI*IA Workshop su Apprendimento Automatico: Metodi e Applicazioni dell'Ottavo Convegno della Associazione Italiana per l'Intelligenza Artificiale* (Siena, Italy, 2002), pp. 59–68.

[18] HONG, J. Guided programming and automated error analysis in an intelligent prolog tutor. *Int. J. Hum.-Comput. Stud. 61*, 4 (2004), 505–534.

[19] JAMESON, A. Numerical uncertainty management in user and student modeling: An overview of systems and issues. *User Model. User-Adapt. Interact. 5*, 4 (1995), 193–251.

[20] JOHNSON, W. L., AND SOLOWAY, E. Proust: Knowledge-based program understanding. *IEEE Transactions on Software Engineering 11*, 3 (1985), 267–275.

[21] JUN HUAN, WEI WANG, J. P. Efficient mining of frequent subgraph in the presence of isomorphism. In *Internation Conference on Data Mining* (Melbourne, Florida, USA, 2003), pp. 549–552.

[22] KOEDINGER, K. R., ALEVEN, V., AND HEFFERNAN, N. T. Toward a rapid development environment for cognitive tutors. In *12th Annual Conference on Behavior Representation in Modeling and Simulation. Simulation Interoperability Standards Organization* (Phoenix, Arizona, USA, 2003).

[23] KOEDINGER, K. R., AND ANDERSON, J. R. Intelligent tutoring goes to the big city. *International Journal of Artificial Intelligence in Education 8* (1997), 30–43.

[24] KURAMOCHI, M., AND KARYPIS, G. Frequent subgraph discovery. In *International Conference on Data Mining* (San Jose, California, USA, 2001), pp. 313–320.

[25] LAVRAČ, N., AND DŽEROSKI, S. *Inductive Logic Programing: Techniques and Applications*. Ellis Horwood, New York, 1994.

[26] LISI, F. A., FERILLI, S., AND FANIZZI, N. Object identity as search bias for pattern spaces. In *15th Eureopean Conference on Artificial Intelligence* (Lyon, France, 2002), pp. 375–379.

[27] MAEDCHE, A., AND STAAB, S. Semi-automatic engineering of ontologies from text. In *12th International Conference on Software Engineering and Knowledge Engineering* (Chicago, Illinois, USA, 2000), pp. 231–239.

[28] MARTIN, B. Constraint-based modelling: Representing student knowledge. *New Zealand Journal of Computing 7*, 2 (1999), 30–38.

[29] MARTIN, B., AND MITROVIC, A. Induction of higher-order knowledge in constraint-based models. In *ITS2000 workshop on applying Machine Learning to ITS Design/Construction* (Montreal, 2000), J. Beck, Ed., pp. 31–36.

[30] MARTIN, B., AND MITROVIC, A. Tailoring feedback by correcting student answers. In *Lecture Notes in Computer Science*, vol. 1839. Springer-Verlag Heidelberg, 2000, pp. 383–392.

[31] MARTIN, B., AND MITROVIC, A. Automatic problem generation in constraint-based tutors. In *Intelligent Tutoring Systems* (Biarritz, France, 2002), pp. 388–398.

[32] MARTIN, B., AND MITROVIC, A. Wetas: A web-based authoring system for constraint-based its. In *Adaptive Hypermedia and Adaptive Web-Based Systems, Second International Conference* (Malaga, Spain, 2002), pp. 543–546.

[33] MITROVIC, A., MAYO, M., SURAWEERA, P., AND MARTIN, B. Constraint-based tutors: A success story. In *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems* (Budapest, Hungary, 2001), pp. 931–940.

[34] MUNRO, A., JOHNSON, M. C., PIZZINI, Q. A., D. S. SURMON, D. M. T., AND WOGULIS, J. L. Authoring simulation-centred tutors with rides. *International Journal of Artificial Intelligence in Education 8* (1997), 284–316.

[35] NAVIGLI, R., VELARDI, P., AND GANGEMI, A. Ontology learning and its application to automated terminology translation. *IEEE Intelligent Systems 18*, 1 (2003), 22–31.

[36] NIJSSEN, S., AND KOK, J. N. Efficient frequent query discovery in farmer. In *7th European Conference on Principles and Practice of Knowledge Discovery in Databases* (Cavtat-Dubrovnik, Croatia, 2003), pp. 350–362.

[37] NIJSSEN, S., AND KOK, J. N. The gaston tool for frequent subgraph mining. In *International Workshop on Graph-Based Tools* (Rome, Italy, October 2004), Elsevier, pp. 77–87.

[38] NIJSSEN, S., AND KOK, J. N. A quickstart in frequent structure mining can make a difference. In *10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Seattle, Washington, USA, 2004), pp. 647–652.

[39] QIU, L., AND RIESBECK, C. An incremental model for developing computer-based learning environments for problem-based learning. In *ICALT* (Finland, August 2004).

[40] RICCUCCI, S., CARBONARO, A., AND CASADEI, G. An architecture for knowledge management in intelligent tutoring system. In *Cognition and Exploratory Learning in Digital Age* (Porto, Portugal, December 2005), pp. 473–476.

[41] RICCUCCI, S., CARBONARO, A., AND CASADEI, G. A framewok for knowledge acquisition in intelligent tutoring systems. In *Informatics in Secondary Schools: Evolution and Perspectives* (Vilnius, Lithuania, November 2006), pp. 128–136.

[42] RICCUCCI, S., CARBONARO, A., AND CASADEI, G. A framewok for knowledge acquisition in intelligent tutoring systems. In *Congresso annuale AICA* (Cesena, Italy, September 2006).

[43] RICCUCCI, S., CARBONARO, A., AND CASADEI, G. Knowledge acquisition in intelligent tutoring system: A data mining approach. In *Lecture Notes in Computer Science*, vol. 4827. Springer-Verlag Heidelberg, 2007, pp. 1195–1205.

[44] SILVERMAN, B. G. Survey of expert critiquing systems: Practical and theoretical frontiers. *Commun. ACM 35*, 4 (1992), 106–127.

[45] SISON, R., AND SHIMURA, M. Student modeling and machine learning. *International Journal of Artificial Intelligence in Education 9* (1998), 128–158.

[46] SOLOWAY, E., RUBIN, E., WOOLF, B., JOHNSON, W. L., AND BONAR, J. Meno ii: An ai-based programming tutor. *Journal of Computer-Based Instruction 10* (1983), 20–34.

[47] SONG, J. S., HAHN, S. H., TAK, K. Y., AND KIM, J. H. An intelligent tutoring system for introductory c language course. *Computers & Education 28*, 2 (1997), 93–102.

[48] SRINIVASAN, A. Aleph: A learning engine for proposing hypotheses. http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph.

[49] STOLLE, C., KARWATH, A., AND RAEDT, L. D. Classiccl: An integrated ilp system. In *Lecture Notes in Computer Science*, vol. 3735. Springer Berlin / Heidelberg, 2005, pp. 354–362.

[50] STUMME, G. Iceberg query lattices for datalog. In *International Conference on Computational Science* (Kraków, Poland, 2004), pp. 109–125.

[51] SURAWEERA, P., MITROVIC, A., AND MARTIN, B. The role of domain ontology in knowledge acquisition for itss. In *Intelligent Tutoring Systems* (Brazil, September 2004), pp. 207–216.

[52] SURAWEERA, P., MITROVIC, A., AND MARTIN, B. The use of ontologies in its domain knowledge authoring. In *Workshop on Applications of Semantic Web for E-learning* (Maceio, Brazil, September 2004), pp. 41–49.

[53] SURAWEERA, P., MITROVIC, A., AND MARTIN, B. A knowledge acquisition system for constraint-based intelligent tutoring systems. In *Artificial Intelligence in Education* (Amsterdam, Nederland, 2005), pp. 638–646.

[54] SYKES, E. R., AND FRANEK, F. A prototype for an intelligent tutoring system for students learning to program in java. In *CATE2003, IASTED International Conference on Computers and Advanced Technology in Education* (Rhodes, Greece, June 2003), pp. 78–83.

[55] TURMO, J., AGENO, A., AND CATALÀ, N. Adaptive information extraction. *ACM Comput. Surv. 38*, 2 (2006).

[56] WEBER, G., AND MLLENBERG, A. Elm-programming-environment: A tutoring system for lisp beginners. In *Cognition and Computer Programming*, Ablex, Ed. Norwood, NJ, 1995, pp. 373–408.

[57] WEBER, G., AND SPECHT, M. User modeling and adaptive navigation support in www-based tutoring systems. In *6th International Conference on User Modeling* (Sardinia, Italy, 1997), pp. 289–300.

[58] WÖRLEIN, M., MEINL, T., FISCHER, I., AND PHILIPPSEN, M. A quantitative comparison of the subgraph miners mofa, gspan, ffsm, and gaston. In *9th European Conference on Principles and Practice of Knowledge Discovery in Databases* (Porto, Portugal, 2005), pp. 392–403.

[59] YAN, X., AND HAN, J. gspan: Graph-based substructure pattern mining. In *International Conference on Data Mining* (Maebashi City, Japan, 2002), pp. 721–724.