Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

DOCTORAL THESIS

February 2012

Ing. Monika Zemenová (Žáková)

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

Exploiting ontologies and higher order knowledge in relational data mining

Doctoral Thesis

Monika Zemenová (Žáková)

Prague, February 2012

Ph.D. Programme: Electrical Engineering and Informatics
Branch of study: Artificial Intelligence and Biocybernetics

Supervisor: Doc. Ing. Filip Železný, Ph.D.

I would like to thank to my supervisor Doc. Filip Železný for the opportunity to work in the research in data mining and for his guidance.

I would also like to express my gratitude to Prof. Nada Lavrač for her invaluable advice and encouragement and to my colleagues Petr Křemen and Vid Podpečan.

My thanks also belong to my family and friends for their love and support.

Abstract

Present day knowledge discovery tasks require mining heterogeneous and structured data and knowledge sources. The key enabling factors for performing these tasks include efficient exploitation of knowledge about the domain of discovery and utilizing meta knowledge about the data mining process, which facilitates the construction of complex workflows consisting of highly specialized algorithms.

In this thesis we first propose a framework for relational data mining with taxonomic domain knowledge. The proposed framework is based on inductive logic programming and enables efficient handling of taxonomies on concepts and predicates by means of a specialized refinement operator. The operator is used to generate first order features from relational data transforming a relational learning problem into a propositional representation on which classical data mining algorithms can be applied. Since the generated features form a subsumption hierarchy, we have also enhanced two rule learning algorithms to exploit this hierarchy to produce more compact rules.

Applying relational data mining to real life problems is an intricate process consisting of multiple interweaving inductive, deductive and transformation algorithms. To alleviate the complexity of this task, we have developed a methodology for automatic construction of knowledge discovery workflows.

Our methodology consists of two main ingredients. The first is a formal conceptualization of knowledge types and algorithms implemented in a knowledge discovery ontology following up on state-of-the-art developments of a unified data mining theory. The developed ontology was used to annotate our algorithms for relational data mining and algorithms available in the Orange4WS data mining platform. The ontology also contains means for annotating workflows, thus facilitating their execution and reuse.

Secondly, a planning algorithm was implemented and employed to assemble workflows for the task specified by the user's input-output task requirements. We have developed two variants of the algorithm. PDDLPlanner uses a standard planning formalism PDDL and therefore requires converting descriptions available in the ontology into PDDL, whereas HierarchyPlanner is based on directly querying the ontology and thus is capable of exploiting the algorithms hierarchy.

We have also developed a prototype integration of the proposed methodology into the Orange4WS data mining platform providing means for an interactive user environment for workflow execution.

Abstrakt

Při řešení mnoha úloh v oboru získávání znalostí z dat je v dnešní době třeba pracovat s různorodými a strukturovanými zdroji dat a znalostí. Klíčovou roli při tom hraje především efektivní využití znalostí ze zkoumaného oblasti a využití meta-znalostí o samotném dolování dat. Tyto meta-znalosti usnadňují sestavování specializovaných algoritmů do komplexních procesů.

Tato práce se nejdříve zabývá návrhem systému pro relační dolování dat s taxonomickou doménovou znalostí. Systém je založen na induktivním logickém programování a umožňuje efektivní zacházení s taxonomiemi konceptů a predikátů pomocí nového specializačního operátoru. Tento operátor je použit pro generování rysů z relačních dat během transformace problému relačního učení do propoziční reprezentace. Na takto transformovaná data lze pak použít klasické algoritmy dolování dat. Vzhledem k tomu, že rysy vygenerované tímto způsobem tvoří subsumpční hierarchii, rozšířili jsme také dva algoritmy pro učení klasifikačních a deskripčních pravidel tak, že využívají tuto hierarchii a díky tomu vytvářejí kompaktnější pravidla.

Použití metod relačního učení na problémy v praxi vede k sestavování složitého procesu, který je složen z mnohočetných transformací a induktivních i deduktivních algoritmů. Proto jsme navrhli metodiku automatického vytváření procesů pro získávání znalostí z dat.

Naše metodika má dvě základní součásti. První z nich je formální reprezentace typů znalostí a algoritmů implementovaná jako ontologie, která navazuje na aktuální vývoj jednotné teorie dolování dat. Vytvořená ontologie byla použita k anotaci našich algoritmů relačního učení a algoritmů, které jsou součástí platformy pro dolování dat Orange4WS. Ontologie také obsahuje pojmy pro anotování procesů a tím usnadňuje jejich provedení a opětovné použití.

Druhou součástí metodiky je plánovací algoritmus, který byl implementován a použit pro sestavení procesů řešících úlohu specifikovanou vstupně/výstupními požadavky uživatele. Vytvořili jsme dvě varianty plánovacího algoritmu. PDDLPlanner využívá standardní formalismus pro plánovací úlohy PDDL a je tedy nutné převést popis úlohy a algoritmů z jazyka ontologie do PDDL. HierarchyPlanner je založen na přímém dotazování ontologie a je tak schopen využít hierarchie algoritmů.

Navrženou metodiku jsme integrovali do platformy pro dolování dat Orange4WS, která poskytuje prostředky interaktivního uživatelského rozhraní pro spouštění sestavených procesů.

Contents

Chapter 1

Introduction

Integration of heterogeneous data sources and inferring new knowledge from such combined information is one of the key challenges in the highly knowledge intensive domains such as present-day life sciences and engineering. Consider for example bioinformatics where for virtually any biological entity (e.g., a gene) vast amounts of relevant background information are available from public web resources.

A significant part of this information has rich relational structure and includes background knowledge at different levels of abstraction e.g. in form of ontologies. Therefore there has been a significant amount of effort to develop algorithms capable of efficiently handling relational data and complex background knowledge. One of the promising technologies utilized in this effort is inductive logic programming (ILP). ILP has already proven abilities for handling relational data, produces rules easily interpretable by users. The challenges for ILP in context of rich hierarchical background knowledge are the capability to handle structures of sizes relevant for a particular task and the ability to produce compact rules.

The data comes also in various other forms such as graphs, sequences and texts. A principled fusion of the relevant data requires the interplay of diverse specialized algorithms resulting in highly intricate workflows. While the mutual relations of such algorithms and principles of their applicability may be mastered by computer scientists, their command cannot be expected from the end user, e.g., a life scientist or a product engineer. A formal capture of this knowledge is thus needed, e.g., in the form of ontologies of relevant services and knowledge/data types, to serve as a basis for intelligent computational support of knowledge discovery workflow composition. A formal representation of the knowledge discovery task can also be used to improve repeatability of experiments, enable reasoning on the results and facilitate reuse of workflows and results.

1.1 Main concepts

In life sciences and engineering vast amounts of knowledge and data are currently available giving rise to the following main challenges: effective reuse of existing knowledge, inferring new knowledge and sharing this newly discovered knowledge. There are two key concepts that are instrumental in responding to these challenges and are in the center of the thesis: knowledge discovery and ontologies.

1.1.1 Knowledge discovery

Knowledge discovery is the process of inferring new knowledge via induction. In the classical formulation new knowledge is being inferred from data, which are represented as fixed feature vectors or attribute-value pairs. In real life applications the examined data often describes objects of more than one kind stored in multiple interlinked tables in a relational database or in a graph. Information relevant for the knowledge discovery task is contained both in properties of the individual objects and in the relations between them. These tasks often cannot be handled efficiently with classical learning techniques. An example task could be to analyze what students are likely to get a job in a research and development company. The data would include information about students, teachers, courses, projects etc. Real life examples of such tasks include chemical compound analysis and website analysis [25].

Relational learning is knowledge discovery from data containing information about several different types of objects with relations between them. The input data are usually represented in a formalism based on first order logic. These more expressive knowledge representation formalisms enable learning not only from data, but also from additional knowledge about the domain in form of rules or constraints that hold in the given domain. An example of such rule could be the student - lecturer relationship inferred from student - course and course - lecturer relationships.

An important framework for relational learning, which contributed significantly to its theoretical foundations, is inductive logic programming (ILP). ILP offers means for dealing with structured data and background knowledge and produces results in form of rules, which are easily interpretable by humans and also in formal representation and thus can be easily added to the existing body of knowledge. Therefore ILP is a framework which will be further examined in detail in this thesis.

There are two general approaches to ILP. The first one is searching the space of first order logic hypotheses directly. The second one is transforming a relational representation of a learning problem into a propositional (attribute-value) representation (this process is known as propositionalization) and subsequently using a propositional learner. An attribute or feature created during propositionalization is essentially a pattern frequently occurring in the data and fulfilling some additional constraints. An example feature would be a student participation in at least one project involving partners from the industry.

The propositionalization approach is more flexible since it separates the process of transformation of data from relational to classical propositional representation and the actual model induction. Thus a wide range of already available propositional algorithms can be used.

1.1.2 Ontologies

Ontology is a term borrowed from philosophy, where to first approximation it is a study of what is and what the most general features and relations of the entities are [50]. In the knowledge representation realm an ontology is a set of concept definitions and relations between the concepts. It can be used to define what entities exist and also what entities may exist within a domain. In the university domain mentioned earlier concepts will include students, advisors, courses, projects, companies, etc. and relations between them will include participation of a company in a project, teaching a course, attending a course, etc.

An important relation used in ontologies is expressing that one concept is more general than another, e.g., that crystallography course is a special case of solid state physics course. This relation allows to build concept hierarchies and reason about the domain at different levels of generality.

1.2 Problems to be solved

Recently there has been a lot of effort to formalize knowledge available for example in bioinformatics, using ontologies and a lot of research on ontology formalisms, which lead to establishing standard ontologies and formalisms.

Therefore time is ripe for examining the possibilities of effective interplay of ontologies and knowledge discovery. There are essentially three types of this interplay:

- using ontologies as domain knowledge in knowledge discovery
- using data mining to construct/validate ontologies
- using ontologies to formalize knowledge discovery task, data and results

The work presented in this thesis started by examining the first type and the conducted research lead to development of a framework, which requires application of several specialized algorithms. The developed framework is quite complex and therefore investigation of possibilities of its deployment, extensibility and reuse naturally motivated research of the third type of interplay.

To distinguish between knowledge about the domain of discovery, for example biology, and knowledge about the discovery task and its ingredients, we a have decided to follow the distinction between data and meta data. Therefore we use the term *domain knowledge* for knowledge about the domain of discovery and the term *meta knowledge* for knowledge about the discovery process itself.

1.2.1 Exploiting domain knowledge

Inductive logic programming has been considerably successful in various knowledge discovery problems such as in bioinformatics [57] and finite element mesh design [31]. Its techniques provide means for using domain knowledge in the form of predicates. However standard ILP techniques do not utilize information about hierarchies of concepts and predicates and other information contained in ontologies.

During propositionalization by relational features construction a large number of features is constructed. Most often used method of feature generation is top to bottom approach starting with the most general features and refining them into more specialized ones. Features constructed in this way form a hierarchy. Utilizing the information about feature hierarchy in propositional search is expected to increase efficiency of the search and increase quality of propositional rules e.g. by preventing conjunction of subsumer with its subsumee.

The main tasks related to exploiting domain knowledge can be summarized as follows:

- *Adapting ILP framework to exploit higher order domain knowledge*
  Propose an extension of standard ILP framework focusing especially on efficient handling of concept and predicate hierarchies defined by means of ontologies.

- *Propositional search using feature subsumption*
  Examine utilizing the feature subsumption in propositional search with respect to efficiency of the search and increase quality of propositional rules.

1.2.2 Exploiting meta knowledge

Today's distributed, heterogeneous, yet instantly accessible wealth of domain information in many branches of science and engineering can only be harnessed through careful orchestration of algorithms dedicated to such various sources of information. It is however the diversity that makes the configuration of the optimal computational/information flow a hard combinatorial task. Therefore, to facilitate performing of these complex knowledge discovery tasks, methods for some level of automation of the knowledge discovery workflows composition are required.

When workflow construction is performed manually by humans it requires insight into types of algorithms used, their inputs and parameters and applicability to particular types of data. For efficient support of knowledge discovery workflow composition this meta knowledge about the algorithms needs to be formalized and used to guide the workflow construction. Formal description of the whole knowledge discovery task including performed experiments and their results could also improve repeatability of experiments and enable reuse of constructed workflows or their parts and the discovered results.

Both these goals can be elegantly achieved by ontology or ontologies formalizing important concepts of the knowledge discovery task such as learning algorithms, available data and knowledge types including results of learning such as patterns or rules. The main tasks related to exploiting meta knowledge can therefore be summarized as follows:

- *Develop an ontology of the knowledge discovery domain*
  The ontology should describe data, algorithms and results of knowledge discovery tasks, in a way that allows for intelligent retrieval of results, composition of workflows from algorithms, storing information about experiments, reuses existing ontologies and semantic web standards. The ontology should cover complex relational learning with domain knowledge.

- *Design an algorithm for automatic workflow generation*
  Design algorithm for automatic workflow composition utilizing information contained in the knowledge discovery ontology and producing non-linear workflows with multiple inputs and outputs.

1.3 Key thesis contributions

The work in Part I focuses on enhancing relational learning to exploit domain knowledge in form of taxonomies. We adopted the approach of solving relational learning task through propositionalization. We enhanced both steps of the propositionalization procedure. We adapted the standard ILP framework by developing a special refinement operator exploiting taxonomies on terms(concepts) and predicates(relations), which leads to an order of magnitude speedup during feature generation. Since the generated features also form a taxonomy, we have also adapted a propositional rule learning algorithm to take this taxonomy into account achieving a significant speedup and more compact rules. This work was originally published in [104].

The work in Part II contributes to the research topic of automatic construction of knowledge discovery workflows. We developed a knowledge discovery ontology (KD ontology) describing key components of the knowledge discovery task including data, algorithms and models. In contrast to the ontologies existing at the time the KD ontology describes also relational learning algorithms and expressive knowledge types and offers support for workflow construction and annotation.

We investigated integration of planning and ontological reasoning for automatic workflow construction. The baseline approach consists of converting the task and algorithms description from ontology to standard planning formalism PDDL and subsequent use of classical planning algorithm. This approach provides a proof-of-concept solution and enables use of third party planning algorithms. This work was originally published in [103].

The second approach implements planning with querying the KD ontology using SPARQL query language and Pellet reasoner. The last version of the algorithm goes one step beyond classical planning by exploiting the taxonomy of algorithms provided by the KD ontology. It significantly improves the scalability of the planner with respect to number of available algorithms. Moreover the taxonomy of algorithms can also be exploited in the presentation of the workflows to the user. This work was published in [86].

The proposed methodology for knowledge discovery workflow annotation and construction was integrated into the Orange4WS[1] knowledge discovery platform.

1.4 Overview of the Thesis

The thesis consists of an introduction, six chapters divided into three main parts, and a conclusion. Part I of the thesis focuses on exploiting domain knowledge in knowledge discovery. Part II is devoted to the development and application of meta knowledge to guide the knowledge discovery process. Part III describes application of the techniques presented in Part I and Part II.

Part I starts with Chapter 2, which introduces the key concepts from knowledge discovery and knowledge management, on which the presented work is based. Chapter 3 starts with an overview of approaches to learning with higher order domain knowledge. Then it presents a framework for relational learning extending the classical ILP framework with

[1]http://orange4ws.ijs.si/

taxonomic background knowledge. The sections describing our framework are an edited version of [104].

Part II starts with Chapter 4, which introduces the concepts of workflow composition and using ontologies for this task. Chapter 5 provides an extensive review of previous efforts to formalize the knowledge discovery domain and presents a knowledge discovery ontology, which we have developed. The sections presenting our KD ontology are an edited version of [103]. Automated workflow composition for knowledge discovery using this ontology is discussed in Chapter 6. The sections describing the PDDLPlanner and PelletPlanner were originally published in [103]. The sections referring to the planner utilizing the algorithms hierarchy are an edited version of [86].

Part III consists of Chapter 7, which presents applications in domains of product engineering, which were originally published in [105] and [103], and service oriented knowledge discovery published in [86].

Finally Chapter 8 contains discussion and concluding remarks.

Part I

Exploiting Domain Knowledge

Chapter 2

Knowledge discovery and knowledge representation

This chapter introduces the key concepts from knowledge discovery and knowledge management on which the presented work is based. Then an overview of the state of the art in each of the relevant areas is presented. However, since this work this work draws upon research in quite diverse fields, a more detailed review of the related work is then included in the chapters addressing the individual issues.

The core of this study is focused on knowledge discovery and data mining and is targeted mainly at members of data mining and knowledge discovery community. Therefore we assume that the reader is familiar with basic concepts from this field and some data mining algorithms, therefore we define only the concepts which are central to our work and concepts for which slightly ambiguous definitions exist. An extensive description of classical approaches to data mining and knowledge discovery is found in [70].

We also assume the knowledge of some principal concepts of mathematical logic and logic programming and introduce only the concepts directly relevant for this work. Complete definitions of these terms and theoretical foundations of inductive logic programming can be found in [79]. For a comprehensive discussion of inductive logic programming approaches and tools refer to [62].

Since the concepts of ontologies and semantic technologies are relatively new in context of data mining, we briefly define and explain notions of ontology and knowledge representation formalisms before proceeding to aspect of this domain directly relevant to this work. For introduction to knowledge representation and ontologies refer to [93]. An extensive description of ontologies including tools, formalisms and development methodologies is provided in [42]. Details on theoretical aspects of knowledge representation formalisms can be found in [5].

2.1 Knowledge Discovery and Data Mining

In the early days of machine learning research knowledge discovery and knowledge representation were inextricably linked. Michalski defined learning as "constructing or modifying representations of what is being experienced" [73]. He added that: "There are two basic forms of learning: knowledge acquisition and skill refinement" [73]. In this work we concentrate on the first form, which he named knowledge acquisition and defined as "learning new symbolic information coupled with the ability to apply that information in an effective manner" [73]. The term knowledge acquisition is currently more commonly used for the process of acquiring knowledge from a human expert for and expert system. Therefore we shall use the term knowledge discovery instead, since this can also be viewed as an extension of the well established concept of knowledge discovery in databases KDD, which is defined as "the the non-trivial process of identifying valid novel, potentially useful and ultimately understandable patterns in data" [36] or alternatively as "an automatic exploratory analysis and modeling of large data repositories. ... the organized process of identifying valid, novel useful, and understandable patterns from large and complex data sets" [70].

The term knowledge discovery in databases is sometimes used interchangeably with the term data mining, let us therefore explore relationship between these two terms before proceeding to our definition of knowledge discovery. In [70] KDD is described as an iterative and interactive process with nine stages: (1) Domain Understanding & KDD Goals, (2) Selection & Addition, (3) Preprocessing, (4) Transformation, (5,6,7) Data Mining, (8) Evaluation & Interpretation, (9) Discovered Knowledge (Visualization & Integration). In this context data mining (DM) is defined as "the core of the KDD process, involving the inferring of algorithms that explore the data, develop the model and discover previously unknown patterns" [70].

An alternative terminology is provided by the CRISP-DM process model [18], which uses the term data mining for the whole process and describes its individual phases as: Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation, Deployment. The CRISP-DM model describes the knowledge discovery process more from industrial point of view. Therefore it emphasizes the activities done by the data owners, while the description of the KDD process examines in more detail the activities performed by the data analysts. Thus CRISP-DM defines Business Understanding as a separate stage followed by Data Understanding stage, while these two stages are combined into Domain Understanding and KDD Goals definition in KDD process description. Also the Deployment stage of CRISP-DM focuses on data owner activities, while the last stage of KDD process focuses more on knowledge visualization and integration. The Modeling stage of CRISP-DM roughly corresponds to the stages labeled as Data Mining as defined by [70].

Since one of the main objectives of this work is formalization of the knowledge discovery task from goals formulation to integration of the discovered knowledge and the analysis of business processes related to knowledge discovery is out of the scope of this work, we adopt the terminology used in [70], rather than CRISP-DM.

We define knowledge discovery as the non-trivial process of constructing or modifying representations of what is being experienced, so that novel, valid and potentially useful knowledge becomes explicit and can be applied. This definition modifies the definition of KDD mainly in the following two ways:

- It removes the limitation of input to data and views both input and output of the KDD process as a form of knowledge representation. We want to emphasize that the input for knowledge discovery is not limited to data, but it can consist of data and background knowledge. The background knowledge itself can also be a result of some previous knowledge discovery process.

- It removes the limitation to large datasets. Based on experience of, e.g., the microarray analysis, the main source of complexity is not necessarily the size of the dataset. On the contrary it can be due to having a small dataset and a complex and large body of background knowledge.

Within this work we focus on Preprocessing to Data Mining phases of KDD process, however the presented framework is designed to contribute to better Evaluation & Integration as well, e.g., in developing a machine understandable representation of discovered knowledge to facilitate more effective integration and retrieval of the discovered knowledge.

Since the knowledge discovery tasks motivating this work require learning from data and background knowledge and since in our definition of knowledge discovery we propose to view data and discovered patterns as forms of knowledge representation inductive logic programming (ILP), which uses uniform representation for examples, background knowledge and hypotheses, is a suitable technology for the learning component.

2.2 Inductive logic programming

Inductive logic programming aims at learning a general theory in a subset of first-order logic from given examples, possibly taking background knowledge into account. The subset of first-order logic most commonly used for ILP is Horn logic [79]. The problem of learning in the framework of ILP can be formally described as follows [54]:

Definition 1 ***(ILP Learning Problem)*** *Given:*

- *a correct provability relation $\vdash$ for a first-order language L, i.e. for all $A, B \in L$: $if A \vdash B$ then $A \models B$,*
- *background knowledge B in language LB, $B \in LB \subseteq L$*
- *positive and negative examples $E = E^+ \cup E^-$ in language $LE \subseteq L$ consistent with B, $(B, E \nvdash \square)$ and not a consequence of B $(\forall e \in E : B \nvdash e)$ and*
- *hypothesis language $LH \subseteq L$.*

Find a hypothesis $H \in LH$ such that:

1. *$(B, H, E \nvdash \square)$, i.e. H is consistent with B and E,*
2. *$(B, H \vdash E^+)$, i.e. $H \wedge B$ explain E^+ and*
3. *$(\forall e \in E^- : B, H \nvdash e)$, i.e. $H \wedge B$ does not explain E^-.*

The tuple $(\vdash, LB, LE, LH)$ is called the ILP-learning problem with B, E^+, E^- defining the problem instance. Deciding whether there exists such an $H \in LH$ is called the ILP-consistency problem.

The set of examples (E) usually consists of ground unit clauses of a single target predicate. Positive examples (E+), are then ground unit definite clauses and negative examples (E-), ground unit headless Horn clauses. Other information about the examples under investigation is included in the background knowledge.

The subset of first-order logic used by most ILP algorithm is the language of Horn clauses, which are defined as follows [79]:

Definition 2 *A definite program clause is a clause containing exactly one positive and zero or more negative literals. A definite goal is a clause containing only negative literals. A Horn clause is either a definite program clause or a definite goal.*

Solving ILP problem is usually formulated as search in the space of possible hypotheses [62]. This search space can be structured by the subsumption relation. In the traditional ILP setting θ-subsumption based on the following definition of elementary substitution is used [79]:

Definition 3 ***(Elementary substitution** θ**)*** *An elementary θ-substitution for atom A is:*

1. *$X/f(X_1, ..., X_n)$ where f is a functor of arity n and X_i are variables not occurring in A*
2. *X/c where c is a constant*
3. *X/Y where X and Y are variables occurring in A*

The θ-subsumption is then defined using θ-substitution on clauses, which is a composition of elementary *theta*-substitutions on atoms [79]:

Definition 4 ***θ-subsumption*** *Let c_1 and c_2 be two clauses, then c_1 θ-subsumes c_2 if and only if $\exists$ substitution θ: $c_1\theta \subseteq c_2$*

In [62] θ-subsumption is used to define syntactic notions of generality and refinement. Clause c_1 is at least as general as clause c_2 ($c_1 \leq c_2$) if c_1 $\theta - subsumes$ c_2. Clause c_1 is more general than clause c_2 ($c_1 \leq c_2$) if $c_1 \leq c_2$ and $c_2 \leq c_1$ does not hold. In this case we say that c_1 is a generalization of c_2 and c_2 is a specialization of c_1.

A downward refinement operator is used to generate specializations of a formula, an upward refinement operator generates a set of generalizations of a formula. More formally [79]:

Definition 5 ***(Downward refinement operator)*** *Let $\langle G, \geq \rangle$ be a quasi-ordered set. A downward refinement operator for $\langle G, \geq \rangle$ is a function rho, such that $\rho(C) \subseteq \{D|C \geq D\}$ for every $C \in G$.*

Analogously an upward refinement operator δ can be defined. A refinement operator induces a refinement graph [79]. This is a directed graph, where nodes are unique members of G and (C, D) is an edge just in case $D \in \rho(C)$.

Most ILP approaches search the hypothesis space top-down, proceeding from the most general hypothesis to a hypothesis special enough to fulfill the required properties with regard to coverage of positive and negative examples, using downward refinement operator.

The branching factor of the refinement graph is very large, therefore it is important to investigate further means of structuring and pruning the graph. One common approach is to reduce the branching factor by making the refinement operator take into account the types of predicate arguments, as well as input/output mode declarations [62]. Other approach is extending the notion of generality beyond θ-subsumption. A combination of these two approaches is investigated in Chapter 3, since the θ-subsumption is not sufficient for efficient representation of subsumption hierarchies on terms and predicates.

In the ILP framework ontological information can be viewed as meta information or higher order information, which could either be built into ILP algorithms e.g. in form of some special refinement operator or could lead to learning in a language more expressive or higher order than Horn logic.

2.3 Ontologies

With the development of semantic web and knowledge management technologies, more and more expert knowledge about particular highly knowledge intensive domains such as biomedicine is becoming available in form of formal representation of concepts and objects of the domain and relationships that hold among them. This formal representation is called an ontology. One of the most famous examples is the Gene Ontology [23].

2.3.1 What is an ontology

The most widely accepted definition proposed in [45] states that an ontology is "a formal, explicit specification of a shared conceptualization." Let us discuss the constituents of the definition one by one.

A *conceptualization* is essentially a description of some domain in terms of concepts and relations, which enables us to reason about the domain. A concept is not a mere word or label of an entity. It provides a definition of the entity, which enables us to recognize whether an object or event is an instance of the given entity. It also defines the relationships of the entity with other entities in the environment and the rules that are valid for all the instances of a given entity. An often cited example is the geometrical concept of a circle. The definition of a circle can be expressed in natural language (circle is a set of points with the same distance from the center) or in mathematical notation ($x^2+y^2=r^2$). A definition of the concept also includes valid theorems on the circle such as the Thales' theorem.

The definition of concept should enable us to recognize instance of the given entity, however for different purposes different characteristics of the instance are important, therefore even though the ontologies should minimize bias resulting from a specific application scenario, it is not feasible to aim at developing one ultimate ontology of a domain. Rather different ontologies are developed representing different points of view on the domain.

Explicit specification means that the concepts are defined using a set of statements that are readable by humans and machines. An ontology thus has a meaning and can be used on its own without the system for which it was created.

Formal in practice means that the specification is encoded in a logic-based language, which enables automatic querying and deriving of new information. The core of

C,D $\rightarrow$	$A\|$	(atomic concept)
	$\top\|$	top concept
	$\bot\ \|$	bottom concept
	$\neg C\|$	complement
	$C \sqcap D\|$	intersection
	$C \sqcup D\|$	union
	$\forall R.C\|$	value restriction
	$\exists R.C\|$	existential quantification
	$(\geq nR)\,\|\,(\leq nR)\,\|$	number restriction
R $\rightarrow$	$P_1 \sqcap ... \sqcap P_m$	role conjunction

Table 2.1: The grammar for $\mathcal{ALCNR}$ language.

a formal ontology is formed by a hierarchy of concepts. Some parts of the hierarchy are defined explicitly using is-a relation, other can be inferred from axioms valid about the concept. Therefore we argue that efficient exploitation of taxonomies is the first step towards using ontologies as background knowledge in learning.

Shared indicates that the main motivation for defining ontologies is to facilitate knowledge sharing and reuse. Since one of the main motivations for creating ontologies is sharing of conceptualization the W3C consortium aims to create standards for the representation language for modeling ontologies. Currently the RDF(S) [15] and OWL [84] languages are the most popular for knowledge modeling for the semantic web. These formalisms are based on description logics, which have good properties with regard to complexity and decidability of reasoning. [5].

2.3.2 Description Logics

As we have said above, the core part of the ontology is formed by a hierarchy of concepts. Description logics is a family of representation languages that have been designed to represent hierarchies of concepts and their properties. DLs are subsets of first-order logic. The description logic $\mathcal{ALC}$, which form a basis of the DLs most commonly used for ontology modeling and knowledge representation, is a fragment of first-order logic with equality containing formulas with at most 2 variables. It contains only unary relations corresponding to the sets of objects in the domain (concepts) and binary relations (roles). A concept is defined by the necessary and sufficient conditions satisfied by objects in a set. A description logic contains a set of constructors that determine the grammar, in which the conditions can be represented. The grammar for $\mathcal{ALCNR}$ language is in Table 2.1.

A concept is interpreted as a set of individuals and a role is interpreted as a set of pairs of individuals. The domain of interpretation can be infinite. Moreover DLs are based on the open world assumption unlike most representation languages used for databases and learning.

Interpretation $\mathcal{I}$ consists of a non-empty set $\Delta^{\mathcal{I}}$ (the domain of the interpretation) and an interpretation function, which assigns to every atomic concept A a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$

and to every atomic role R a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation function is extended to concept descriptions by the inductive definitions [5].

In addition to concept definitions there are terminological axioms, which relate concepts or roles to each other. The two most generally used terminological axioms are inclusion statement and concept definition

$$C \sqsubseteq D(R \sqsubseteq S) \text{ and } C \equiv D(R \equiv S),$$

where C, D are concepts and R, S are roles. An equality with atomic concept as left-hand side is a *definition*. Definitions are used to introduce *symbolic names* for complex descriptions. E.g. by the axiom

$$Mother \equiv Woman \sqcap \exists hasChild.Person$$

A finite set of definitions $\mathcal{T}$ is called a *terminology* or TBox if no symbolic name is defined more than once. Atomic concepts in $\mathcal{T}$ occurring on the left-hand side of the axioms are called *name symbols* or *defined* concepts and concepts occurring only on the right hand side of axioms are *base symbols* or *primitive* concepts.

In addition to TBox, there is the second component of knowledge base - the world description or ABox containing assertions about individuals. In the ABox individuals are introduced by giving them names and asserting properties of these individuals.

Reasoning in description logics is usually implemented using tableau algorithms. These algorithms use negation to reduce subsumption to (un)satisfiability of concept descriptions. A tableau algorithm is defined by a set of transformation rules. The details and examples of tableau algorithms are described in [5]. There are currently several highly optimized reasoning engines available, therefore we have decided to investigate the approach of integrating the existing reasoners into our learning framework rather than developing our own reasoner.

2.3.3 Ontologies as Domain Knowledge

In this section we will focus on ontologies, which can be used as domain knowledge in a knowledge discovery task. In Part II we will discuss using ontologies to describe the knowledge discovery task. Other popular uses of ontologies, e.g., in text mining and multi agent systems are out of scope of this work.

In recent years, there has been a lot of effort in some domains to organize and formalize the knowledge available in the domain, to remove ambiguities and arrive to a standardized representation of core concepts of the domain mainly for the purpose of efficient information integration and intelligent access. This results in development of domain reference ontologies. The most notable of these is the Gene Ontology project in bioinformatics with the aim of ”standardizing the representation of gene and gene product attributes across species and databases”. Since the development the three major bioinformatic terminology bases, SNOMED-CT, Galen, and GO, started before the definition of semantic representation languages reached maturity and also since the main focus was on standardization of terms, the ontologies are lightweight and are expressible in $\mathcal{EL}$ description logic.

Semantic representation is becoming popular even in industrial use [1] for sharing and efficient searching of information in companies. The developed reference ontologies are also relatively light weight, since efficiency of reasoning is considered more important than expressivity.

In the bioinformatics domain the potential of using ontologies as additional sources of information for knowledge discovery has been recognized[2], but mostly in statistical analysis or rule learning [102]. The rule learning algorithms either worked at some level of granularity or on all levels without preserving the links, thus producing a large number of redundant rules.

In other domains the use of ontologies as background knowledge for learning has been relatively rare. In Chapter 3 we therefore examine how to efficiently exploit ontologies as background knowledge in knowledge discovery focusing on the lightweight reference domain ontologies.

[1] REPCON technological platform developed by Semantic Systems, http://www.repcon.es/

[2] GO web page lists over 100 papers on GO in data or text mining

Chapter 3

Knowledge discovery with higher order domain knowledge

Classical relational learning frameworks are able to learn not only from structured data expressed in a subset of first order logic, but also from some domain knowledge about the field under investigation expressed in the same formalism. This domain knowledge is incorporated into the background knowledge used by relational data mining algorithm.

The term background knowledge is used in the logical and relational learning realm to designate both structured information about the individual examples and any other domain knowledge available. The problem of knowledge discovery with higher order domain knowledge can be in this context viewed as a problem of integration of ontological background knowledge into relational data mining.

3.1 State of the art

The approaches to utilization of ontological background knowledge in learning can be split into 3 categories: introducing learning into an extension of description logics, using a hybrid language integrating description logics and Horn logic and learning in a more expressive formalism such as F logic. Work on sorted refinement presented in [39] also contributes to integration of hierarchical background knowledge into learning.

3.1.1 Sorted downward refinement

The background knowledge built into this refinement is based on sorted logic, which encodes the taxonomies. Sorted logic contains in addition to predicate and function symbols also a disjoint set of sort symbols. A sort symbol denotes a subset of the domain called a sort [39]. A sorted variable is a pair, $x : \tau$, where x is a variable name and τ is a sort symbol. Semantically, a sorted variable ranges over only the subset of the domain denoted by its sort symbol. The semantics of universally-quantified sorted sentences can be defined in terms of their equivalence to ordinary sentences: $\forall x : \tau \phi$ is logically equivalent to $\forall x : \neg\tau(x) \vee \phi'$ where ϕ' is the result of substituting x for all free occurrences of $x : \tau \in \phi$.

The background knowledge that is to be built into the instantiation, subsumption and refinement of sorted clauses is known as a sort theory. A sort theory is a finite set of sentences that express relationships among the sorts and the sortal behavior of the functions. Sentences of the sort theory are constructed like ordinary sentences of first-order predicate calculus except that they contain no ordinary predicate symbols; in their place are sort symbols acting as monadic predicate symbols. In [39] the form of the sort theory is restricted to two types of sentences: function sentences and subsort sentences.

Function sentence

$$\forall x_1, \ldots, x_n \tau_1(x_1) \wedge \cdots \wedge \tau_n(x_n) \rightarrow \tau(f(x_1, \ldots, x_n))$$

Subsort sentence

$$\forall x \tau_1(x) \rightarrow \tau_2(x).$$

Graph of the sort theory has to be acyclic and singly rooted. For the sort theory special substitution is defined:

Definition 6 *Sorted substitution θ is a Σ-substitution if for every variable $x : \tau$, it holds that $\Sigma \models \bar{\forall}\tau(t)$ where t is $(x : \tau)\theta$.*

Using sorted substitution, a sorted downward refinement is defined as follows:

Definition 7 ***(Sorted Downward Refinement)***. *If C is a sorted clause, its downward Σ-refinement, written $\rho_\Sigma(C)$, is the smallest set such that:*

1. *For each θ that is an elementary Σ-substitution for $C, \rho_\Sigma(C)$ contains $C\theta$.*

2. *For each n-ary predicate symbol P, let $x_1 : \mathtt{UNIV}, ..., x_n : \mathtt{UNIV}$ be distinct variables not appearing in C. Then $\rho_\Sigma(C)$ contains $C \vee P(x_1 : \mathtt{UNIV}, ..., x_n : \mathtt{UNIV})$ and $C \vee \neg P(x_1 : \mathtt{UNIV}, ..., x_n : \mathtt{UNIV})$.*

In [39] it was proved there that the sorted downward refinement is finite for finite set of predicate symbols and that it is correct and complete.

3.1.2 Learning in Description Logics

Learning in a description logics has been investigated first in [21] and then thoroughly in [6] and [53].

Refinement for description logics

[6] addresses learning in DLs using downward (and upward) refinement operators. A complete and proper refinement operator for the $\mathcal{ALER}$ description logic is constructed. To avoid overfitting, disjunctions are not allowed in the target DL. The learning problem in DLs can be stated as follows.

Definition 8 *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL knowledge base. A subset $\mathcal{A}' \subseteq \mathcal{A}$ of the assertions $\mathcal{A}$ can be viewed as (ground) examples of the target concept A:*
$\mathcal{A}' = \{A(a_1), A(a_2), ..., \neg A(b_1), \neg A(b_2), ...\}$. *The DL learning problem consists in inducing a set of concept definitions for A:*

$$\mathcal{T}'' = \{A \leftarrow C_1, A \leftarrow C_2, ...\}$$

that covers the examples $\mathcal{A}'$, i.e.

$$\left\langle \mathcal{T} \cup \mathcal{T}'', \mathcal{A}\ \mathcal{A}' \right\rangle \models \mathcal{A}'.$$

Note that the learning problem formulated above is very similar to the ILP-problem defined in section 2.2. There are two main differences though. The first one consists in the different expressivities of the hypothesis language. The second one is that DLs are based on the Open World Assumption (OWA), whereas in ILP the Closed World Assumption is usually made.

While ILP systems learn logic programs from examples, a DL-learning system should learn DL descriptions from Abox instances. Both types of learning systems traverse large spaces of hypotheses in an attempt to come up with an optimal consistent hypothesis as fast as possible. Various heuristics can be used to guide this search, for instance based on information gain, MDL [78], etc.

Refinement operators allow us to decouple the heuristic from the search algorithm. Downward (upward) refinement operators construct specializations (generalizations) of hypotheses and are usable in a top-down (respectively bottom-up) search of the space of hypotheses.

Definition 9 *A downward (upward) refinement operator is a mapping ρ from hypotheses to sets of hypotheses (called refinements), which produces only specializations (generalizations), i.e. $H' \in \rho(H)$ implies $H \models H'(respectively H' \models H)$.*

Definition 10 *A downward refinement operator ρ on a set of concepts ordered by the subsumption relationship $\sqsubseteq$ is called*

- *locally finite iff $\rho(C)$ is finite for every hypothesis C.*
- *complete iff for all C and D, if D is strictly subsumed by C, then $\exists E \in \rho * (C)$ such that $E \equiv D$.*
- *weakly complete iff $\rho * (\top)$ = the entire set of hypotheses.*
- *redundant iff there exists a refinement chain from C_1 to D not going through C_2 and a refinement chain from C_2 to D not going through C_1.*
- *minimal iff for all C, $\rho(C)$ contains only downward covers and all its elements are incomparable.*
- *proper iff for all C and D, $D \in \rho(C)$ entails $D \sqsupset C$.*

Refinement operator for $\mathcal{ALNER}$ is defined by the refinement rules shown above:

$[\exists\top]\, C \sqcap \exists R.\top \rightsquigarrow C \sqcap \exists R.L$ with L a DL-literal
$[\exists C]\, C \sqcap \exists R.C_1 \rightsquigarrow C \sqcap \exists R.C_2$ if $C_1 \rightsquigarrow^{\rho'} C_2$
$[\exists\exists]\, C \sqcap \exists R_1.C_1 \sqcap \exists R_2.C_2 \rightsquigarrow C \sqcap \exists(R_1 \sqcap R_2).(C_1 \sqcap C_2)$
$[\forall C]\, C \sqcap \forall R.C_1 \rightsquigarrow C \sqcap \forall R.C_2$ if $C_1 \rightsquigarrow^{\rho'} C_2$ or $C_2 = \bot$
$[\forall R]\, C \sqcap \forall R_1.D \rightsquigarrow C \sqcap \forall R_2.D$ if $R_2 \rightsquigarrow R_1$
$[PR]\, R \rightsquigarrow R \sqcap P$ with P a primitive role.
$[Lit]\, C \rightsquigarrow C \sqcap L$ with L a DL-literal such that $C \sqcap L$ is consistent.

Proposition 1 *There exist no minimal and complete $\mathcal{ALER}$ refinement operators.*

There cannot exist a minimal refinement step $C \rightsquigarrow \forall P.\ \bot$, since there exists a sufficiently large n (for example, larger than the size of C) such that $C \sqsupset C \sqcap \underbrace{\forall P. ... \forall P.}_{n} A \sqsupset \forall P.\ \bot$.

A significant difference in expressiveness between DL and HL is that DL definitions like $A \leftarrow \forall R.C$ and $A \rightarrow \exists R.C$ involve existentially quantified variables and thus cannot be expressed in pure LP. Using the meta-predicate *forall*, we could approximate $A \leftarrow \forall R.C$ as $A(X) \leftarrow forall(R(X,Y), C(Y))$. But while the former definition is interpreted w.r.t. the OWA, the latter is interpreted w.r.t. the CWA, which makes it closer to $A \leftarrow \forall KR.C$, where K is an epistemic operator as in [33]. Also, while DLs provide inference services (like subsumption checking) to reason about such descriptions, LP systems with the meta-predicate forall can only answer queries, but not reason about such descriptions.

Although the OWA is sometimes preferable to the CWA [65], the OWA is a problem when testing that a definition, for example $A \leftarrow \forall R.C$, covers a given example, for instance $A(a_i)$. Examples are unavoidably incomplete. Even if all the known R-fillers of a_i from the Abox verify C:
$A = \{R(a_i, b_{i1}), C(b_{i1}), ..., R(a_i, b_{in}), C(b_{in})\}$ this does not mean that a_i verifies $\forall R.C$ [64], so the antecedent $\forall R.C$ of $A \leftarrow \forall R.C$ will never be satisfied by an example a_i (unless explicitly stated in the KB). However, a_i will verify $\forall KR.C$ because all the known R-fillers of a_i verify C, so the definition $A \leftarrow \forall KR.C$ covers the example $A(a_i)$, as expected. Thus, when checking example coverage. we need to close the roles (for example, by replacing R with KR, or, equivalently, assuming that the known fillers are all the fillers).

Definition 11 *In the case of a DL knowledge base $\langle \mathcal{T}, \mathcal{A} \rangle$, for which $\mathcal{A}' = A(a_1), ...A(a_p), \neg A(a_{p+1}), ...\neg A(a_{p+n}) \subseteq \mathcal{A}$ are considered examples, we say that the sufficient definition $A \leftarrow C$ covers the example a_i iff $cl \left\langle \mathcal{T} \cup A \leftarrow C, \mathcal{A}\mathcal{A}' \right\rangle \models A(a_i)$, which is equivalent with the inconsistency of $cl \left\langle \mathcal{T} \cup A \leftarrow C, \mathcal{A} \backslash \mathcal{A}' \cup \neg A(a_i) \right\rangle$, where $cl \langle \mathcal{T}, \mathcal{A} \rangle$ denotes the role-closure of the knowledge base $\langle \mathcal{T}, \mathcal{A} \rangle$ (which amounts to replacing roles R with KR).*

Definition 12 *The necessary definition $A \rightarrow C$ is verified in $\langle \mathcal{T}, \mathcal{A} \rangle$ iff $\forall A(a_i) \in \mathcal{A}', cl \langle \mathcal{T}, \mathcal{A} \cup \neg C(a_i) \rangle$ is inconsistent.*

A top-down DL learning algorithm would simply refine a very general definition of the target concept, like $A \leftarrow T$, using a downward refinement operator until it covers no negative examples.

Learning of $\mathcal{ALN}$ concept descriptions by ILP methods

In [53] a different approach to learning of DL concepts is introduced. The paper proposes an encoding of $\mathcal{ALN}$ terms into Horn clauses. It can be proved that the predicates created by this encoding are always used in such a way that semantic constraints on the interpretation that would formalize the semantic of the encoded DL-terms are respected, i.e. that this encoding function is a polynomial reduction of IDLP to ILP. Before the encoding function is applied, $\mathcal{ALN}$ terms have to be normalized. The encoding function is defined below.

Definition 13 ***$\mathcal{ALN}$ encoding into constraint Horn logic***
$\Phi(C) = h(X) \leftarrow \Phi(norm(C), X).$
$\Phi(\bot, X) = \bot\ (X)$
$\Phi(P \sqcap C, X) = cp_P(X), \Phi(C, X)$
$\Phi(\neg P \sqcap C, X) = cn_P(X), \Phi(C, X)$
$\Phi(\geq nR \sqcap = mR \sqcap \forall R.C_R \sqcap C, X) = rr_R(X, [n..m], Y), \Phi(C_R, Y), \Phi(C, X)$
$\Phi(\leq mR \sqcap \forall R.C_R \sqcap C, X) = rr_R(X, [0..m], Y), \Phi(C_R, Y), \Phi(C, X)$
$\Phi(\geq nR \sqcap \forall R.C_R \sqcap C, X) = rr_R(X, [n..*], Y), \Phi(C_R, Y), \Phi(C, X)$
$\Phi(\geq nR \sqcap = mR \sqcap C, X) = rr_R(X, [n..m], Y), \bot\ (Y) i\Phi m = 0, \Phi(C, X)$
$\Phi(\forall R.C_R \sqcap C, X) = rr_R(X, [0..*], Y), \Phi(CR, Y), \Phi(C, X)$
$\Phi(\leq mR \sqcap C, X) = rr_R(X, [0..m], Y), \bot\ (Y) i\Phi m = 0, \Phi(C, X)$
$\Phi(\geq nR \sqcap C, X) = rr_R(X, [n..*], Y), \Phi(C, X)$
where Y is always a new variable not used so far and $\sqcap C$ means, if there are conjuncts left, recursion on them , $\Phi(C, X)$ has to continue. For any normalized concept term only the first matching one has to be applied.

The paper then proposes simulation of subsumption and lcs in the following theorem.

Theorem 1 ***(Simulation of subsumption and lcs).*** *A concept description C subsumes a concept description D ($C \sqsubseteq D$), iff the encoding of C $\theta_{I\bot}$-subsumes the encoding of D and $lcs(C, D) \equiv \Phi^{-1}(lgg_{I\bot}(\Phi(C), \Phi(D)))$.*

3.1.3 Hybrid languages

The language for learning most often used in ILP is the language of Horn clauses. It is suitable for learning, because it is a tractable subset of first-order logic and several efficient systems for learning Horn clauses have been developed [94],[78]. However Horn clauses are not expressive enough to model rich hierarchical structure. Description logics on the other hand were designed specially to model hierarchies of concepts and relations. They are based on open world assumption and may have infinite domain of interpretation.

Therefore Horn clauses and description logics are incomparable subsets of the first-order logic i.e. they cannot be reduced one to the other.

The first system that combined frames with statements in first-order logic was KRYPTON [13]. KRYPTON allowed statements that were more expressive than Horn rules, but used very limited description logic. The reasoning engine of KRYPTON was based on modifying a resolution engine to account for terminological inferences and was not complete. Combination of Horn rules and description logics was then considered in the CLASP system based on LOOM [69] . Since LOOM is undecidable language, CLASP algorithms are also not complete.

There are approaches to combining function free Horn clauses with some description logic to form a decidable system: $\mathcal{AL}$-log [67] and CARIN [65]. We will examine each of them in more detail in the following sections.

$\mathcal{AL}$-log

$\mathcal{AL}$-log combines DATALOG [17] and $\mathcal{ALC}$ description logic. Concept assertions are used essentially as type constraints on variables. For constrained DATALOG clauses $\mathcal{B}$-subsumption based on ground substitution is introduced. The background knowledge is represented as an *$\mathcal{AL}$-log* i.e. DL knowledge base. Hypotheses are represented as constrained DATALOG clauses that are called $\mathcal{O}$-queries and are essentially linked and connected (or range-restricted) constrained DATALOG clauses [67].

Definition 14 *A constrained DATALOG clause is an implication of the form* $\alpha_0 \leftarrow \alpha_1, ...\alpha_m \& \gamma_1, ...\gamma_n where m \geq 0, n \geq 0, \alpha_i$ *are DATALOG atoms and* γ_i *are constraints. A constrained DATALOG program* Π *is a set of constrained DATALOG clauses.*

Reasoning in *$\mathcal{AL}$-log* is based on constrained SLD resolution.

Definition 15 *Let* $Q^{(0)}$ *be a query* $\leftarrow \beta_1, ...\beta_m \& \gamma_1, ...\gamma_n$ *to a $\mathcal{AL}$-log knowledge base* $\mathcal{B}$*. A constrained SLD-refutation for* $Q^{(0)}$ *in* $\mathcal{B}$ *such that:*

1. *for each derivation* d_i*,* $1 \leq i \leq s$*, the last query* $Q^{(n_i)}$ *of* d_i *is a constrained empty clause;*
2. *for every model* $\mathcal{I}$ *of* $\mathcal{B}$*, there exists at least one derivation* d_i*,* $1 \leq i \leq s$*, such that* $\mathcal{I} \models Q^{(n_i)}$*.*

Constrained SLD-refutation is a complete and sound method for answering ground queries. DATALOG clauses are also compliant with the bias of Object Identity (OI). This bias can be considered as an extension of the unique names assumption from the semantic level to the syntactic one.

[67] presents an ideal refinement operator for *$\mathcal{AL}$-log*. Learning in *$\mathcal{AL}$-log* was implemented in $\mathcal{AL}$-QuIn system. This system solves a variant of the frequent pattern discovery problem, which takes concept hierarchies into account during the discovery process. $\mathcal{AL}$-QuIn solves the problem of frequent patterns discovery at l levels of description granularity using the level-wise method i.e. uses breadth-first search and alternates candidate generation and candidate evaluation phases.

CARIN-$\mathcal{ALNCR}$

CARIN knowledge bases contain both function-free Horn rules and $\mathcal{ALNCR}$ terminology. The two formalisms are combined by allowing the concepts and roles, defined in the DL terminology, to appear as predicates in the antecedents of Horn rules. It assumes that the terminological component completely describes the hierarchical structure in the domain and therefore the rules should not allow to make new inferences about the structure.

The semantics of CARIN are derived fro the semantics of its component languages. An interpretation $\mathcal{I}$ is a model of a knowledge base $\mathcal{K}$ if it is a model of each of its components [64]. An interpretation I is a mode of a rule r if, whenever α is a mapping from the variables of r to the domain $\Delta^{\mathcal{I}}$, such that $\alpha\left(\bar{X}_i\right) \in p_i^{\mathcal{I}}$ for every atom of the antecedent of r, then $\alpha\left(\bar{Y}\right) \in q^{\mathcal{I}}$, where $q\left(\bar{Y}\right)$ is the consequent of r. Finally $\mathcal{I}$ is a model of a ground fact $p\left(\bar{a}\right)$ if $\bar{a}^{\mathcal{I}} \in p^{\mathcal{I}}$. We make the *unique names assumption*, i.e. if a and b are constants in $\mathcal{K}$, then $a^{\mathcal{I}} \neq b^{\mathcal{I}}$.

Reasoning in CARIN is provided by means of existential entailment. In [65] there is provided a sound, complete and terminating algorithm for existential entailment knowledge bases that satisfies one of the following restrictions: (1) the description logic does not contain the constructors $(\forall R.C)$ and $(\leq nR)$ and the terminology contains only concept definitions that are acyclic, (2) Horn rules are role-safe, i.e. in every role atom at least one variable that appears in the atom also appears in an atom of a base predicate (a predicate that does not appear in the consequent of a Horn rule and is not a concept or role predicate). It can be proved that each of the above constructions causes undecidability. The proof can be obtained by encoding the execution of a Turing machine in a knowledge base and obtaining a reduction from the halting problem [64].

The language resulting form removing the constructors $\forall R.C$ and $(\leq nR)$ and terminological cycles the reasoning problem is decidable. This language is called CARIN-MARC (Maximal $\mathcal{ALCNR}$ Recursive CARIN). The inference algorithm proceeds in two steps. In the first step we apply a set of *propagation rules* to the ground facts in the knowledge base, thereby constructing a finite number of *completions*. The propagation phase is based on the general method of *constraint systems* that was used in [16] and [32] for deciding of satisfiability of $\mathcal{ALCNR}$ knowledge bases. Each completion describes a subset of possible modes of the ground facts and terminology of the knowledge base. Together the union of the completions describes all the possible models of the ground facts and the terminology. In the second step, the algorithm applies a Horn-rule reasoning procedure in each of the completions.

Description Logic Programs

Apart from hybrid languages attempting to combine subsets of Horn rules and description logics, [44] propose a method for interoperation between the leading Semantic Web approaches to rules (RuleML Logic Programs) and ontologies (OWL/DAML+OIL i.e. $\mathcal{SHOIQ}(D)$ Description Logic) via analyzing their expressive intersection. They define a new intermediate knowledge representation (KR) contained within this intersection: Description Logic Programs (DLP), and the closely related Description Horn Logic (DHL).

The DLP-fusion is performed via the bidirectional translation of premises and inferences (including typical kinds of queries) from the DLP fragment of DL to LP, and vice versa. The mapping and inferences have been implemented in the system Bubo, but the original results have been discouraging [44].

3.1.4 Learning in more expressive languages

F-logic

Frame Logic (or F-logic) provides a logical foundation for frame-based and object-oriented languages for data and knowledge representation. It accounts in a declarative fashion for structural aspects of object-oriented and frame-based languages. It is a representation language with higher order syntax e.g. it contains set-valued functions. Also classes in the IS-A hierarchy can be seen as sets ordered by the subset relation. Furthermore, attributes and methods are also viewed as objects. However the semantics of F-logic is first-order.

Definition 16 *The alphabet of an F-logic language, (L), consists of:*

- *a set of object constructors, (F)*
- *an infinite set of variables, $\mathcal{V}$*
- *auxiliary symbols, such as, (,),[,],$\rightarrow, \twoheadrightarrow, \Rightarrow$ etc.*
- *usual logic connectives and quantifiers, $\vee, \wedge, \neg, \leftarrow, \forall, \exists$*

Definition 17 *(Molecular Formulas) A molecule in F-logic is one of the following statements:*

1. *An is-a assertion of the form C::D or of the form O:C, where C,D and O are id-terms.*
2. *An object molecule of the form O[a ';'-separated list of method expressions]. A method expression can be either a non-inheritable data expression an, inheritable data expression or a signature expression.*
   - *Non-inheritable data expression take one of the following two forms:*
     - *a non-inheritable scalar expression ($k \geq 0$):*
       ScalarMethod @ $Q_1, ..., Q_k \rightarrow T$
     - *a non-inheritable set-valued expression ($l, m \geq 0$):*
       SetMethod @ $R_1, ...R_l \twoheadrightarrow S_1, ..., S_m$
   - *Inheritable scalar and set-valued data expression are like the non-inheritable expression except that $\rightarrow$ is replaced with $\rightarrowtail$ and similarly for $\twoheadrightarrow$.*
   - *Signature expressions also take two forms:*
     - *A scalar signature expression ($n, r \geq 0$):*
       ScalarMethod @ $V_1, ...V_n \Rightarrow (A_1, ..., A_r)$

– *A set-valued signature expression* ($s, t \geq 0$)*:*
SetMethod @ $W_1, ...W_s \Rightarrow\!> (B_1, ...B_t)$

Complex F-formulae are built out of simpler F-formulae by means of logical connectives and quantifiers. F-logic database includes database facts, general class information, deductive rules and queries. To incorporate predicates directly, F-language can be extended with a new set $\mathcal{P}$ of predicate symbols. If $p \in \mathcal{P}$ is an n-ary predicate symbol and $T_1, ..., T_n$ are id-terms, then $p(T_1, ..., T_n)$ is a predicate molecule.

In [56] a proof theory for monotonic part of F-logic is provided together with proofs of its soundness and completeness. The non-monotonic elements of F-logic include non-monotonic inheritance, semantics of well-typing and semantics of F-logic programs.

To avoid unnecessary complexity, a subset of F-logic suitable for our task would have to be defined. Also a transformation of semantic annotations from description logics, which are relation-centered into object-oriented language, would have to be defined.

HiLog

HiLog [19] is an object-centered language with higher-order syntax and first-order semantics. The concept of equality is not defined extensionally in HiLog. HiLog was originally developed to provide a clean declarative semantics to higher-order logic programming represented e.g. by meta-predicates in Prolog e.g. Prolog meta-predicate call.

Definition 18 *The alphabet of a language $\mathcal{L}$ of HiLog, consists of:*

- *a set countably infinite set of variables, $\mathcal{V}$*
- *a countable set of parameter symbols, $\mathcal{S}$ (assuming $\mathcal{V}$ and $\mathcal{S}$ are disjoint)*
- *usual logic connectives and quantifiers*
- *auxiliary symbols, such parentheses etc.*

The set $\mathcal{T}$ of HiLog terms of $\mathcal{L}$ is a minimal set of strings over the alphabet satisfying the following conditions:

- $\mathcal{V} \cup \mathcal{S} \subseteq \mathcal{T}$
- *If $t, t_1, ..., t_n$ are in $\mathcal{T}$, then $t(t_1, ..., t_n) \in \mathcal{T}$, where $n \geq 1$.*

HiLog supports multiple roles for parameter symbols. Parameters are arityless and there is no distinction between predicate, function and constant symbols. HiLog allows complex terms to be viewed as functions, predicates and atomic formulae. E.g. a generic transitive closure predicate can be defined as follows:
$closure(R)(X, Y) \leftarrow R(X, Y).$
$closure(R)(X, Y) \leftarrow R(X, Z), closure(R), (Z, Y).$

A semantic structure for HiLog is defined as follows:

Definition 19 *A semantic structure for HiLog, **M**, is a quadruple $\langle U, U_{true}, I, \mathcal{F} \rangle$, where*

- *U is a nonempty set of intensions for the domain of $\boldsymbol{M}$*
- *U_true is a subset of U that specifies which of the elements in U are intensions of true propositions*
- *$I : \mathcal{S} \longmapsto U$ is a function that associates an intension with each logical symbol*
- *$\mathcal{F} : U \longmapsto \prod_{k=1}^{\inf} \left[U^k \longmapsto U\right]$ is a function, such that for every $u \in U$ and $k \geq 1$, the k-th projection of $\mathcal{F}(u)$, is a function in $\left[U^k \longmapsto U\right]$. Here Π denotes the Cartesian product of sets and $\left[U^k \longmapsto U\right]$ is the set of all functions that map U^k to U.*

It can be shown that every HiLog formula can be encoded in predicate calculus.

3.1.5 Summary

Two approaches to learning in DLs have been described: refinement in DL and transformation of learning in DL to standard ILP framework. Both approaches deal with the open world assumption of DL in a similar way using the epistemic operator defined for DL [5]. This technique can be utilized in transformation of information encoded in DL to any representation based on the closed world assumption. However, the expressivity of DLs is too limited for our purposes, since no rules with a free variable can be expressed in DLs.

To overcome this basic limitation of DLs and still preserve decidability, the hybrid languages attempt to combine the description of concept hierarchy in a DL with a rule component expressed in DATALOG. The coupling of description logics and DATALOG in *$\mathcal{AL}$-log* is relatively loose. Use of description logics is limited to concept definitions, which are evaluated in DL framework and then form only constraints for DATALOG queries. The description logic used is not sufficiently expressive to represent hierarchies on properties. In CARIN, the description logics component is more closely coupled with the rule component. The DL used in CARIN is also expressive enough to support hierarchy on predicates. However DATALOG does not allow to express constraints on predicates, e.g., a predicate is symmetric, in a declarative way. Therefore more expressive languages were investigated.

Two expressive languages were introduced: F-logic and HiLog. F-logic was designed to account in a declarative fashion for most of the structural aspects of object-oriented and frame-based languages. It contains constructs specific to object-oriented programming, which have no use in our application. The higher order information such as constrains on predicates cannot be expressed directly. It can only be encoded using objects. Therefore transformation would have to be designed not only for description of concept hierarchy described in DLs but also for constraints on predicates. HiLog enables us to express constraint in a declarative way and is also expressive enough to describe hierarchies of concepts usually represented in DLs. HiLog is also considered as language suitable for describing more complex aspects of the semantic web, which cannot be modeled in the standard DLs [110], [77].

While learning in HiLog covers all our requirements developing a framework based on HiLog would include defining a suitable subset of HiLog and designing transformations of knowledge expressed in ontology modeling formalisms into HiLog. Moreover a special engine for HiLog programs execution is required. To avoid the complexity of development based on HiLog, which would exceed the scope of this thesis, we decided to investigate integration of higher order background knowledge by proposing a specialized refinement operator exploiting taxonomic background knowledge. This solution also meets our core requirements and is more tractable in the initial stage of development.

3.2 Feature generation with taxonomic background knowledge

The presented approach is based on extending the classical ILP task using a special refinement operator. The proposed operator focuses on two core relations present in the ontologies: subsumption relation on concepts and on predicates.

3.2.1 Integration of taxonomies

The simple ontology written in RDF formalism can be represented by acyclic directed graph (DAG). Concepts are defined only by means of declaring class and its place in the class hierarchy. No set operators or restrictions commonly used in OWL are present in the background knowledge and dataset. Only domain and range are defined for the properties and a hierarchy on properties is induced by means of the *subpropertyOf* relation. The definition of *rdfs:subPropertyOf* relation in [15] originally states: If a property P is a subproperty of property P', then all pairs of resources which are related by P are also related by P'. For our purposes the definition of subPropertyOf relation is restricted to cases where domain and range of P and P' are defined by some class or set of classes. Then it must hold that domain of P is equivalent to or subclass of the domain of P' and the same holds for range. Therefore we have to deal essentially with taxonomies on terms and predicates.

Our approach to propositionalization is based on RSD system [106]. In RSD, a predicate declaration assigns a type symbol to each argument, from a finite set of type symbols. The presented approach replaces the notion of type with that of *sort* borrowed from the formalism of sorted logic, which is suitable for encoding term taxonomies. Sorted logic was introduced in Section 3.1.1. We shall introduce its use by an example. The Gene Function Ontology declares a concept `binding` and its subconcept `protein_binding`. Such concepts are reflected by terms in ILP. It is possible to declare in background knowledge e.g.

```
subclass(binding, protein_binding).
geneFunction(G, F1) :- geneFunction(G, F2), subclassTC(F1, F2).
```

(where `subclassTC/2` is defined as the transitive closure of `subclass/2`). Unfortunately, in such an approach, for the following two exemplary clauses (hypotheses)

$C =$ `activeGene(G):- geneFuction(G, binding).`
$D =$ `activeGene(G):- geneFuction(G, protein_binding).`

it does not hold $C\theta \subseteq D$, so clause D is not obtained by applying a specialization refinement operator onto clause C. Similar reasoning applies to taxonomies on relations (predicates).

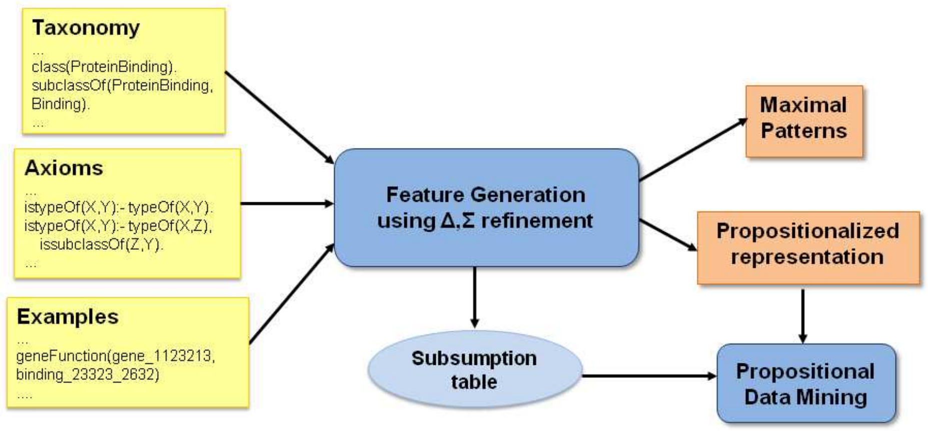

Taxonomy
...
class(ProteinBinding).
subclassOf(ProteinBinding, Binding).
...
Axioms
...
istypeOf(X,Y):- typeOf(X,Y).
istypeOf(X,Y):- typeOf(X,Z), issubclassOf(Z,Y).
...
Examples
...
geneFunction(gene_1123213, binding_23323_2632)
....
Feature Generation using Δ,Σ refinement
Subsumption table
Maximal Patterns
Propositionalized representation
Propositional Data Mining

Figure 3.1: An overview of the RDM system consisting of feature genearation using extended sorted refinement and subsequent propositional learning.

In the present propositionalization approach, terms in features are constants or sorted variables. Background knowledge consists of an ordinary first-order theory and a sort theory Σ. A *declaration* for a predicate of symbol π and arity n has the form

$$\pi(m_1\tau_1, \ldots, m_n\tau_n)$$

where $m_i \in \{+, -\}$ denotes whether i-th argument is an input (+) or an output (-). Besides the constraints imposed on features in RSD, correct features must respect the sort relationships. Formally, a literal Lit may appear in a feature only if there is a declaration $\pi(m_1\tau_1, \ldots, m_n\tau_n)$ and a Σ-sorted substitution θ such that $\pi(\tau_1, \ldots, \tau_n)\theta = Lit$. Next we turn attention to the refinement operator through which features are constructed.

3.2.2 Extending θ-subsumption with Σ-substitution

We have adapted the *sorted downward refinement* from [39], which accounts for term taxonomies, to further account for the earlier defined feature constraints and predicate declarations used in propositionalization, and for a further kind of taxonomy – the *predicate taxonomy* – often available in ontology data. This taxonomy is encoded through meta-predicates in the form

`subrelation(`$pred_1/n$`, `$pred_2/n$`).`

providing the explicit meta-information that goal $pred_1(Arg_1, \ldots, Arg_n)$ succeeds whenever goal $pred_2(Arg_1, \ldots, Arg_n)$ succeeds, i.e. $pred_1$ is more general. The directed graph corresponding to the entire set of the `subrelation/2` statements (where direction is such that edges start in the more general goal) is assumed to be a forest. The set of its roots is exactly the set of predicates declared through the predicate declarations defined in the previous section. It is assumed that the non-root predicates inherit the argument declarations from their respective roots.

As feature heads are fixed in our propositionalization framework, we are concerned with refinement of their bodies, i.e. conjunctions. We will use the notion of an *elementary* Σ-*substitution*. Its general definition can be found in [39], however, adapted to our framework, the definition simplifies.

An elementary Σ-*substitution* for a sorted conjunction C is $\{x : \tau_1\} \to \{x : \tau_2\}$ where $\{x : \tau_1\}$ occurs in C and Σ contains the subsort formula $\forall\psi\tau_2(\psi) \to \tau_1(\psi)$ for some variable ψ. If $\{x : \tau_2\}$ already occurs in C, then x is deterministically renamed[1] to a variable not occurring in C. Unlike in [39], we can disregard the case of substituting a sorted variable by a function (as we work with function-free features) and, similarly to RSD [106], we neither allow to unify two distinct variables (an equality theory can be defined instead in background knowledge).

Let C be a conjunction of non-negated atoms where any term is either a constant or a sorted variable, Σ be a sort theory, and Δ a set of predicate declarations. We define the *downward* Δ, Σ-*refinement*, written $\rho_{\Delta,\Sigma}(C)$, as the smallest set such that:

1. For each θ that is an elementary Σ-substitution for $C, \rho_{\Delta,\Sigma}(C)$ contains $C\theta$.

2. Let $\pi(m_1\tau_1, \ldots, m_n\tau_n)$ be a declaration in Δ such that for each i for which $m_i = +$, C contains a variable (denote it x_i) of sort τ'_i which equals or is a subsort of τ_i. Let further $\{x_i | m_i = -\}$ be a set of distinct variables not appearing in C. Then $\rho_{\Delta,\Sigma}(C)$ contains $C \wedge \pi(x_1 : \upsilon_1, ..., x_n : \upsilon_n)$, where $\upsilon_i = \tau'_i$ if $m_i = +$ and $\upsilon_i = \tau_i$ otherwise.

3. Let C contain a literal $pred_1(x_1\tau_1, \ldots, x_n\tau_n)$ and let $pred_2$ be a direct subrelation of $pred_1$. Then $\rho_{\Delta,\Sigma}(C)$ contains C', which is acquired by replacing $pred_1(x_1\tau_1, \ldots, x_n\tau_n)$ with $pred_2(x_1\tau_1, \ldots, x_n\tau_n)$ in C.

Under very general assumptions on Δ, the defined refinement operator is (i) finite, (ii) complete, in that all correct features (as defined in [106] and Section 3.1.1) up to variable renaming are enumerated by its recursive closure, whenever the initial C in the recursive application of $\rho_{\Delta,\Sigma}(C)$ is true, and also (iii) non-redundant, in that $\rho_{\Delta,\Sigma}(C_1) \cap \rho_{\Delta,\Sigma}(C_2) = \{\}$ if $C_1 \neq C_2$. However, the operator is not necessarily correct, in that all its products would be correct feature bodies. In particular, it may produce a conjunction violating the undecomposability condition defined in [106].

3.3 Feature taxonomies

During the recursive application of the refinement operator, a feature generality taxonomy becomes explicit. For purposes of enhancing the performance of the propositional learning algorithm applied subsequently on the propositionalized data, we pass the feature taxonomy information to the learner through two Boolean matrices.[2] Assume that features

[1]That is, we do not allow several elementary substitutions differing only in the chosen renaming.

[2]While alternative data structures are of course possible for this sake, the elected binary matrix form requires modest space for encoding (our implementation uses one byte for each 8 matrix elements) and also is conveniently processed in the propositional algorithm implementation.

$Propositionalize(\Delta, B, \Sigma, E, l)$: **Given**, a set Δ of predicate declarations, a first-order theory (background knowledge) B, a sort theory Σ, a set of unary ground facts (examples) $E = \{e_1, \ldots, e_m\}$ and a natural number l; **returns** a set $\{f_1, \ldots, f_n\}$ of constructed features, each with at most l atoms in the body, an elementary subsumption matrix $\mathbf{E}$, an exclusion matrix $\mathbf{X}$, and an attribute-value matrix $\mathbf{A}$ where $\mathbf{A}_{i,j} = 1$ whenever f_i is true for e_j and $\mathbf{A}_{i,j} = 0$ otherwise.

```
 1. $n = 0$; $Agenda$ = a single element list $[(C, 0)]$, where $C$ = true;
 2. If $Agenda = []$: go to 10
 3. $(Curr, Parent)$ := **Head**($Agenda$); $Tail$ := **Tail**($Agenda$)
 4. If **Nonempty**($Curr$) and **Undecomposable**($Curr$):
 5.     $n := n + 1$; $f_n$ = **AddFeatureHead**($Curr$);
 6.     $\mathbf{E}_{n,Parent} = 1$; $Parent = n$;
 7.     $\mathbf{A}_{n,1\ldots l}$ =**Coverage**($Curr, E, B, \Sigma, \mathbf{A}_{Parent,1\ldots l}$)
 8. $Rfs := \rho_{\Delta,\Sigma}(Curr)$; $Rfs := \{(Cnj, Parent) | Cnj \in Rfs, |Cnj| \leq l\}$
 9. $Agenda$ := **Append**(Rfs, Tail); go to 2
10. $\mathbf{X}$ = **Closure**($\mathbf{E}$)
11. **Return** $f_1, \ldots, f_n$, $\mathbf{E}$, $\mathbf{X}$, $\mathbf{A}$
```

Figure 3.2: A skeleton of the algorithm for propositionalization through relational feature construction using the sorted refinement operator $\rho_{\Delta,\Sigma}$.

$f_1, \ldots f_n$ have been generated with corresponding conjunctive bodies $b_1, \ldots b_n$. The *elementary subsumption matrix* $\mathbf{E}$ of n rows and n columns is defined such that $\mathbf{E}_{i,j} = 1$ whenever $b_i \in \rho_{\Delta,\Sigma}(b_j)$ and $\mathbf{E}_{i,j} = 0$ otherwise. The *exclusion matrix* $\mathbf{X}$ of n rows and n columns is defined such that $\mathbf{X}_{i,j} = 1$ whenever $i = j$ or $b_i \in \rho_{\Delta,\Sigma}(\rho_{\Delta,\Sigma}(\ldots \rho_{\Delta,\Sigma}(b_j) \ldots))$ and $\mathbf{X}_{i,j} = 0$ otherwise.

A skeleton of the propositionalization algorithm is shown in Fig. 3.2. The algorithm is a depth-first search generally similar to the feature constructor of RSD [106]. The main difference lies in using the novel sorted refinement operator $\rho_{\Delta,\Sigma}$ and also in creating the matrices $\mathbf{E}$ and $\mathbf{X}$ storing the generality taxonomy of constructed features. The **Undecomposable** procedure checks whether a feature is not a conjunction of already generated features, through a method used in RSD and detailed in [106]. The **AddFeatureHead** forms a feature clause by formally attaching a head to the body, which consists of the constructed conjunction $Curr$. The **Coverage** procedure verifies the truth value of a conjunction for all examples in E returning a vector of Boolean values. The verification is done by a transformation of the sorted conjunction $Curr$ to an ordinary first-order conjunction as explained in Sec. 3.1.1 and then using a standard resolution procedure against a Prolog database consisting of B and Σ. For efficiency, the procedure obtains the coverage $\mathbf{A}_{Parent,1\ldots l}$ of the closest ancestor (subsuming) conjunction whose coverage was tested: any example i such that $\mathbf{A}_{Parent,i}$ is false can be left out of testing as it makes the current conjunction necessarily false as well. The **Closure** procedure computes the transitive closure of the elementary subsumption relation captured in $\mathbf{E}$ in the manner described above, and represents the closed relation analogically in matrix $\mathbf{X}$, in which it further sets $\mathbf{X}_{i,i} = 1$ for all $1 \leq i \leq n$.

We have adapted two rule learning algorithms to account for the feature taxonomy information provided by the propositionalization algorithm. The first algorithm stems from the rule inducer of RSD [106]. It is based on a heuristic general-to-specific deterministic beam search for the induction of a single propositional conjunctive rule for a given target class, and a cover-set wrapper for the induction of the entire rule set for the class. Given a set of features $F = \{f_1, \ldots f_n\}$, the standard algorithm refines a conjunction C of features into the set $\{C \wedge f_i | f_i \in F, f_i \notin C\}$. In our enhanced version, the algorithm is provided with the elementary subsumption matrix $\mathbf{E}$ and the exclusion matrix $\mathbf{X}$. Using these matrices it can prevent the useless combination of a feature and its subsumee within the conjunction, and specialize a conjunction by replacing a feature with its elementary (direct) subsumee. Furthermore, we have similarly enhanced the stochastic local DNF search algorithm introduced in [88] and later transferred into the propositionalization framework by [81]. This algorithm conducts search in the space of DNF formulas, i.e. it refines entire propositional rule sets. Refinement is done by local, non-deterministic DNF term changes detailed in [88]. In our version, the $\mathbf{X}$ matrix is used to prevent the combination of a feature and its subsumee within a DNF term.

3.4 Experimental Results

We designed experiments to assess the runtime impact of (i) the novel taxonomy-aware refinement operator in propositionalization, and (ii) the exploitation of the feature-taxonomy in subsequent propositional learning. We conducted tests in two domains. The first concerns genomics, where we used data and language declarations from [101]. The second is concerned with learning from product design data. Here the examples are semantically annotated CAD documents. We used the same learning setting and ontology data as in [105].

Figures 3.3 and 3.4 illustrate on log scale the number of conjunctions searched (Fig. 3.3) and the time spent on search (Fig. 3.4) to enumerate all conjunctions true for at least 80% examples, for increasing maximum conjunction size l. Here, we distinguish the sorted refinement operator using a special sort theory Σ encoding the taxonomy information, against the standard refinement operator, which treats the taxonomy information only as part of background knowledge. While in both cases exactly the same set of conjunctions is produced, an order-of-magnitude runtime improvement is observed for the 'taxonomy-aware' operator.

Results of experiments for propositional rule learning exploiting feature taxonomy are summarized Table 3.1. It shows the runtime spent of inducing a rule set by two algorithms (top-down and stochastic) through 10-fold cross validation in two scenarios: in the first, no feature taxonomy information is used by the algorithms, in the second, feature taxonomy is exploited. A significant speedup is observed when feature taxonomy is used without compromising the predictive accuracy.

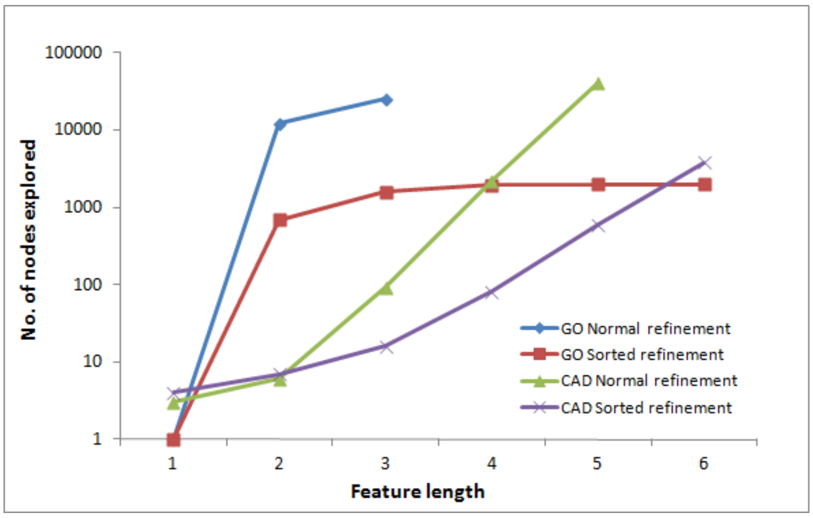


Figure 3.3: Sorted refinement vs. standard refinement on CAD and Genomic data: number of nodes explored

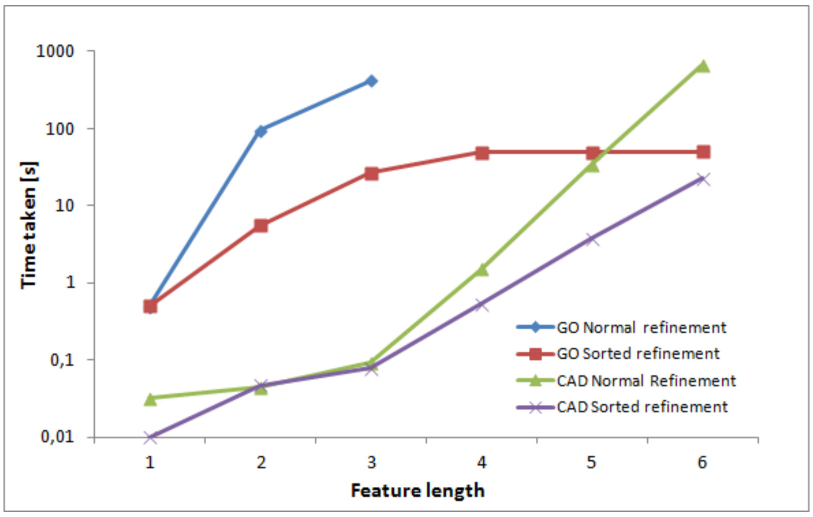


Figure 3.4: Sorted refinement vs. standard refinement on CAD and Genomic data: time taken. (Experiments exceeding 1000s were discarded)

Table 3.1: Propositional rule learning from CAD and Genomic data

Domain	CAD data		Genomic data	
Algorithm	Time taken	Predict. acc.	Time taken	Predict. acc.
Top-down	0.22 ± 0.08	0.66 ± 0.21	0.99 ± 0.65	0.79 ± 0.13
Top-down, FT	0.06 ± 0.02	0.66 ± 0.22	0.34 ± 0.19	0.76 ± 0.07
SLS	0.63 ± 1.45	0.62 ± 0.18	3.00 ± 2.59	0.79 ± 0.13
SLS, FT	0.28 ± 0.83	0.61 ± 0.19	1.90 ± 1.69	0.76 ± 0.07

Part II

Exploiting Meta Knowledge

Chapter 4

Knowledge discovery task formalization and planning

Testing and deployment of algorithms presented in Chapter 3 lead to identification of the following challenges: adding results to the existing body of knowledge for reasoning and retrieval, efficient reuse of specialized third party algorithms, flexible modifications of the framework for different data and result types. This motivated investigation of the possibilities to formalize the main ingredients of the knowledge discovery task: task formulation, input data and knowledge, algorithms and procedures and the results.

The presented system for relational data mining is in line with the trend that efficient use of domain knowledge requires a combination of diverse specialized algorithms in complex workflows, which are difficult to assemble manually for someone not expert in data mining. However once all the important components of the workflows have been formalized using the knowledge discovery ontology, planning and reasoning can be used for automatic workflow construction.

In this part of the thesis we propose a framework for automated knowledge discovery workflow construction based on a combination of planning guided by the knowledge discovery ontology. We start with introducing the key ingredients of this framework. In this chapter we present an overview of existing ontologies and standards relevant for knowledge discovery task formalization and describe the general problem of workflow construction and the main approaches to solving this problem.

4.1 Relevant Foundation Ontologies and Standards

Ontologies and their use as domain knowledge in knowledge discovery tasks were already introduced in Part I. In this section we examine the existing foundation ontologies and standards, which are relevant for constructing the knowledge discovery ontology. The relevant ontologies and standards can be divided into three categories:

- well-founded reference ontologies, such as BFO and DOLCE [72]
- standards and light-weight ontologies for web services, such as OWL-S [35] and WSDL-S [1]

- extending activities in bioinformatics to more general domains

There exist several generic toplevel ontologies, based on different philosophical orientations such as SUMO[1], Cyc[2], DOLCE[3], BFO[4] and COSMO[5]. We do not want to enter into philosophical discussions regarding the toplevel structure of these ontologies, therefore we shall examine only those ontologies, which were created to support domain ontologies for scientific research or for which some domain ontologies for scientific research have been developed - namely BFO, which explicitly declares to focus on the task of providing a genuine upper ontology which can be used in support of domain ontologies developed for scientific research, and DOLCE, for which extensions for information objects and plans already exist. Moreover, we shall not go into a systematic comparison of philosophical orientation of BFO and DOLCE, which can be found, e.g., in [43], but only examine examine the differences between BFO and DOLCE from the point of view of representation of the knowledge discovery domain.

Web services are a technology, which is available on the web and is growing rapidly, however to achieve real interoperability semantics is required. Most of the characteristics of web services are applicable to software components implementing algorithms, therefore we could reuse the formalisms and techniques of web services, even though we do not want to restrict our KD ontology to algorithms available as web services. Moreover using a standard ontology for representing the algorithms would significantly lower the barrier of adding annotations of new algorithm implementations to our ontology and allow us to utilize already available tools.

There is currently no standard ontology for web services, however there are three W3C Member submissions proposing ontologies describing the domain of web services: OWL-S [35], SWSF-SWSL [7] and WSMO [61]. The principal aim of these approaches is not to define a rich reference ontology, but rather to enable the users and software agents to discover, invoke, compose, and monitor web services.

An alternative way of adding semantics to web services was proposed in WSDL-S[1] and SAWSDL [71]. Rather than trying to develop an ontology of web services, WSDL-S provides a methodology for annotating WSDL descriptions of web services using ontologies. SAWSDL, which has the status of W3C Recommendation for adding semantics to web services, is based on ideas of WSDL-S. SAWSDL provides mechanisms by which concepts from the semantic models can be referenced from within WSDL. It does not rely on any particular semantic modeling language, only requires concepts identified by URI. It allows for multiple annotations and provides no way to dereference the model. It also does not provide a mechanism for propagating model references when an interface is created by extending other interface(s). Despite these open issues SAWSDL indicates that W3C chose

[1]http://www.ontologyportal.org/

[2]http://opencyc.org/

[3]http://www.loa-cnr.it/DOLCE.html

[4]http://www.ifomis.org/bfo

[5]http://www.micra.com/COSMO/COSMO.owl

not to commit to any particular ontology for web services.

There is also an initiative of an industrial consortium based on BPEL language for business processes [4]. However WS-BPEL lacks semantics and therefore is not suitable as a core representation language for our framework. Defining semantics for WS-BPEL is a research topic in itself, therefore we take WS-BPEL into account only by introducing the criteria of existence of a mapping between an ontology of web services and WS-BPEL into evaluation of the proposed W3C ontologies.

The Ontology for Biomedical Investigations (OBI) [24] is being developed to represent design of an investigation, protocols and instrumentation used, material used, data generated and type of analysis performed on it. Its core concepts are mapped to the BFO. The domain described by OBI overlaps with knowledge discovery and one branch of OBI ontology is devoted to data transformation. Also the applications of our knowledge discovery algorithms are often in bioinformatics. Therefore, in Chapter 5 we discuss possibilities of reuse of concepts and relations from OBI.

4.2 Knowledge Discovery Task Formalization

There has been some effort to formalize the knowledge discovery task and provide categorization of data mining algorithms. The data mining and knowledge discovery ontologies developed and used within systems for automatic workflow construction are most relevant for our work. These include the work described in [9], [14], [107], [98] and recent ongoing work described in [55] and [29]. Most of these ontologies are limited to classical propositional data mining and do not provide sufficient means for describing complex and expressive background knowledge and the results of knowledge discovery.

Other efforts to provide a systematic formalization of the DM tasks include projects MiningMart [76], DataMiningGrid [96], Knowledge Grid [22] and Universal Knowledge Grid [66]. The systems MiningMart and DataMiningGrid focus on mining propositional patterns from data stored in a relational database. All four systems contain a meta-model for representing and structuring information about data and algorithms, however, only in case of Universal Knowledge Grid is the meta-models expressed in an ontology language.

In parallel to our work, the OntoDM [83] ontology is being developed on the basis of [34]. A principled top-down approach was adopted in the development of OntoDM aiming for its maximum generality. Given the complexity of the domain, the ontology was not sufficiently refined for purposes of workflow construction at the time when we considering using it [82]. Other ongoing effort include KDDOnto [29] and DMOnto [47], which is being developed to support meta-learning for algorithm selection. Both these ontologies focus on the propositional data mining. DMOnto concentrates in particular on the classification task.

Since none of the above mentioned approaches to knowledge discovery task formalization could be directly reused for our purposes, we decided to develop a new knowledge discovery ontology. We discuss the related work in more detail in Chapter 5, while describing the design choices for our knowledge discovery ontology.

4.3 Automatic workflow construction

The term *knowledge discovery workflow* allows a wide scope of interpretations. In this work we define it as a *progression of steps (inductive, deductive, format-conversion procedures etc.) involved in generalizing specific data (e.g., measurements) into patterns, which (under appropriate interpretation) may represent novel knowledge about the domain under investigation.* Therefore is can be viewed as a special form of scientific workflow [100], currently covering the data preprocessing to data mining stages of the KDD process.

Intelligent management of knowledge discovery workflows has attracted a lot of interest in recent years. The development builds upon technologies provided by several information science fields, the two most notable of them being the *semantic web* and *grid computing*. The former provides the building blocks through which workflows can be annotated, facilitates automatic service discovery, efficient management of workflows or even their automated composition. The latter technology allows to execute workflows in a distributed computing environment while optimizing factors such as total runtime, security, etc. Both technologies actively overlap, such that, e.g., annotations extend also to physical constituents of the computing environment enabling an intelligent translation of an *abstract* (resource independent) workflow to a *concrete* one, where tasks are mapped onto particular computing resources.

Our work is mainly concerned with automatic composition of data mining and knowledge discovery workflows and we view this problem in the context of planning. In this thesis we focus on generating abstract workflows rather than on providing a workflow editing environment focused on the integration of computational resources and middleware and efficient execution, such as Triana [99], the system for scientific workflows developed in Kepler[6], WCT developed within the K-WF grid[7], and the tools developed within the DiscoveryNet project [87] and project ADMIRE [52].

4.3.1 Planning

Automatic workflow construction can be formulated as a classical planning task with algorithms as operators/actions. We adhere to notations of STRIPS planning tasks introduced in [49]. The notation is as follows.

Definition 20 ***(State)*** *A state S is a finite set of logical atoms.*

Definition 21 ***(Strips Action)*** *A STRIPS action o is a triple $o = (pre(o), add(o), del(o))$, where $pre(o)$ are the preconditions of o, $add(o)$ is the add list of o and $del(o)$ is the delete list of the action, each being a set of atoms. For an atom $f \in add(o)$, we say that o achieves f. The result of applying a single STRIPS action to a state is defined as follows:*

$$Result(S, \langle o \rangle) = (S \bigcup add(o)) \backslash del(o) \text{ if } pre(o) \subseteq S, \text{ otherwise undefined}$$

[6]http://kepler-project.org

[7]http://www.kwfgrid.eu/

Definition 22 ***(Planning Task)*** *A planning task $\mathcal{P} = (\mathcal{O}, \mathcal{I}, \mathcal{G})$ is a triple where $\mathcal{O}$ is the set of actions, and $\mathcal{I}$ (the initial state) and $\mathcal{G}$ (the goals) are sets of atoms.*

Several previous works have explored planning for knowledge discovery workflows. In this section we present only a brief overview of the notable projects and we discuss the work most relevant for our research in more detail in Chapter 6.

Within the Pegasus project [26] a planner was used to construct a concrete workflow given an abstract workflow. In our research we tackle a related yet different problem; given an ontology and a task description, we use a planner to construct an abstract workflow. A similar aim was followed by [14], [9] and [8], however this work is limited to proposing simple, non-splitting workflows, unless specific splitting templates are provided. The systems CITRUS [107] and CAMLET [98] allowed for more complex workflows, however they only made a limited use of planning for process decomposition starting from a manually defined structure.

There are two ongoing efforts to develop a framework for automatic workflow construction through planning within projects e-Lico and Virtual Mart, which are described in [55] and [30], respectively. [30] uses STRIPS planning formalization and backward chaining algorithm. [55] uses the Hierarchical Task Network (HTN) planning [89], which relies on an explicitly defined task decomposition hierarchy.

4.3.2 Web service composition

Recently workflow construction has been used for web service composition. Since we are reusing WS technologies for representation, some of the work addressing the problem of *web service composition* in the framework of planning is also relevant.

There exists also previous work dealing with domain descriptions encoded in an ontology and using a planning algorithm for service composition [91], [58] and [68]. This approach is most suitable for our work, since we use the KD ontology for formalization of the knowledge discovery task and the available algorithms. It also allows us to build upon the existing planning algorithms and standards. Therefore we focus on this approach in developing an algorithm for automatic composition of knowledge discovery workflows.

An interesting approach presented in [63] relies on computing a *causal link matrix* for all available services. Informally, this matrix captures semantic input/output compatibility among pairs of services. Services can be then viewed as nodes in a graph with the link matrix defining the edge labels. Finding a suitable sequence of services can then be elegantly reduced to finding a path in this graph. In our framework, however, we work with a more general, non-linear notion of a plan, where the inputs of an algorithm (action) combine the outputs of multiple other algorithms. Thus, *pairwise* semantic compatibility does not carry sufficient information to construct a plan in our framework and we have to rely on general planning strategies.

Other work in web service composition is less relevant for our task, since often the planning techniques focus on addressing the problems of nondeterminism, partial observability and extended goals (constraints on the behavior of a process), e.g., [85]. While the first two are important characteristics of web services composition for practical applications, they are not directly relevant for generating abstract knowledge discovery workflows.

Chapter 5

Knowledge discovery ontology

A formal conceptualization of the knowledge discovery domain by means of the Knowledge Discovery Ontology (KD ontology, for short) is the central point of our framework. The primary purpose of the ontology is to enable the workflow planner to reason about which algorithms can be used for a particular knowledge discovery task specified by the input knowledge and data and the required type of results. We are not attempting to develop a generic reference ontology of the knowledge discovery domain, but rather we would like to contribute to the effort to devise a set of ontologies for this domain, which could be mapped to each other. We propose an ontology describing relational descriptive data mining with background knowledge, which has not been covered by any of the existing ontologies.

5.1 Design Methodology

In order to facilitate ontology evaluation and compatibility with other existing ontologies for this domain, it is necessary to adhere to some set of design criteria and an ontology design methodology. A set of basic design criteria, which is cited by most ontology design methodologies, was proposed by [45]. The following 5 criteria were defined:

- **clarity**: *Definitions should be objective, i.e. independent on social/computational context, and complete. When a definition can be stated in logical axioms, it should be. All definitions should be documented with natural language.* The condition of objectivity is difficult to attain, because of a trade-off between efficiency for the primary objective of the ontology and context independence. Therefore this condition is usually reduced to trying to provide unambiguous definitions, avoid partition errors etc. The conditions of completeness and formalization also in practice lead to a trade-off between expressive power and efficiency of reasoning required for a particular task.
- **coherence**: This condition essentially states that *no contradiction can be inferred.* This condition is easy to validate by using a reasoner.
- **extendibility**: *One should be able to define new terms for special uses without requiring revision of the existing definitions.*. Fulfilling this condition can be aided by adopting a top-level ontology.

- **minimal encoding bias**: *Representation choices should not be made purely for the convenience of notation or implementation.*
- **minimal ontological commitment**: *Defining only those terms that are essential to the communication of knowledge consistent with that theory.*

These criteria are good guiding principles, however no procedure for design was specified in [45] and currently there is still no standard ontology design methodology. In a survey of ontology methodologies [41] the following two methodologies come out as the most complete and well-defined: METHONTOLOGY [37] and On-To-Knowledge [95].

On-To-Knowledge methodology covers the complete ontology lifecycle from ontology kickoff to maintenance. It defines four main stages of ontology development:
1. Ontology Kickoff. This phase consists of the following steps: requirement specification, e.g., using competency questions, analysis of input sources and development of baseline taxonomy.
2. Refinement. Refine the baseline taxonomy using concept elicitation with domain experts, conceptualization and formalization and adding relations and axioms.
3. Evaluation. Revise and expand the ontology based on feedback, analysis of usage patterns, analysis of competency questions.
4. Maintenance It is important to clarify who is responsible for the maintenance and how this should be carried out.

METHONTOLOGY has the same scope and provides more details for some of the phases. It is divided into the following 7 steps:
1. Plan. The schedule activity that identifies the tasks to be performed, their arrangement, and the time and resources needed for their completion.
2. Specification. Identify the primary objective of the ontology, its purpose, granularity level and scope.
3. Knowledge elicitation. Find and evaluate candidate ontologies and other knowledge sources to be reused.
4. Conceptualization. It consists of a sequence of tasks creating the following: glossary of terms, concept taxonomy, binary relations diagram, concept dictionary (including concept name, synonyms, instances, relations, class-attributes, instance attributes), binary relations table, instance attributes table, class attributes table, constants table, instances table, formula table and axioms table.
5. Formalization. Encode the objects created during conceptualization in a logical language
6. Integration. Integrate the ontology with existing top-level ontologies by using their leaf terms as top level concepts and possibly other domain ontologies by defining a mapping
7. Implementation. Implement the ontology in some knowledge modeling formalism, e.g., OWL DL using some ontology editor, for example Protégé[1]
8. Evaluation. Evaluate the ontology with respect to the specification, ontology design criteria and the system in which it was used

[1]The Protégé Ontology Editor and Knowledge Acquisition System, http://protege.stanford.edu/

9. Documentation. No standard guidelines exist, all concepts and relations should have also definitions in natural language.
10. Maintenance. Provide guidelines for ontology maintenance.

Our methodology is based on METHONTOLOGY and takes some inspiration from On-To-Knowledge. It consists of the following steps:
1. Specification. Identify primary objective, purpose, granularity level and scope of the ontology. Use competency questions as means for determining scope and granularity level.
2. Knowledge elicitation. Analyze knowledge sources, consider reusing existing ontologies and identify upper ontologies, which could provide top level structure of the ontology.
3. Conceptualization. Create a glossary of terms, concept taxonomy, relations and axioms.
4. Formalization + implementation. Encode the objects created during conceptualization in a logical language and implement the ontology in some knowledge modeling formalism using some ontology editor.
5. Evaluation. Evaluate whether the ontology is consistent. Evaluate the ontology with respect to the competency questions and the system in which it was used.
6. Documentation. Each concept will be defined in natural language.
7. Maintenance. Provide guidelines for ontology maintenance.

In the following sections we describe the specification phase of KD ontology development, in which the scope of the ontology is specified, and the knowledge elicitation phase, in which we also review existing relevant ontologies. After that the developed KD ontology is presented in detail.

5.2 Specification

Our KD ontology should enable the planner to reason about which algorithms can be used to produce intermediary or final results required by a specified knowledge discovery task. It should also provide support for retrieval of results of knowledge discovery tasks and reasoning about the results. Furthermore it is expected that the ontology will facilitate the management of available algorithms including their required I/O parameters and the management of essential information about performed experiments including the information about runtime for different algorithms and workflows. Therefore, besides being used by the automated workflow planner, the ontology will also be used for annotation of manually created workflows for reuse and possibly meta learning. The ontology should focus on descriptive data mining and rule learning from relational data with complex background knowledge. Thus the ontology should also formalize the types of background knowledge.

The scope and granularity of the ontology is determined by the set of competency questions that were created on the basis of three application scenarios: SEVENPRO project[2], knowledge discovery from gene expression data [102] and Orange toolkit [27]. Some examples of the competency questions are shown in Figure 5.1. These competency questions were used to identify the key concepts and to analyze existing domain ontologies.

[2]SEVENPRO, Semantic Virtual Engineering Environment for Product Design, was the project IST-027473 (2006-2008) funded under the 6th Framework Programme of the European Commission.

Retrieve all decision trees for dataset D.
Retrieve all rules for a dataset derived from dataset D with support > Y.
Retrieve all rules for class M.
Which algorithms were used to classify dataset D? What were their runtimes and accuracy?
Given dataset D and background knowledge B, which algorithms can be used to get a decision tree model?
Retrieve all workflows using algorithm A.
Retrieve all available discretization algorithms.
What is the influence of language bias on number of rules and efficiency?
What is the influence influence of kernel function on classification accuracy?
Compare different discretization methods for a particular dataset.
Can algorithm A be used on dataset D directly? If not, which preprocessing steps are necessary?
Which DM tasks have been performed on dataset D?
Retrieve all rules containing attribute F.
Retrieve all rules, which are not in background knowledge B.
What is the strength of interaction between attributes F and G?

Figure 5.1: Examples of the competency questions for the KD ontology

Furthermore the competency questions were translated from natural language to SPARQL queries to be used for ontology evaluation.

5.3 Knowledge elicitation

Since the primary purpose of our KD ontology is to support semi-automatic planning of knowledge discovery workflows, we focus on describing a collection of knowledge discovery tools necessary for our application scenarios: relational data mining algorithms based on inductive logic programming used in SEVENPRO project and in bioinformatics and algorithms available in data mining toolkits Orange and Weka [108].

To gain a broader view of the domain, we examine the existing domain ontologies, formalizations, and other attempts to construct a hierarchy of algorithms and data developed within other projects or found in data mining literature. We also examine the foundation ontologies DOLCE and BFO as candidates for providing the top level structure of our KD ontology.

5.3.1 Application scenarios

The main application scenario, which originally motivated our work, is discovering descriptive rules in relational data utilizing complex background knowledge from various sources. Solving this task requires a complex knowledge discovery workflow of interweaving inductive, deductive and format-conversion procedures. It is very difficult to construct this workflow manually without deep knowledge of the individual algorithms. The task is further complicated by the fact that the individual algorithms are often developed by

SEVENPRO Scenario 1: Relational descriptive rules

Task Generate *descriptive rules* from annotations of CAD drawings of different products. The descriptive rules characterizing *class* ABLiner are of particular interest. The *input data* consists of a list of *identifiers* of CAD drawings, CAD *ontology* expressed in *RDFS* and the annotations of the individual CAD drawings stored in *RDF/XML format*. The rules with *support* ≥ 0.3 and *confidence* ≥ 0.7 should be stored for querying parametrized by support, confidence, *rule length* and *attribute values* present. The information about individual *applications of the algorithm* including *runtimes* and discovered rules should be stored for querying.

Our solution We approached this task through *propositionalization* by generating *relational features* and subsequent propositional *rule learning*. The relational features were generated using an *ILP algorithm* based on *sorted refinement*. The algorithm was implemented in *Prolog* and requires as input *mode declarations*, *sort theory* and a set of *examples* described using *Prolog facts*. The mode declarations and sort theory were *extracted* by two simple procedures from the CAD ontology. The design annotations were *converted* into Prolog facts and used as *background knowledge* by the ILP algorithm. The generated features were *evaluated* with respect to their presence in each example of the *training set*. This information forms *attribute vectors* for subsequent propositional rule learning algorithm.

Figure 5.2: SEVENPRO Scenario 1: The scenario of generating relational descriptive rules. The key concepts identified in the scenario description are in italics.

different people or even at different institutions. A concrete example of a scenario solved within the SEVENPRO project is described in Figure 5.2. We used scenario descriptions created in SEVENPRO to identify key concepts and relations for our ontology.

Apart from such specialized complex task we wanted our approach to be applicable also for classical data mining scenarios, which can be realized within data mining toolkits. Therefore it was necessary to select some of these toolkits and examine how algorithms and data mining workflows are described and organized there. We chose Weka data mining toolkit, because it is the most commonly used toolkit and some of the algorithm implementations in Weka are considered as standard. As the second toolkit we chose Orange data mining toolkit. It contains workflow creation and execution functionalities and workflow description in XML. Moreover its extension Orange4WS enables using external algorithms available as web services within the workflow environment. These functionalities make Orange and Orange4WS a suitable testbed for our ontology.

In Weka each example in the dataset has to be represented by a vector of attribute values. In the workflows, which can be created manually in the Knowledge Flow environment, any linear part can consist of several preprocessing steps, one model building step, one evaluation step and one visualization step. The available preprocessing algorithms are further divided into supervised and unsupervised and each of these categories is divided into methods for attributes and instances. The algorithms for building models are divided into three types: classifiers, clusterers and associations. The classifiers are subdivided into bayesian, functions, lazy, meta, multi instance, trees, rules and miscellaneous. No criteria are specified for categorization of algorithms. We took a sample of about 20 algorithms covering all the above mentioned categories and used their descriptions for identification of key concepts required for characterizing the algorithms.

The core part of the Orange toolkit has similar restrictions on input data and workflows as Weka. We decided that the group of widgets for text mining integrated into Orange is out of scope of our ontology. The core algorithms are classified on two levels. On the first level the algorithms are divided into Association, Classification, Visualization, Regression, Data [transformation], Unsupervised and Evaluation. The second level of classification is provided, e.g., several discretization algorithms are grouped in the widget Discretize etc. Descriptions of over 60 algorithms available in Orange were used.

Some of the algorithms, e.g., C4.5 are implemented in both tools. This helped us to analyze the differences between algorithm specifications and implementation.

5.3.2 Relevant ontologies and formalizations

Examining the existing foundation and domain ontologies is important for ontology sharing and reuse. We start by discussing foundation ontologies and mid-level ontologies for domains related to knowledge discovery, such as ontology of plans, software and bioinformatics investigations. Then the existing ontologies and formalizations of knowledge discovery and the standards for web services are described. Finally we devote a separate section to the ongoing domain ontology development efforts.

Foundation and mid-level ontologies

Foundation ontologies most widely used as top-level ontologies for modeling in domains of computer science and bioinformatics are the Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) [72] and Basic Formal Ontology (BFO) [43].

DOLCE is a top-level ontology originally developed with a bias towards the domain of linguistic and cognitive engineering. It contains detailed and well-defined top level structure defining classes such as Endurant, Perdurant, Quality, Fact, etc. trying essentially to formalize descriptions of these terms used in philosophy. Similarly it defines core relations such as parthood, dependence, participation. The relations are defined using a large number of axioms, making DOLCE a rather heavyweight ontology.

Three extensions of DOLCE have been developed: the ontology of Description and Situation, Ontology of Information Objects, Core Ontology of Services formalizing the domain of web services and Ontology of Plans for the planning domain. The ontologies have a rich and complex structure, well defined and philosophically grounded concepts and relations. Their original version is independent on representation formalism, however OWL version of all ontologies is currently available. The ontology of plans is presented in [40]. However its objectives are explicitly stated as being an ontology for specification of social or cognitive plans and not characterizing the planning task and computationally executable plans and its reusability for our work is limited.

DOLCE and its extensions were reused in an effort to provide a well-founded reference ontology inspired by the domain of large software systems described in [80]. The paper presents a framework of ontologies containing Core Software Ontology, which forms the basis for Core Ontology of Software Components and Core Ontology of Web Services.

The Core Software Ontology defines concepts Data, Software, ComputationalObject, ComputationalActvitity and ComputationalTask as subclasses of concepts defined in DOLCE extensions and formalizes API description, workflow information and access rights and policies. Software is essentially defined as code of a program in some programming language and is a subclass of Data. Data is a subclass of InformationObject.

BFO was designed with the objective of providing a top-level ontology for domain ontologies developed for scientific research, such as the Ontology for Biomedical Investigations (OBI)[3]. It contains no concepts that would interfere with domain ontologies. Philosophical foundation of BFO is perspectivist realism, which essentially states that an ontology is a view of reality. Thus concepts which do not have instantiations in reality are not admitted. The core concepts are continuant ($\sim$ substance) and occurrent ($\sim$ process). Continuants are further divided into independent and dependent continuants. Only independent continuants can have qualities. Plans, algorithms and data are considered to be dependent continuants, since in BFO information entities exist in a way which makes them dependent on provenance and on processors.

As a spin off of the OBI development the Information artifacts ontology (IAO)[4] was developed using BFO as its top level ontology. IAO aims to be a mid-level ontology focusing on information content entities and processes that consume or produce them. IAO defines the concept of `information content entity`. Two of the subclasses of this concept are `data item`, which has the concept `dataset` among its subclasses, and `directive information entity`, which subsumes the concept of `plan specification`. `software`, `algorithm` and `programming language` are subclasses of `plan specification`. For `algorithm` two of its constituents are described using `has part` relation: `objective specification` and `action specification`. At the time of our KD ontology development several issues connected to modeling different types of knowledge such as hypotheses, propositions and patterns and how they should be linked to algorithms were still open.

Neither of the mid level ontologies describing algorithms and data mentioned above were fitting at their current state to our requirements and their parts relevant for our purposes were still under development. Bypassing the existing mid level ontologies and aligning the KD ontology directly to one of the foundation ontologies would require in depth analysis of design principles of the ontologies and their implications on complexity of reasoning required by the competency questions of the KD ontology. Such a task is beyond the scope of the current work. Therefore, while design patterns of DOLCE and BFO are taken into account in the development of the KD ontology, aligning of the KD ontology to one of these foundation ontologies will be the subject of future work.

Ontologies of Knowledge Discovery

There has been quite a lot of efforts to provide a taxonomy or ontology of the knowledge discovery domain, however to the best of our knowledge none of the existing ontologies cover relational data and background knowledge and support creation of complex

[3]The OBI Consortium http://purl.obolibrary.org/obo/obi

[4]$http://obofoundry.org/cgi-bin/detail.cgi?id=information_artifact$

workflows at the same time. For a systematic comparison of the existing approaches, we have defined the following set of criteria: purpose, key concepts, formal definition, executable language, use of top-level ontology, support of relational learning, support of workflows.

The ontologies of the knowledge discovery domain developed to support workflow creation are most closely related to our work, therefore we start with examining the systems using ontologies for this purpose.

The most relevant for our work is the IDEA system described in [9]. The system uses an ontology of DM operators for automatic workflow composition and ranking. The ontology provides a relatively detailed structure of the propositional DM algorithms and contains also heuristics for the ranking by various criteria (speed, accuracy, etc.). Workflow construction focuses on classical DM processes, which contain three subsequent steps: pre-processing, model induction and post-processing. In contrast, we address more complex, relational DM workflows with possibly multiple interleaved occurrences of steps pertaining to the three categories. The NExT system [8] is an extension of IDEA using OWL-S [35] ontology to describe the DM operators.

CAMLET [98] uses an ontology of algorithms and data structures. The CAMLET system defines a top-level control structure consisting of the following five steps: generating training and validation sets, generating a rule set, estimate data and rule sets, modifying a training data set and modifying a rule set. The individual steps form the top level components of the ontology of processes. The leaves are particular algorithm implementations. The ontology is limited to classification task. The inputs and outputs of the inductive learning processes are assumed to be data sets or rule sets. No ontology language used is mentioned.

The CITRUS system [107] is based on an extension of SPSS Clementine, which provides a visual interface for manual workflow construction. CITRUS uses an object oriented schema to model relationships between the algorithms.

Other efforts to provide a systematic formalization of the DM tasks include projects MiningMart [76], DataMiningGrid [96] and Universal Knowledge Grid [66]. Both MiningMart and DataMiningGrid contain a meta-model for representing and structuring information about data and algorithms, however, none of the meta-models is expressed in an ontology language.

MiningMart [76] is a KDD preprocessing tool specialized in data mining on large data stored in relational databases. The meta-model M4 is used to store data on two levels: the logic level describes the database schema and allows consistent access to information and the conceptual (ontology) level uses concepts with features and relationships to model data. The ontology captures all the data preprocessing therefore gives a better understanding and reuse of the data. However this meta-model is expressed in UML/XML and not in an ontology language.

The objective of DataMiningGrid[96] is to provide tools and services, which facilitate making data mining applications available in the grid environment. An Application Description Schema (ADS) was developed withing the project. ADS is centered on data mining application and contains information about CRISP-DM phase the application is used for and the data mining task to which the application applies. It also describes the data mining method that the application employs, the data mining algorithm and its particular implementation. An application must be described using ADS to be registered in the grid. ADS description is then used for search, dynamic configuration of GUIs, resource brokering, etc.

Universal Knowledge Grid [66] aims at developing an ontology-based grid architecture. The key concepts include: `Data_mining`, `Function`, `Type_of_data`, `Algorithm`, `Application_domain`, `Sub_domain`, `Application_task`, `Solution` and `Predictive_model`. The proposed ontology only contains one level of algorithm class hierarchy. The paper states that data can be structured, but provides no details on modeling the data. Definition of concepts appear to be overly tailored to a particular use case of the ontology, for example an `Application_task`, e.g., "to characterize unusual access sequences" is a subclass of `Application_domain`, e.g., fraud detection. The ontology is not used for planning or automated workflow composition, but rather for annotation of algorithms available within the knowledge grid and tasks for which the algorithms have been or can be applied. The intelligent composer does not produce workflows but transforms natural language queries into semantic queries.

Ontologies for web services and workflows

Since many of the knowledge discovery algorithms are becoming available as web services, we examine the existing frameworks for semantic markup of web services. There is currently no universally accepted framework. The most visible ongoing development is on Web Service Modeling Ontology (WSMO) and OWL-S.

WSMO [61] contains four top level components: ontologies, goals, web services, mediators. Focus of WSMO is on integration and mediation between agents with different concept models. Thus its central topic is out of scope of our requirements for workflows ontology. WSMO is implemented in Web Service Modeling Language (WSML), which is based on F-logic. However, a mapping to OWL exists. The goals capture only results and no pre- or post-conditions. On the other hand WSMO supports goal combination, while in OWL-S combination of goals is possible only using subclassing. WSMO does not provide any specification of control flow and data flow, which is specified by OWL-S composite process. WSDL grounding in OWL-S is more detailed than in WSMO.

The principal aim of OWL-S [35] is not to define a rich reference ontology, but rather to enable the users and software agents to to discover, invoke, compose, and monitor web services. The ontology has three main parts: the service profile, the process model and the grounding. Profile model duplicates the descriptions embodied in the rest of WSDL (namely inputs and outputs). OWL-S does not separate what the user wants from what the service provides. The Profile ontology is used both as an advertisement for the service and as a request to find a service and it is assumed that there is only one way to interact with the service. Thus mediators are not handled as special components. The process model gives a detailed description of a service's operation and the grounding provides details on how to interoperate with a service. This scope and approach better matches requirements on workflows subontology, therefore, despite some design shortcomings of OWL-S [75], it is a more suitable ontology to serve as a basis for our workflow ontology. Also exists work on mapping OWL-S to/from WS-BPEL [3], [12], thus workflows described in OWL-S can be executed on workflow engines operating on WS-BPEL.

Ongoing domain ontology development efforts

There are currently two ongoing projects, which aim to provide a framework for semi-automatic composition of DM/KD workflows using an ontology of the DM/KD ingredients: e-Lico[5] and KDDVM[6].

KDDVM project aims to construct a service-oriented knowledge discovery support system. It proposes a service-based architecture containing three levels of services [28]: the algorithms involved in the KDD process, support services, e.g. wrapping, workflow manager and service broker, and advanced services including a database for storing the information about KD processes and a service for workflow composition relying on an ontology of KD algorithms. The representation of information about the KD processes was still an open issue at the time when we started developing our ontology. An ontology of algorithms developed within this project and its use for automatic workflow composition are described in [30] and [29]. KDDOnto is a light weight ontology focused on classical propositional DM models. The explicitly defined taxonomy of algorithms starts with categorizing the algorithms according to phases of the KDD process. Description of data is limited to and ends on the level of dataset.

The within the e-Lico project two ontologies of DM ingredients are being developed: DMOnto focused on use in algorithm selection and meta-mining [47] and an ontology for automatic workflow construction using planning [55]. The ontology proposed in [55] is strongly biased towards the planning task and more specifically towards checking of validity of the workflows using preconditions and effects of the algorithms. Its core part is implemented in OWL, however the preconditions and effects are described using SWRL with some extensions. The planning thus relies on a special reasoner. The ontology does not provide means for operator nesting an annotation of workflows and information about their execution and results. The representation of data is limited to datasets consisting of data tables.

DMOnto aims to provide a more detailed description of the datasets, models and methods, which could be exploited in algorithm selection. It describes the algorithms from two points of view: representation bias given by the model structure and preference bias given by cost function and representation strategy. However the version of DMOnto described in [47] has quite strong representation bias towards classification and views learning as optimization of some model parameters. This approach would be difficult to extend to descriptive data mining and rule induction, which form one of the important areas resulting from the competency questions.

In parallel to our work, the OntoDM [83] ontology is being developed on the basis of [34] and attempts to capture minimum information required for describing data mining investigations. A principled top-down approach was adopted to the development of OntoDM aiming at its maximum generality. OntoDM uses BFO as top-level ontology and OBO Relational Ontology to define relations and is fully aligned with top level structure of OBI. It uses concepts from draft version of IAO (Information Artifact Ontology) for dealing

[5]www.e-lico.eu

[6]http://boole.diiga.univpm.it/

with information. Numerous relations between information artifacts are described using a single property hasInformation. The concept of an algorithm is modeled on three levels: information content entity, process, and realizable entity, thus separating specification, implementation and application. For modeling workflows the process aspect of an algorithm could be used. However information about inputs and outputs is given only for application, while for workflow construction we need to reason with types of inputs/outputs at specification level. Also OntoDM is not compatible with OWL-S or other existing workflows or web services formalizations. Examining the possibilities of mapping between OntoDM and our KD ontology will be subject to future work, however, as has already been stated, at the time when we were developing the KD ontology, OntoDM was not sufficiently refined for purposes of workflow construction [82].

5.4 Knowledge Discovery Ontology

Since none of the available domain ontologies discussed above cover our requirements, we have decided to develop our own KD ontology. We follow up on the presently emerging research attempting to establish a *unifying theory of data mining* [109, 34]. This research did not reach the form of a formal representation, yet it helped us to design the core parts of the ontology, namely the concepts of knowledge, representation language, pattern, dataset, evaluation, and further specialized concepts.

The top level structure of the ontology is inspired by the framework for data mining proposed in [34]. In this framework three basic concepts of data mining are identified: 'data', 'patterns and models' and 'data mining task'. Following this view, our three core concepts are: *knowledge*, capturing the declarative elements in knowledge discovery, *algorithms*, which serve to transform knowledge into (another form of) knowledge, and *knowledge discovery task*, which we have extended to involve workflows.

The ontology is implemented in the description logic variant of the semantic web language OWL-DL [84]. Our primary reasons for this choice were OWL's sufficient expressiveness, modularity, availability of ontology authoring tools and optimized reasoners.

An illustrative part of the top level structure of the KD ontology is shown in Fig. 5.3. The whole ontology contains more than 150 concepts and 500 instances. The three core concepts are marked in yellow and we shall discuss them in the following sections.

5.4.1 Knowledge

Any declarative ingredient of the knowledge discovery process such as datasets, constraints, background knowledge, rules, etc. are instances of the `Knowledge` class. Fig. 5.4 shows an illustrative part of the class hierarchy of knowledge types.

In data mining, many knowledge types can be regarded as sets of more elementary pieces of knowledge [34]. For example, first-order logic theories consist of formulas. Similarly, the common notion of a *dataset* corresponds either to a set of attribute-value n-tuples or to a set of relational structures, each of which describes an individual object. This lead us to formalizing the notion of complex types of knowledge and their composition from atomic knowledge types. This structure is accounted for through the property `contains`,

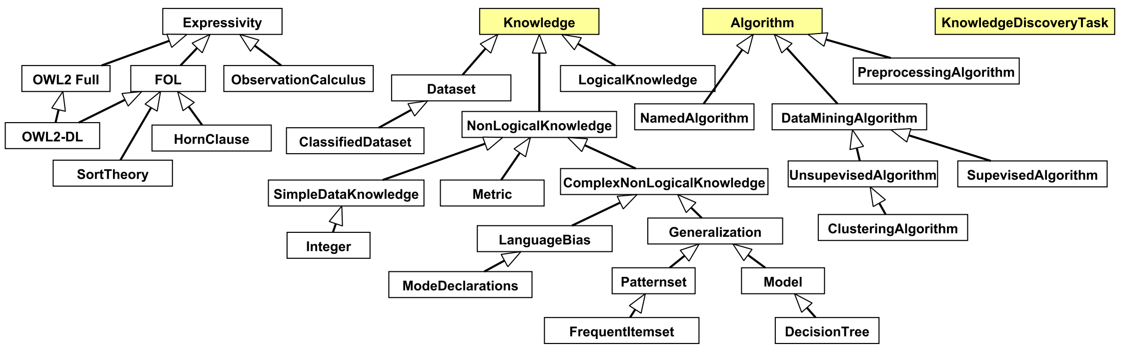


Figure 5.3: An illustrative part of the top level structure of the KD ontology (the whole ontology contains more than 150 concepts and 500 instances). Subclass relations are shown through solid arrows.

e.g. a first order theory `contains` first order formulas, a dendrogram `contains` clusters.

Moreover, some knowledge types may be categorized according to the expressivity of the language in which they are encoded. For this purpose we have designed a hierarchy of language expressivity, of which Fig. 5.5 shows a fraction. The hierarchy is an acyclic directed graph, however, for better readability only a tree structure is shown in Fig. 5.5. In designing the ontology we were aiming to avoid multiple inheritance as far as possible, however the currently used definitions of knowledge representation languages do not follow a single inheritance hierarchy, e.g., OWL-DL is a subset of first order logic, but it is also a subset of OWL-FULL.

In literature and most of the existing ontologies the knowledge types are mainly defined by the role they play in the knowledge discovery process. Therefore we also distinguish certain knowledge types, which play special roles in knowledge discovery, such dataset, example, pattern, model, etc.

These guiding principles lead us to define the following three key subclasses of the `Knowledge` class: `Dataset`, `LogicalKnowledge` and `NonLogicalKnowledge`.

Most general definition of dataset is that it is a collection of data. Therefore, the `Dataset` class is defined as `Knowledge`, which contains `Example`s, or more formally:

$$\texttt{Dataset} \equiv \texttt{Knowledge} \sqcap \exists\, \texttt{example} \cdot \texttt{Example}$$

The concept of logical knowledge was introduced to describe knowledge types, which are encoded in a particular expressive language, such as first order logic or OWL-FULL. Thus expressivity is a defining feature of the `LogicalKnowledge`:

$$\texttt{LogicalKnowledge} \equiv \texttt{Knowledge} \sqcap \exists\, \texttt{hasExpressivity} \cdot \texttt{Expressivity}$$

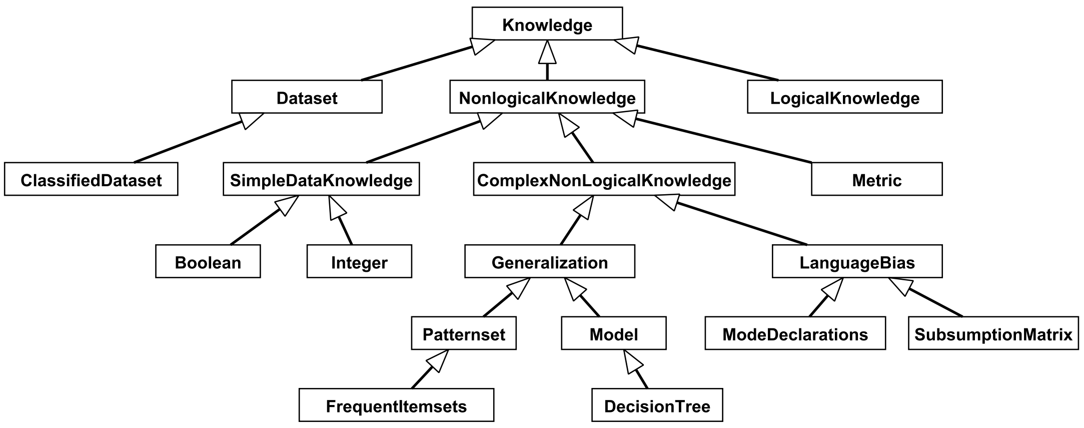


Figure 5.4: Part of the top level structure of the knowledge type part of the ontology with subclass relations shown through arrows.

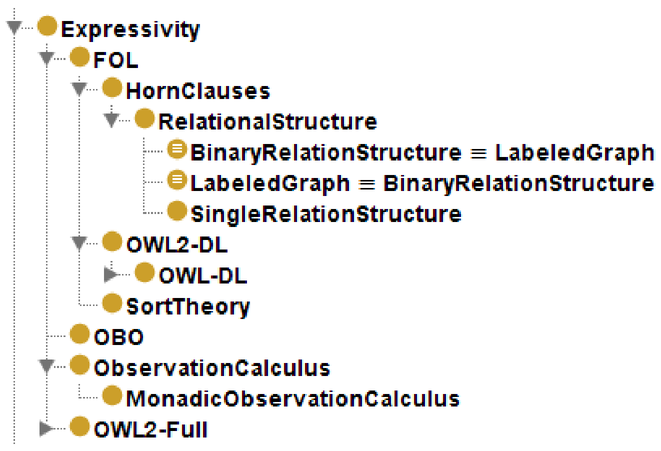
The `hasExpressivity` property can be also applied to datasets to distinguish between propositional datasets and relational datasets. A classified relational dataset `ClassifiedRelationalDataset` can be defined as:

$$\begin{aligned}\texttt{ClassifiedRelationalDataset} \sqsubseteq\ & \texttt{Dataset}\\ \sqcap\ & \exists\,\texttt{classAttribute}\cdot\texttt{CategoricalAttribute}\\ \sqcap\ & \forall\,\texttt{example}\cdot\texttt{ClassifiedExample}\\ \sqcap\ & \forall\,\texttt{hasExpressivity}\cdot\texttt{RelationalStructure}\end{aligned}$$

All the other types of knowledge such as pattern sets, models and constraints (e.g., language bias) are clustered under the class `NonLogicalKnowledge`. It contains the essential concept of a `Generalization`. A generalization is a knowledge class with the special property that it *defines a mapping* from one or more knowledge classes to another knowledge class. Intuitively, this class serves to hold the results of inductive mining algorithms; such results generally can be viewed in a unified fashion as mappings [34]. Of course, the generalization's mapping, i.e. its semantics, is ultimately assigned to it by an *algorithm* used to interpret it. No information about the algorithm is build into the `Generalization` class.

The `Generalization` class currently contains two subclasses, which can be distinguished by the property of decomposability and also by the type of algorithms used to produce it. `Model` is a result of a predictive algorithm and it cannot be decomposed into independent parts, e.g. a decision tree. `Patternset` on the other hand can be decomposed into independent parts and is usually produced by a descriptive algorithm, such as an association rules learner.

We shall describe the association rules example in more detail, since it illustrates



```
Expressivity
    FOL
        HornClauses
            RelationalStructure
                BinaryRelationStructure ≡ LabeledGraph
                LabeledGraph ≡ BinaryRelationStructure
                SingleRelationStructure
        OWL2-DL
            OWL-DL
        SortTheory
    OBO
    ObservationCalculus
        MonadicObservationCalculus
    OWL2-Full
```

Figure 5.5: A part of the expressivity hierarchy shown in the Protégóntology editor. Expressivity is an essential part of definition of the `LogicalKnowledge` class.

both the decomposability and modeling of results of data mining. Association rule set is defined as a patternset consisting of association rules:

$$\begin{aligned}\texttt{AssociationRules} \sqsubseteq\ & \texttt{Patternset}\\ \sqcap\ & \forall\texttt{contains}\cdot\texttt{AssociationRule}\end{aligned}$$

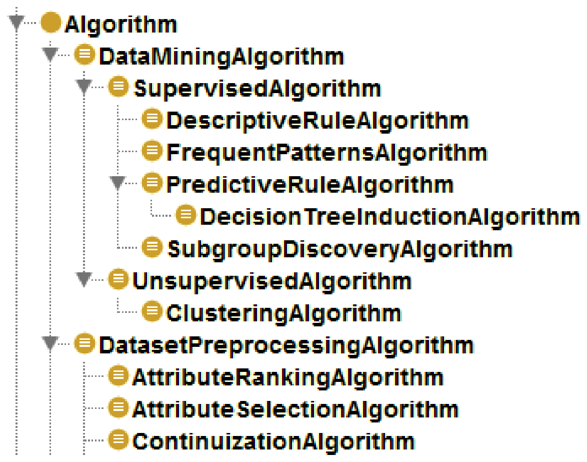
A single association rule is defined as atomic knowledge composed of an antecedent and a consequent:

$$\begin{aligned}\texttt{AssociationRule} \equiv\ & \texttt{AtomicKnowledge}\\ \sqcap\ & \exists\texttt{antecedent}\cdot\texttt{And}\\ \sqcap\ & \exists\texttt{consequent}\cdot\texttt{And}\\ \sqcap\ & \exists\texttt{quantifier}\cdot\texttt{Quantifier}\end{aligned}$$

where `And` is a conjunction of boolean atoms and `Quantifier` is an evaluation measure, e.g. support, confidence.

5.4.2 Algorithms

The notion of an algorithm involves all executable routines that can be used in a knowledge discovery process, like inductive algorithms and knowledge format transformations. Any algorithm turns a knowledge instance into another knowledge instance. For example, inductive algorithms will typically produce a `Patternset` or `Model` instance out of a `Dataset` instance. Of importance are also auxiliary representation changers, transforming datasets to other datasets. These may be simple format converters (e.g. only changing the separator character in a textual data file), or more complex transformations characterized



- Algorithm
  - DataMiningAlgorithm
    - SupervisedAlgorithm
      - DescriptiveRuleAlgorithm
      - FrequentPatternsAlgorithm
      - PredictiveRuleAlgorithm
        - DecisionTreeInductionAlgorithm
      - SubgroupDiscoveryAlgorithm
    - UnsupervisedAlgorithm
      - ClusteringAlgorithm
  - DatasetPreprocessingAlgorithm
    - AttributeRankingAlgorithm
    - AttributeSelectionAlgorithm
    - ContinuizationAlgorithm

Figure 5.6: A part of the algorithms hierarchy shown in the Protégóntology editor.

by information loss. This may be incurred either due to a conversion into a language class with lower expressiveness (e.g. for 'propositionalization' algorithms [106]) or even without expressiveness change (e.g. for principal component representation of real vectors).

The `Algorithm` class is a base class for all algorithms, like Apriori or JRip (algorithm for decision rule induction implemented in Weka data mining platform), in the example below. The of algorithms hierarchy contains some fully defined classes, like `FrequentPatternsAlgorithm` or `PredictiveRuleAlgorithm` for a fine-grained categorization of data mining algorithms according to their functionality. An illustrative part of this hierarchy is shown in Fig. 5.6. This explicit hierarchy allows for the formulation of additional user constraints on the workflows. For example, constraints can refer to some particular category of data mining algorithms, e.g. DiscretizationAlgorithm, FormatChangingAlgorithm, ClusteringAlgorithm, etc.

Each algorithm configuration is defined by its input and output knowledge specifications and by its parameters. In order to maintain the compatibility with OWL-S, the `Algorithm` class is defined as an equivalent of the OWL-S class `Process` and the algorithm configuration is an instance of its subclass `NamedAlgorithm`. Both the input knowledge and the parameters are instances of `AlgorithmParameter` and defined using the `input` property. The output knowledge specifications are instances of `AlgorithmParameter` and defined using the `output` property. The parameter instances are then mapped to the appropriate `Knowledge` subclasses using the `isRangeOf` property.

In the current version the ontology contains only a limited reference to the actual algorithm implementation within some toolkit or as a web service. Each named algorithm can be linked to its implementation using the `stringRepresentation` property, which is aimed at automatically running the generated workflows within a knowledge-discovery engine in a uniform way. The value of `stringRepresentation` can contain a reference to an algorithm executable or to a service location.

As an example of annotation of an algorithm we present the definition of the JRip algorithm in the description logic notation using the extended ABox syntax [5]:

$$\begin{aligned} \{\texttt{JRip}\} \sqsubseteq\ & \texttt{NamedAlgorithm} \\ \sqcap\ & \exists\,\texttt{output} \cdot \{\texttt{JRip-O-PredictiveRules}\} \\ \sqcap\ & \exists\,\texttt{input} \cdot \{\texttt{JRip-I-Dataset}\} \\ \sqcap\ & \exists\,\texttt{input} \cdot \{\texttt{JRip-I-Pruning}\} \\ \sqcap\ & \exists\,\texttt{input} \cdot \{\texttt{JRip-I-MinNo}\} \end{aligned}$$

$$\begin{aligned} & \texttt{JRip-I-Dataset-Range} \\ \equiv\ & \exists\,\texttt{isRangeOf} \cdot \{\texttt{JRip-I-Dataset}\} \\ \equiv\ & \texttt{ClassifiedDataset} \\ \sqcap\ & \forall\,\texttt{hasExpressivity} \cdot \\ & \texttt{SingleRelationStructure} \\ \sqcap\ & \forall\,\texttt{hasFormat} \cdot \{\texttt{ARFF,CSV}\} \end{aligned}$$

The JRip algorithm is defined as an algorithm that has two parameters: one stipulating whether to use pruning and one determining the minimum number of examples covered by each single rule. It can be applied to a single relation classified dataset in the CSV or ARFF format and produces a result in the form of predictive rules (patterns defining a mapping to a distinguished set of classes).

A particular execution of a given algorithm configuration on a given dataset is an instance of `AlgorithmExecution`. It contains information about algorithm configuration with instantiated input and output data. Information about time taken for the execution is stored using `cpuTimeSeconds`.

5.4.3 Knowledge Discovery Task

In order to formalize the knowledge discovery task description and for storing the created workflows in a knowledge-based representation, we have created a small ontology for workflows, which extends the KD ontology. The workflows subontology has two central notions: `KnowledgeDiscoveryTask` and `Workflow`.

Each `KnowledgeDiscoveryTask` is defined by its `init` and `goal` specifications. As an example we present the definition of the problem of generating predictive rules from relational data (RRules) in the description logic notation:

$$\begin{aligned} \texttt{RRulesTask} \sqsubseteq\ & \texttt{KnowledgeDiscoveryTask} \\ \sqcap\ & \exists\,\texttt{goal} \cdot \texttt{PredictiveRules} \\ \sqcap\ & \exists\,\texttt{init} \cdot (\texttt{ClassifiedDataset} \sqcap \\ & \forall\,\texttt{hasExpressivity} \cdot \texttt{RelationalStructure} \sqcap \\ & \forall\,\texttt{hasFormat} \cdot \{\texttt{RDFXML}\}) \\ \sqcap\ & \exists\,\texttt{init} \cdot (\texttt{LogicalKnowledge} \sqcap \\ & \forall\,\texttt{hasExpressivity} \cdot \texttt{OWL-DL} \sqcap \\ & \exists\,\texttt{hasFormat} \cdot \{\texttt{RDFXML}\}) \end{aligned}$$

RRules problem is defined as a problem of generating relational predictive rules from a relational classified dataset and an ontology in OWL-DL as background knowledge, both expressed in the RDFXML format. Currently the ontology describes a few problem types, which were developed for our use cases and which should serve as a template for users to specify problem types relevant for their knowledge discovery tasks.

An abstract workflow is represented by the `Workflow` class, which is a subclass of the `Algorithm` class in the KD ontology. This allows us to encapsulate the often repeated workflows and construct hierarchical workflows. `Workflow` is defined as a set of `Action`s specified using `hasAction` property. An action is defined by the `hasAlgorithm` property, specifying the algorithm configuration used by this action, and by `startTime` specifying the step within the plan in which the action should be carried out. The dependencies between actions are represented using the `predecessor` property. The formal representation of abstract workflows is used for workflow instantiation and execution within the knowledge discovery engine and also for storing and reuse of the generated workflows.

To utilize the worflow description capabilities of OWL-S, a bridge between KD ontology and OWL-S was designed:

`kd:Algorithm` ≡ `Process`

`kd:NamedAlgorithm` ≡ `AtomicProcess`

`kd:Workflow` ≡ `CompositeProcess`

`kd:AlgorithmInput` ≡ `Input`

`kd:AlgorithmOutput` ≡ `Output`

For annotation of links between the algorithms and annotation of workflow inputs and outputs OWL-S is used directly. `CompositeProcess`, which is equivalent to `kd:Workflow`, consists of instances of `Perform` arranged using an instance of some subclass of `ControlConstruct`, e.g. `Sequence`. `Perform` is a kind of a wrapper for algorithm, which enables us to reference the algorithms within the workflow and thus define links between inputs and outputs of algorithms. The link between output of one algorithm and input of another algorithm is implemented using `Binding`. Currently only `Sequence` control construct is used, i.e. possible parallelization is not explicitly declared using OWL-S constructs, it is only indicated by bindings between algorithms.

Having introduced our KD ontology and its extension covering knowledge discovery tasks and workflows, we can proceed to discussion of the KD workflows themselves and their automatic construction.

Chapter 6

Knowledge discovery workflows

Using the KD ontology, specific classes of algorithms can be annotated according to their functionality. For example, inductive algorithms (given a particular pattern evaluation function) will produce patterns out of datasets, format conversion algorithms will produce datasets out of datasets etc. The ontology implicitly delimits the variability of possible workflows for a given task. For example, if the user desires to mine patterns in the language L of propositional formulas, any algorithm may be employed that is annotated as to produce patterns in L or in any language subsumed by L (e.g. propositional conjunctions).

In this chapter we investigate how algorithms can be assembled into complex knowledge discovery workflows automatically with the use of a knowledge discovery ontology and a planning algorithm accepting task descriptions automatically formed using the vocabulary of the ontology.

Our original work was motivated mainly by complex relational data mining tasks such as discovery of rules to distinguish between two types of leukemia based on gene expression and gene annotations using terms of the Gene Ontology[1]. In previous studies in bioinformatics conducted by some members of our group [101], knowledge discovery workflow of interweaving inductive, deductive and format-conversion procedures had to be constructed manually. We use the mentioned bioinformatics study as a use case throughout this chapter. To demonstrate the generality of the proposed approach, we also test our methodology in the domain of engineering[2], where knowledge discovery workflows exhibit features similar to scientific workflows [105], namely their complexity and their inductive character. Since specialized algorithms are needed for these tasks, the number of alternative workflows, which can be produced, is quite small. Therefore for more extensive evaluation tasks including classical propositional data mining algorithms available within the Orange toolkit are included as well.

Our methodology focuses on automatic construction of abstract workflows consisting of data mining algorithms. The mapping to concrete computational resources, particular data sets and algorithm parameters are not taken into account during workflow construc-

[1]www.geneontology.org

[2]Specifically within the project SEVENPRO, Semantic Virtual Engineering Environment for Product Design, IST-027473 (2006-2008), 6th Framework Programme of the European Commission.

tion. Each generated workflow is stored as an instance of the `Workflow` class and can be instantiated with a specific algorithm configuration either manually or using a predefined default configuration. We treat the automatic workflow construction as a planning task, in which algorithms represent operators and their input and output knowledge types represent preconditions and effects.

As a baseline approach we have decided to develop a planner based on the Planning Domain Definition Language (PDDL) [92] standard for planning problem description and a procedure for converting task and algorithm descriptions available in the KD ontology into PDDL. Then we investigated a tighter integration of ontological reasoning and planning. We implemented a planning algorithm capable of directly querying the KD ontology using a reasoner, which we called PelletPlanner.

When we extended the KD ontology with annotations of algorithms available in the Orange toolkit, we encountered the problem of having sets of algorithms, which on the basis of their inputs and outputs subsume each other or are even equivalent. For tasks such as inducing association rules from a propositional dataset, this led to producing a large number of workflows, a lot of which were very similar. In this work we alleviate this problem by developing an enhanced version of PelletPlanner capable of exploiting the algorithm subsumption hierarchy.

6.1 Related Work

The existing knowledge discovery platforms, which use a formal representation of the knowledge discovery task and offer some means of automated workflow composition, were introduced in Chapter 5. Therefore in this section we only examine their workflow composition techniques.

IDEA [9] and NExT [8] create workflows for classical data mining processes containing three subsequent steps: pre-processing, model induction, and post-processing. Both systems provide the user with a ranked list of alternative workflows, however the proposed workflows are linear, whereas our workflows are directed acyclic graphs. IDEA does not use any planning technique, but simply enumerates all workflows valid for a particular task. The NExT system uses the planning system XPlan [58].

The CAMLET [98] system defines a top level control structure consisting of the following steps: generating training and validation sets, generating a rule set, estimate data and rule sets, modifying a training data set and modifying a rule set. The individual steps form the top level components of the ontology of processes. The leaves are particular algorithm implementations. The ontology is limited to the classification task. Genetic programming is used to create workflows, which are instantiations of some sub-structure of the defined top level control structure with particular algorithms. The workflows are evaluated with respect to accuracy on a given dataset and thus execution of the generated workflows is an integral part of the workflow construction process. The inputs and outputs of the inductive learning processes are assumed to be data sets or rule sets. I/O type compatibility is checked during the compilation phase.

KDDVM project uses the KDDOnto for automatic workflow composition [30, 29]. The workflows are constructed by a backward chaining planning based only on the algorithm inputs and outputs, not utilizing the algorithms hierarchy. The system provides means for approximate matching of inputs and outputs based on ontological distance. The user can impose further constraints, which are external to the ontology, such as not-with, not-before and also maximal number of steps. Some inputs can be specified as optional - precondition relaxation. The workflow ranking is based on exactness of matching, precondition relaxation and performance evaluation. For performance evaluation only complexity class of the algorithm is used. The workflows are limited to classical data mining with clearly delimited phases of the data mining process.

Within the e-Lico project a planning algorithm based on Hierarchical Task Network (HTN) planning [89] is used for workflow construction for classical data mining tasks [55]. The planning algorithm uses a specialized reasoner for checking preconditions and effects of algorithms, which are described using SWRL with some extensions. HTN planning relies on an explicitly defined task decomposition hierarchy. Since the complex relational data mining tasks motivating this work do not have a clear tree-based task decomposition hierarchy, we have decided not to use HTN planning at this stage.

Another system for automatic workflow construction using a knowledge discovery ontology is described in [14], however this work is focused only on automatic formation of linear sequences of tasks.

Workflow composition is used in some distributed data mining frameworks to facilitate parallelization, data integration and demonstrate feasibility of service-oriented data mining [2, 20, 46, 60]. WS composition for DDM [2] is based on a design pattern library. The created workflow is exported in XML and then executed within Triana. Service-Oriented DDM [20] uses the principle of learning from abstractions. It is based on BPEL4WS and one of its main contributions is to handle datasets partitioned and physically distributed on different data sources. Anteater [46] also deals with DM algorithms distributed on different servers. Its main objectives are to harness parallel implementation of DM algorithms to handle large volumes of data and decrease computational costs. It uses an XML metadata model to describe algorithms and data. DDM Framework [60] focuses on integration of intermediate results and service-oriented implementation of algorithms. However algorithms are described in natural language only, not formally using an ontology.

Also closely relevant to our work are the existing approaches to web service composition, which use a planning algorithm and an ontology for domain description. Similarly to our approach, [91], [58] and [68] translate an OWL description to a planning formalism based on PDDL. While approaches presented in [58] and [68] use classical STRIPS [38] planning, in [91], HTN planning is employed.

Planning directly in description logics is addressed in [48]. Currently the algorithm can only deal with DL-Lite descriptions with reasonable efficiency.

We make a step beyond the work presented in [68] and [58], where a reasoner is used in the pre-processing phase, by investigating the possibility of integrating a reasoning engine directly with the planner. Moreover, our procedure for converting the task descriptions to PDDL does not rely on OWL-S, therefore we do not require the involved algorithms to be implemented as web services.

6.2 Automatic Workflows Construction

We propose to treat the automatic workflow construction as a classical planning task, in which algorithms represent operators and their required input and output knowledge types represent preconditions and effects. Both the information about the available algorithms and knowledge types as well as the specification of the knowledge discovery task is encoded through an ontology. At the same time we want to be compatible with established planning standards. For these reasons we have decided to explore the approach of generating a description of the domain and the problem description in the PDDL language [92] using elements of the KD ontology and implementing a planning algorithm, which uses PDDL descriptions.

6.2.1 Generating domain and problem descriptions in PDDL

We use PDDL 2.0 with type hierarchy and domain axioms. Planning algorithms require two main inputs. The first one is the description of the domain specifying the available types of objects and actions. The second one is the problem description specifying the initial state, goal state and the available objects. We have implemented a procedure for generating the domain description from the KD ontology.

The domain description is generated by converting `NamedAlgorithm`s into PDDL actions, with `input`s specifying the preconditions and `output`s specifying the effects. Both inputs and outputs are restricted to conjunctions of OWL classes. We consider only those inputs that are specified by instances of classes disjoint with `SimpleDataKnowledge`, which is used to represent algorithm parameters. Since PDDL can handle only named types and their hierarchies, it is necessary to pre-process classes defined using owl:Restriction.

A skeleton of the conversion procedure is presented in Fig. 6.1. Both the list of instances of `NamedAlgorithm` and the list of input and output specifications are obtained by a SPARQL-DL query. Procedure **transformIO** converts an i/o specification defined by an instance of `AlgorithmParameter` into a class equivalent to its range, which consists of an intersection of the `Knowledge` subclasses and restrictions defined in the KD ontology. The equivalent class is again obtained by a SPARQL-DL query. The procedure returns null for subclasses of `SimpleDataKnowledge` representing algorithm parameters.

The procedure **convertIO2pddl** converts an i/o specification defined by a named class or an owl:intersectionOf class into PDDL predicates. The individual operands of the owl:intersectionOf class specified by named classes and universal restrictions on properties `contains` and `hasExpressivity` are converted into named classes. The named classes are added to the *uknow* list and their hierarchy is later inferred by an OWL reasoner. The named class is converted to a unary predicate `available` and also added to action parameters. Operands specified by restrictions on other properties are converted using the procedure **rest2preds** to binary predicates with the first argument being the previously defined named class and the second argument is given by the restriction value. All the generated predicates and parameters are then added to action preconditions or effects using **addIO**.

As an example of how algorithm is represented represented using an action in PDDL, we present the definition of the action generated from the JRip, the algorithm for

```
$uknow$ = []; used subclasses of Knowledge
$actions$ = []; created PDDL actions
$types$ = []; hierarchy PDDL types
onto2pddl():
 classify KD ontology;
 $algs$ := {instances of NamedAlgorithm};
 for ( $al$ : $algs$)
      $act$ := new Action($al$.name);
      $iospecs$ := {input and output specifications};
      for ($ios$ : $iospecs$)
          $eqios$ = transformIO($ios$);
          if ($eqios$ == null) continue;
          $act$.addIO(convertIO2pddl($equios$, $uknow$,
              $act$.varnames));
      $actions$.add($act$);
 add $uknow$ classes to KD ontology and classify;
 $types$ := classes2types($uknow$);
 return createDomainPDDL($actions$,$types$);
$preds$ = []; a list of predicates describing the io specification
$params$ = []; a list of variables used in the io specification
convertIO2pddl($ios$,$uknow$,$varnames$):
 if ($ios$.isNamedClass())
      return {{available($v_i$)}, {$v_i$ - $ios$.name}};
 else if ($ios$.isIntersectionClass())
      $rest4cls$ = {operands of $ios$ represented by
          named classes or restrictions
          on hasExpressivity and contains};
      $rest4pred$ = {operands of $ios$ represented by
          restrictions on other properties};
      $comp$ = createCompositeClass($rest4cls$);
      $uknow$.add($comp$);
      $params$.add($v_j$ - $comp$.name);
      $preds$.add(available($v_j$));
      rest2preds($rest4pred$,$v_j$,$preds$,$params$);
 return preds, params;
```

Figure 6.1: A skeleton of the procedure for converting descriptions of algorithms from the KD ontology into PDDL.

the predictive rule learning described in Section 5.4 :

```
(:action JRip
 :parameters (
  ?v0 - Dataset_hasExpressivity
   _SingleRelationKnowledge
  ?v1 - CSV
  ?v2 - PredictiveRules )
 :precondition (and (available ?v0)
                    (format ?v0 ?v1))
 :effect (and (available ?v2)))
```

The information about the output of JRip algorithm is expressed using the class `PredictiveRules`. Therefore the effects of the action using the JRip algorithm are represented using the unary predicate `available` applied on the class `PredictiveRules`.

Finally, procedure **createDomainPDDL** takes the list of actions and hierarchy of PDDL types and fits them into a domain file template in PDDL.

Problem description in PDDL is generated in a very similar way, except we are dealing with objects instead of variables. The objects appearing in `init` and `goal` conditions are generated from an individual of type `KnowledgeDiscoveryTask` in the KD ontology, which represents a particular problem e.g. producing a set of predictive rules from a dataset stored in a relational database.

6.2.2 Planning algorithm

We implemented a planning algorithm based on the Fast-Forward (FF) planning system [49] to generate abstract workflows automatically. The Fast-Forward planning system uses a modified version of a hill climbing algorithm called *enforced hill climbing* to perform forward state space search. Enforced hill climbing is based on the commonly used hill-climbing algorithm for local search, however in case there is no strictly heuristically better successor in the immediate neighborhood of the current state, breadth-first search is used to find a sequence of actions leading to a heuristically better successor. The basic architecture of the FF system is shown in Fig. 6.2.

The heuristics used by the enforced hill-climbing algorithm is defined as the number of operators in the plan constructed using relaxed GRAPHPLAN [11]. The relaxed planning task essentially ignores delete lists of actions, i.e. effects of the actions, which make some previously valid preconditions no longer valid. In [49] a proof is presented, that GRAPHPLAN solves the relaxed task in polynomial time.

In [49] the search space is pruned using two heuristics: a helpful actions heuristic, which considers only actions that add at least one goal at the first time step, and added goal deletion heuristics, which exploits goal ordering. If the enforced hill-climbing algorithm fails, the problem is solved using a complete search algorithm.

We implemented the basic architecture of the Fast-Forward planning system consisting of the enforced hill climbing algorithm and the relaxed GRAPHPLAN in our algorithm, which we call PDDLPlanner. Since our current formulation of the knowledge

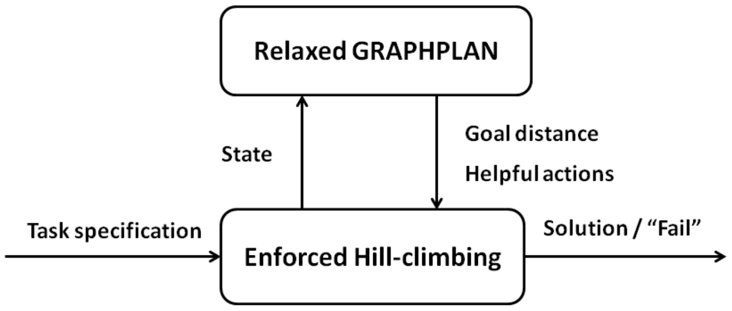


Figure 6.2: Base system architecture of the Fast-Forward (FF) planning system as it is characterized in [49]

discovery workflow construction contains no goal ordering, no mechanisms for exploiting goal ordering are implemented. In order to produce several relevant workflows in a reasonable time frame, the algorithm is run repeatedly, randomly permuting the order in which immediate neighbors of one state are added to the open-set during the breadth-first search.

The PDDLPlanner works with description of planning task and all available actions in a standard PDDL file. Our current implementation is capable of handling PDDL only with STRIPS [38] expressivity and using types. The description of planning task and the available algorithms using KD ontology is converted to PDDL in pre-processing stage. The KD ontology is then not used throughout the planning time.

6.3 Integration of planning and reasoning

Naturally, since the knowledge discovery task, the available algorithms and also the generated workflows are modeled using the KD ontology, the next step in the investigation is exploring the possibilities of tighter integration of planning and ontological reasoning to avoid transforming the complete knowledge discovery task description from the OWL formalism of KD ontology into PDDL. A planner directly working with the KD ontology would facilitate exploiting the expressivity of the ontology representation formalism and imposing further constraints on the required workflows.

To address these issues we developed an algorithm called PelletPlanner, which utilizes ontological reasoning capabilities implemented in the Pellet reasoner [90] during planning time. PelletPlanner has the same basic structure as the PDDLPlanner described in the previous section consisting of the enforced hill climbing algorithm and the relaxed GRAPHPLAN. However PelletPlanner directly accepts task descriptions defined in OWL using the KD ontology.

The main modification of the planner is that it obtains neighboring states during enforced hill-climbing by matching preconditions of available algorithms with currently satisfied conditions. Each matching is conducted in the planning time via posing an appropriate SPARQL-DL query to the KD ontology. The KD ontology is classified by the reasoner upon the initialization of the reasoner, therefore no time-consuming inferences are performed during the actual planning time.

6.4 Empirical evaluation

As explained in the previous sections, our system solves a task not addressed by existing algorithms. Empirical tests should thus primarily serve as a proof of concept, showing that the approach scales, with acceptable computational demands, to reasonably large real-life problem instances. We have conducted workflow construction experiments in two domains: genomics and product engineering. The workflows pertaining to both of the use cases are required to merge data with non-trivial relational structure, including ontology background knowledge. Again, this setting precludes the application of previous workflow construction systems, limiting the scope for comparative evaluation. However, we do run comparative experiments to evaluate the effects of employing either of the two earlier described planning strategies.

Also, to trace the dependence of runtime on the size of the KD ontology and the number of available algorithms annotated using the ontology, we perform experiments with two versions of the KD ontology and with growing set of algorithms for the first version. The second version of the KD ontology is a strict extension of our original KD ontology with added classes required for annotating algorithms from the Orange [27] system.

6.4.1 Use Cases

Genomics

In analyzing gene expression data we are dealing with the following sources of information: gene expression microarray data sets, Gene Ontology (GO) [23] and gene annotations. Annotations of genes using GO terms can be extracted from a public database.

Task The task was to apply relational machine learning algorithms to produce a set of descriptive rules for groups of genes differentially expressed in specific conditions, more specifically for the acute lymphoblastic leukemia and acute myeloid leukemia. The data sources available were a gene expression microarray data set, GO and gene annotations from the Entrez database[3]. The operators are algorithms for preparing inputs for the relational data mining (RDM) described in [101] and components of the framework for RDM with taxonomic background knowledge described in [104].

Engineering

Product engineering deals with very specific knowledge types such as CAD, documentation, ERP/database, etc. The SEVENPRO project addressed the problem of the effective reuse of heterogeneous knowledge and past designs by providing a unified view of the available knowledge through commonly agreed ontologies. Engineering designs capturing implicit expert knowledge have relational nature, specifying various numbers of primitive

[3]Maintained by US National Center for Biotechnology Information, ftp://ftp.ncbi.nlm.nih.gov/gene/.

Task	Ontology	No. of algorithms	PDDLPlanner Prep.	PDDLPlanner Plan	PelletPlanner Prep.	PelletPlanner Plan
GEN	KD-RDM	18	27	0.641	13	0.766
GEN	KD-RDM	43	75	0.828	38	2.703
GEN	KD-RDM	60	480	0.906	133	3.891
GEN	KD-Orange	43	402	0.984	205	3.688
ENG	KD-RDM	18	28	0.407	15	0.782
ENG	KD-RDM	43	68	0.906	33	2.438
ENG	KD-RDM	60	387	0.922	134	3.250
ENG	KD-Orange	43	349	0.625	218	3.062

Table 6.1: Planner performance results, with respect to the domain and ontology used and the number of algorithms available. The time for preprocessing (Prep.) and planning (Plan) is shown in seconds.

objects and relations between them. In the SEVENPRO environment data are encoded in a subset of the RDFS formalism.

Task One of the tasks solved within the SEVENPRO project was to generate descriptive and predictive rules from annotations of CAD drawings of different products. We were particularly interested in descriptive rules characterizing a particular class. The classification task was carried out as well in order to verify that we can distinguish between the product classes based on the provided information. The input data consisted of a list of identifiers of CAD drawings, CAD ontology and the annotations of individual CAD drawings.

6.4.2 Results

Experiments were carried out on a 1.8GHz Intel Centrino PC with 1GB memory. We used each planner for the two tasks described above and we used two versions of the KD ontology. The first version contains classes necessary for annotation of algorithms available in the RDM Manager tool (KD-RDM) [103], whereas the second version (KD-Orange) contains also classes necessary for annotations of algorithms available in the Orange data mining platform. KD-RDM contains 179 classes, 58 properties and 112 individuals. KD-Orange contains 266 classes, 59 properties and 198 individuals. The ontology KD-RDM was used to annotate 18 algorithms, which are part of the RDM Engine. The ontology KD-Orange was used to annotate also algorithms available in Orange [27], in total 43 algorithms. Other algorithm annotations for KD-RDM were created artificially.

For PDDLPlanner, the preprocessing stage includes conversion into PDDL. The results are summarized in Table 6.1. None of the Orange algorithms were employed in the produced workflows, they only served to make the search task harder.

The results primarily show that successful workflows (exemplified below) can be automatically achieved in small absolute run times. Further, we observe rather small sensitivity of the run times to the size of the KD ontology (more specifically, the number of algorithms it contains). This suggests that expected further extensions to this ontology will

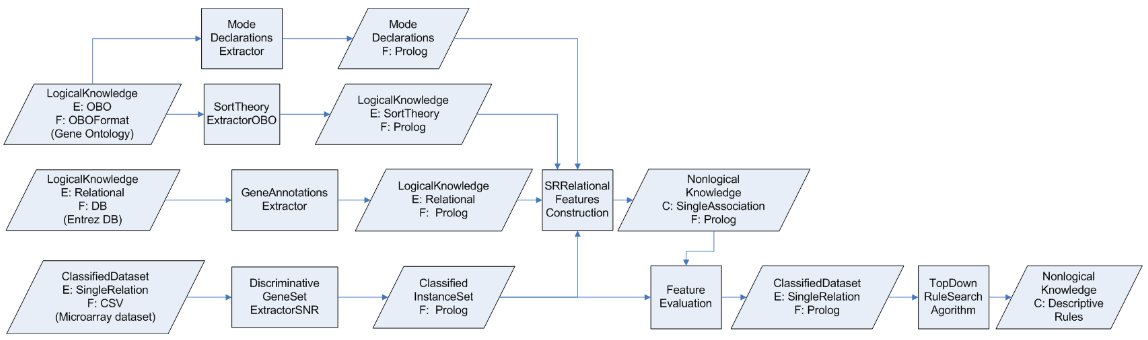

Mode Declarations Extractor
Mode Declarations F: Prolog
LogicalKnowledge E: OBO F: OBOFormat (Gene Ontology)
SortTheory ExtractorOBO
LogicalKnowledge E: SortTheory F: Prolog
LogicalKnowledge E: Relational F: DB (Entrez DB)
GeneAnnotations Extractor
LogicalKnowledge E: Relational F: Prolog
SRRelational Features Construction
Nonlogical Knowledge C: SingleAssociation F: Prolog
ClassifiedDataset E: SingleRelation F: CSV (Microarray dataset)
Discriminative GeneSet ExtractorSNR
Classified InstanceSet F: Prolog
Feature Evaluation
ClassifiedDataset E: SingleRelation F: Prolog
TopDown RuleSearch Agorithm
Nonlogical Knowledge C: Descriptive Rules

Figure 6.3: Abstract workflow generated for obtaining descriptive rules for groups of differentially expressed genes for AML vs. ALL. Rectangles represent algorithms and parallelograms represent data passed between them. Properties are abbreviated as follows: E: - hasExpressivity, C: - contains and F: hasFormat.

likely not deteriorate the system's performance significantly.

Interestingly, the results also show the superiority of the innovative PelletPlanner strategy of 'online querying for actions' over the baseline PDDLPlanner strategy in case of formulating new and diverse tasks, which is a typical scenario in both investigated domains. The single factor contributing to this superiority is the preprocessing time, smaller for PelletPlanner. This is mainly because ontology classification, the most time consuming operation within preprocessing, has to be performed twice by the reasoner when converting to PDDL. On the other hand, in preprocessing for PelletPlanner, this operation is performed only once. The described headstart of PelletPlanner is then reduced in the actual planning phase but still remains significant due to the relatively small proportion of planning time within the combined run time. In case of a set of planning tasks using the same domain description, the PDDLPlanner is however a better choice, since in this case the preprocessing phase can be run only once for the whole set of tasks.

An example of an abstract workflow generated for the genomics task described in 6.4.1 is shown in Fig. 6.3. The generated workflow utilizes algorithms developed by several different researchers and some of the tasks (e.g. discriminative gene set extraction) are independent of the rest. Using an automatically generated and semantically described workflow makes it far easier to conduct a series of experiments focusing influence of variations in one particular step of the process on the result of the whole data mining process without having to understand some other steps.

An example of an abstract workflow generated for the engineering task described in 6.4.1 is shown in Fig. 6.4. The same workflow had been produced manually within the SEVEPRO project and it was successfully rediscovered by the planner and executed using the RDM Manager tool developed within the SEVENPRO project.

6.5 Enhancing workflow construction

To evaluate the generality of the methodology presented in the previous sections and to identify additional challenges, we investigated automatic generation of workflows

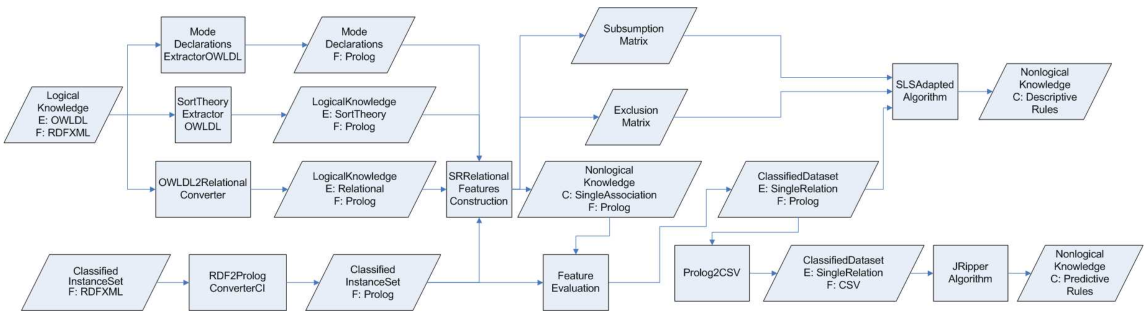

Logical Knowledge E: OWLDL F: RDFXML
Mode Declarations ExtractorOWLDL
SortTheory Extractor OWLDL
OWLDL2Relational Converter
Mode Declarations F: Prolog
LogicalKnowledge E: SortTheory F: Prolog
LogicalKnowledge E: Relational F: Prolog
SRRelational Features Construction
Subsumption Matrix
Exclusion Matrix
Nonlogical Knowledge C: SingleAssociation F: Prolog
ClassifiedDataset E: SingleRelation F: Prolog
SLSAdapted Algorithm
Nonlogical Knowledge C: Descriptive Rules
Classified InstanceSet F: RDFXML
RDF2Prolog ConverterCI
Classified InstanceSet F: Prolog
Feature Evaluation
Prolog2CSV
ClassifiedDataset E: SingleRelation F: CSV
JRipper Algorithm
Nonlogical Knowledge C: Predictive Rules

Figure 6.4: Abstract workflow generated for obtaining predictive and descriptive rules from annotations of CAD design drawings.

of data processing and data mining algorithms available within a single selected toolkit. Algorithms available in the Orange [27] data mining toolkit were annotated using the KD ontology. Two challenges were identified. Firstly, there are collections of algorithms, which are equivalent from the point of view of input/output description (e.g. set of ranking algorithms resulting in redundant search and a large number of generated workflows). Secondly, indefinitely long graphs can be constructed from the algorithms (e.g. using a sequence of preprocessing steps). We propose to address these two problems by imposing additional constraints during the planning phase.

6.5.1 Annotating Orange Algorithms

The KD ontology was used to annotate most of the algorithms available in the Orange toolkit. More than 60 algorithms have been annotated so far. As an example we present a definition of the Apriori algorithm in the description logic notation using the extended ABox syntax [5]:

$$\begin{aligned}
\{\texttt{Apriori}\} \sqsubseteq\ & \texttt{NamedAlgorithm} \\
\sqcap\ & \exists\,\texttt{output} \cdot \{\texttt{Apriori-O-Rules}\} \\
\sqcap\ & \exists\,\texttt{input} \cdot \{\texttt{Apriori-I-Dataset}\} \\
\sqcap\ & \exists\,\texttt{input} \cdot \{\texttt{Apriori-I-MinSupport}\} \\
\sqcap\ & \exists\,\texttt{input} \cdot \{\texttt{Apriori-I-MinConfidence}\}
\end{aligned}$$

$$\begin{aligned}
\{\texttt{Apriori-I-Dataset-Range}\} \equiv\ & \texttt{isRangeOf} \cdot \{\texttt{Apriori-I-Dataset}\} \\
\equiv\ & \texttt{Dataset} \sqcap \forall\,\texttt{hasFormat} \cdot \{\texttt{TAB}\} \\
& \sqcap \forall\,\texttt{hasExpressivity} \cdot \texttt{SingleRelationStructure} \\
& \sqcap \forall\,\texttt{hasAttributesType} \cdot \{\texttt{dDiscrete}\}
\end{aligned}$$

$$\begin{aligned}
\{\texttt{Apriori-O-Rules-Range}\} \equiv\ & \texttt{isRangeOf} \cdot \{\texttt{Apriori-O-Rules}\} \\
\equiv\ & \texttt{Patternset} \sqcap \forall\,\texttt{contains} \cdot \texttt{AssociationRule}
\end{aligned}$$

```
$task$ - instance of KnowledgeDiscoveryTask, $maxSteps$ - max length of the workflow, $constr$ -
additional constraints on the workflows
generateWorkflows($task$, $maxSteps$, $constr$):
  classify KD ontology;
  $algs$ := {instances of NamedAlgorithm};
  $algforest$ := inferAlgorithmHierarchy($algs$);
  $workflows$ := runPlanner($task$, $algforest$, $maxSteps$);
  $atomicW$ := expandWorkflows($workflows$, $algforest$);
  $filteredW$ := filterWorkflows($atomicW$, $constr$);
```

Figure 6.5: A skeleton of the procedure for workflow composition using the KD ontology.

The Apriori algorithm is defined as an algorithm that can be applied to a single relation dataset in the TAB format containing only discrete attributes and produces a result in the form of a set of association rules. It has two parameters: minimal support and minimal confidence of the rule. All three parameters are specified by integer values. The other input parameters were omitted from this example.

The algorithms were annotated manually, since no systematic description of these algorithms e.g. in PMML[4] or WSDL[5] was available. The annotated algorithms also served as case studies to validate and extend the KD ontology, therefore developing a procedure for semi-automatic annotation is a subject for future work.

6.5.2 Exploiting algorithm hierarchy

In this section we present an enhanced version of the algorithm described in 6.3. In the original version of the PelletPlanner , there are no mechanisms for exploiting the algorithms hierarchy. We have enhanced the algorithm in two ways: a hierarchy of algorithms based on defined classes and input/output specifications computed and in searching for neighboring states the planner exploits the algorithm hierarchy.

A hierarchy of algorithms is inferred before the actual planning. It needs to be recomputed only when a new algorithm is added to the ontology. The hierarchy of algorithms is based on the inputs and outputs of the algorithms and on the defined algorithm classes such as `PreprocessingAlgorithm`. An algorithm $A_j \sqsubseteq A_i$, if for every input of I_{ik} A_i there is an input I_{jl} of algorithm A_j such that range of $I_{ik} \sqsubseteq I_{jl}$. An algorithm $A_i \equiv A_j$, if $A_j \sqsubseteq A_i$ and $A_i \sqsubseteq A_j$. The subsumption relation on algorithms is used to construct a forest of algorithms with roots given by the explicitly defined top-level algorithm classes e.g. `DataPreprocessingAlgorithm`.

The planning algorithm was adapted so that in the search for the next possible algorithm it traverses the forest structure instead of only a list of algorithms and considers a set of equivalent algorithms as a single algorithm. Currently, only constraints on

[4]http://www.dmg.org/pmml-v4-0.html

[5]www.w3.org/TR/wsdl

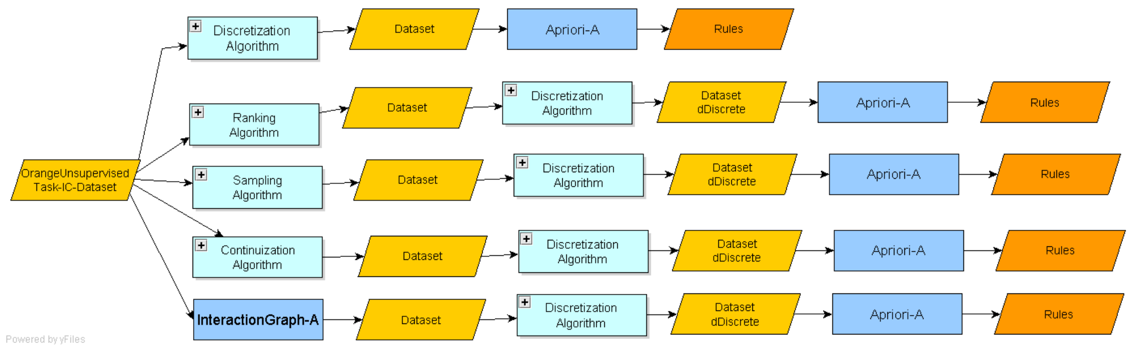


Figure 6.6: An example of workflows for discovering association rules in Orange.

Task	No. of	Planner		HierarchyPlanner	
	algorithms	Prep.	Plan	Prep.	Plan
GEN	71	72	0.854	115	0.560
GEN	99	104	1.123	155	0.568
ASSOC	71	94	27.291	125	25.154
ASSOC	99	98	107.549	153	24.354

Table 6.2: Planner performance results, with respect to the task, the number of algorithms available. The time for preprocessing (Prep.) and planning (Plan) is shown in seconds.

repetition of some kind of algorithms in a linear part of the workflow are built into the planner. The additional constraints on workflows are used only in filtering of workflows during post-processing (procedure **filterWorkflows**). Workflows for all the members of an equivalence set are generated using the procedure **expandWorfklows**. The information about algorithms subsumption is also used in workflow presentation. An overview of the whole procedure for workflow generation is shown in Figure 6.5.

The generated workflows are presented to the user using an interactive visualization, which enables the user to browse the workflows from the most abstract level to specific combination of algorithm instances. The workflows with the smallest number of steps are presented first. An example of a set of workflows generated for discovering association rules in Orange is in Figure 6.6.

6.5.3 Enhanced Planner Evaluation

We carried out experiments comparing the enhanced planner exploiting the algorithm hierarchy with the original classical planner. We used each planner for two tasks. The first task was the complex and specialized task of discovering descriptive rules in the genomics domain. The second task was a simple task of discovering association rules. The KD ontology including the subontologies for annotation of the individual algorithms contains about 500 classes and 500 individuals.

The results summarized in Table 6.2 indicate that the HierarchyPlanner exploiting the algorithm hierarchy needs shorter time for planning for all the tested settings. With increasing number of equivalent algorithms the time taken for planning rises less rapidly for the HierarchyPlanner. The preprocessing stage lasts longer for the HierarchyPlanner due to the construction of algorithms hierarchy, however this task can be performed offline and repeated only when the ontology changes.

The example of a set of generated workflows shown in Figure 6.6 illustrates the use of algorithm hierarchy for workflow presentation. Since there are 4 discretization, 4 sampling, 5 ranking and 6 continuization algorithms, it would be infeasible to present all the generated workflows without using the algorithm hierarchy. The automatic selection of some relevant subset of workflows is non-trivial and will be a subject of future work.

The primary objective of this study was to investigate challenges of data mining workflow construction resulting mainly from sets of similar or equivalent algorithms, which are typically available in a data mining toolkit and to develop a methodology for integrating our approach to automatic workflow composition to a data mining toolkit, which contains means for manual workflow creation and execution.

We have developed a planner, which exploits the hierarchy of algorithms annotated using the KD ontology, and shown that during the planning stage, this planner is faster and more scalable than the classical planner. The construction of algorithm hierarchy is time consuming, however it needs to be recomputed only when a new algorithm is added to the ontology. Moreover the hierarchy can also be exploited in the presentation of the workflows to the user. In future work we plan to include additional constraints and user preferences into the planner.

Part III

Application

Chapter 7

Application of the developed methods

This chapter presents more details on applications of methods developed in Part I and Part II of the thesis. The complete application showcase of automatic construction of workflows for complex relational data mining task in the domain of product engineering is described and a more extensive evaluation of sorted refinement proposed in Chapter 3 on tasks from this domain is provided. In the second part of this chapter a prototype of integration of the workflow construction methodology into the Orange4WS knowledge discovery platform is discussed.

7.1 Relational data mining for Product Engineering

7.1.1 Description of the application area

Engineering is one of the most knowledge-intensive activities that exist. More specifically, product engineering has been a key to the development of a strong and specialized manufacturing industry in Europe, organized in RTD departments and technical offices. Product engineering involves dealing with very specific knowledge types, like product structures, CAD designs, technical specifications, standards, and homologations. Moreover, specific electrical, mechanical, thermodynamic and chemical knowledge may include empirical data, simulation models and Computer-aided engineering analysis(CAE) tools that serve to optimize relevant features of the design. The result is rich and complex knowledge stored in many heterogeneous formats, of which probably CAD, documentation and ERP/database are the most frequently found, and which constitute the focus of the SEVENPRO project. The project addresses the most important problem encountered by engineering teams: the effective reuse of knowledge and past designs.

Most engineering teams currently have to access heterogeneous information sources from different tools which, in most cases, do not interoperate. The development of a new product, or a product order with high level of customization, requires a new engineering approach. During the development process, engineering staff works out new product item designs by means of CAD tools. CAD designs contain vast amounts of implicit knowledge

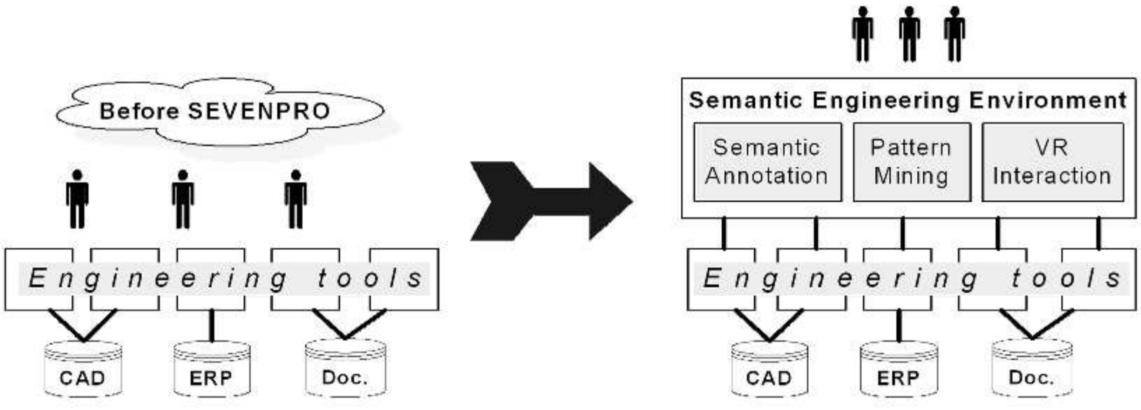


Figure 7.1: Engineering heterogeneous information sources before and after SEVENPRO.

about the way experienced engineers design very specialized parts. Efficient reuse of knowledge depends on appropriate organization of information and the capability of retrieving it for later use. Engineering teams still have to spend lots of time trying to find existing designs from a vast CAD repository; in many occasions, they design again and again items very similar to others already existing. Moreover, the different types of knowledge described are supported by different systems, which are used separately and have no communication with each other, like CAD tools, ERP systems, documents and spreadsheets, etc. This situation is illustrated in Fig. 7.1(left).

To efficiently retrieve information, it is necessary to be able to carry out complex searches, combining geometrical, technical and managerial aspects. This would allow the engineer, for example, to query about "parts of type clamp [itemFamily], with more than 6 holes [Geometry], set in order later than November/2004 [Management], compliant with ISO-23013 [Documentation]". This is not possible with current information systems, unless an expensive and complex Product Lifecycle Management (PLM) system is set up, whose maintenance in terms of information updates is burdensome for every company and simply unaffordable in terms of cost for SMEs. The only feasible approach to this is by using semantic-knowledge technologies and a well automated semantic-annotation system from the different information sources, able to extract from them all the knowledge that is useful for the engineering activity. In order to achieve this, an integrated architecture is required, able to extract and maintain a layer of semantic annotations from the different information sources targeted, namely ERP, CAD and Document repositories. As shown in Fig. 7.1(right), a novel semantic virtual engineering product design scenario, aims at a better integration and reuse of design knowledge.

7.1.2 Relational data mining of CAD data

The SEVENPRO ontologies and the corresponding annotations cover a large spectrum of engineering concepts (items, orders, norms, problems, versioning, among many

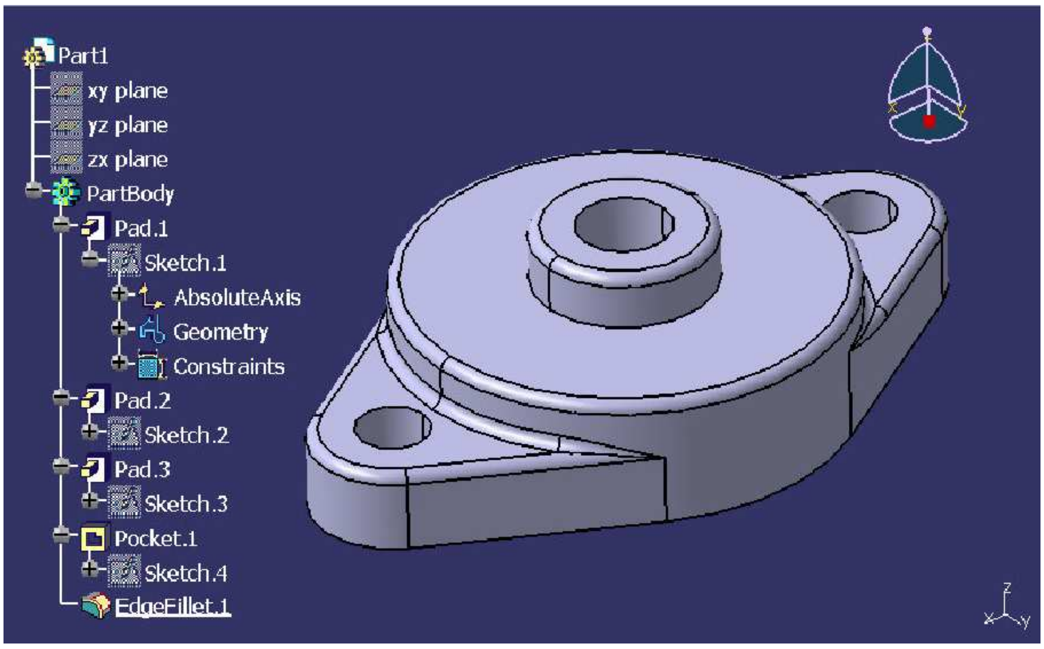


Figure 7.2: Example of a CAD design including commands history.

others). As mentioned, this allows for complex queries across the available knowledge. An important facet of this knowledge is the CAD design information. Engineering departments generate a large amount of CAD files. Such files can be 3D part-definition files, 3D assembly definition files or 2D drafting files. In addition, relevant information ranges from textual data (like block, operation or part names) and generic document structure (like assembly structure), to detailed design information in the case of 3D parts. In the later case, the shape of a 3D part is the result of sequence of operations specified by the designer. This sequence of design operations (design features) is where most of the designer's knowledge resides, as it is a reflection of the designer's experience.

Fig. 7.2 represents a simple mechanical part, a two bolt flange. Notice the command history (at the left-hand side of the figure) leading to the particular virtually designed object. In command histories the basic operations are "creating" matter (e.g., a pad, a stiffener) and "removing" matter (e.g., a chamfer, an edgeFillet).

This design history conveys the higher level information on how the object was designed as well as high level dimensional information, as the commands have parameters associated to them (like the height of an extrusion or the radius of a fillet). This information would be more difficult to determine using only the final shape of the part, however, having it associated to the operation not only makes it easily accessible but also keeps its real meaning. The design history, presented at the left-hand side of Fig. 7.2, is depicted in the annotation layer as a design sequence in terms of ontology classes and instances, as shown in Fig. 7.3.

This kind of highly relational data exists for all the annotated files, and is the input to a RDM algorithm. The generated instance schema is simplified with respect to the

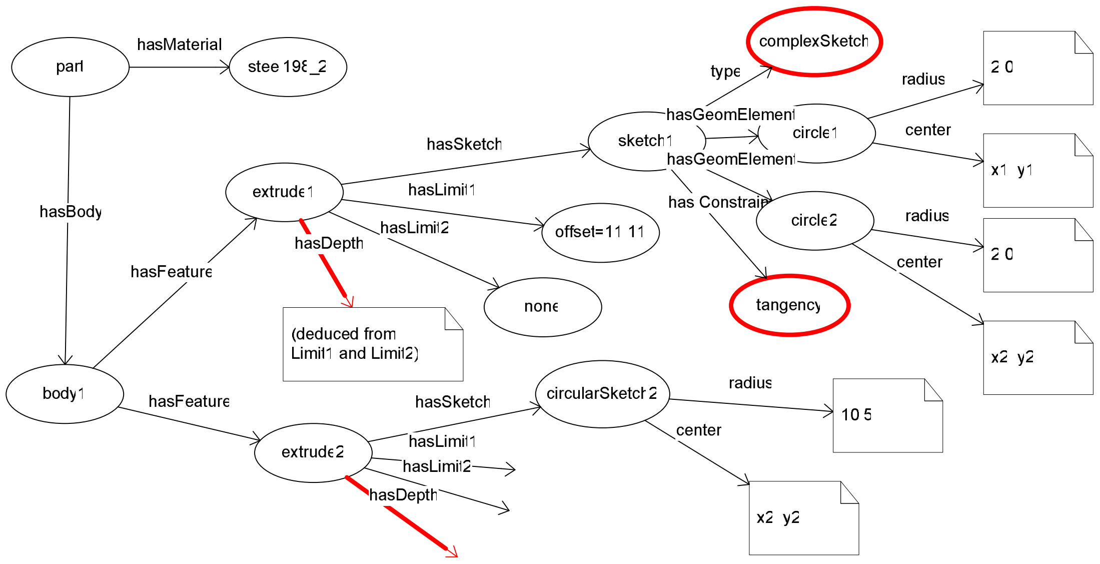


Figure 7.3: Part of a semantic annotation of the design shown in fig. 7.2

internal CAD representation. For example, if a sketch does not belong to any predefined category, it is identified as a complexSketch and it is not further elaborated. The schema also contains some properties derived from other properties, e.g. property hasDepth of extrude is derived from the two limits. In SEVENPRO, this representation has been converted into Prolog facts, more suitable as input for the RDM algorithms. An example of Prolog facts describing part of CAD design is in Table 7.1.2.

Experimental setting

Experiments were performed on a dataset containing 160 examples of CAD design drawings provided by a metal casting company that participates in the project: Fundiciones del Estanda. Two main types of experiments were run:

- searching for relational patterns present in all examples of a given class, to compare efficiency of the sorted refinement enriched RDM to a baseline ILP system
- classification based on constructing propositional features to evaluate predictive accuracy of the propositionalisation approach to classification.

Comparison of sorted refinement with Aleph

We conducted experiments to compare the efficiency of RDM including sorted refinement (SR) on one hand and a standard ILP system on the other hand. The baseline ILP system chosen for comparison was Aleph. The specific goal of the experiment was to determine the volumes of search space traversed by the respective systems in order to find patterns covering all of the provided positive examples.

Table 7.1: Prolog facts describing a part of CAD design.

```
hasCADEntity('eItemT_BA1341',part_183260395_10554).
typeOf('eItemT_BA1341', eItemT).
typeOf(part_183260395_10554, cADPart).
hasBody(part_183260395_10554,body_183260395_10555).
typeOf(body_183260395_10555, body).
hasFeature(body_183260395_10555,extrude_183260395_10556).
typeOf(extrude_183260395_10556, extrude).
hasSketch(extrude_183260395_10556,complexSketch_183260395_10557).
typeOf(complexSketch_183260395_10557, complexSketch).
hasGeomElem(complexSketch_183260395_10557,circle_183260395_10558).
typeOf(circle_183260395_10558, circle).
hasDepth(extrude_183260395_10556,0).
hasFeature(body_183260395_10555,pocket_183260395_10580).
typeOf(pocket_183260395_10580, pocket).
hasSketch(pocket_183260395_10580,complexSketch_183260395_10581).
typeOf(complexSketch_183260395_10581, complexSketch).
```

The majority class of examples is considered as positive. For the sake of this experiment, no negative examples are needed. There were 57 examples, where each examples contained a description of one CAD design drawing. Around 100 predicates were used to describe each example.

The tests were performed for pattern length from 1 to 8. For pattern length greater than 5, pattern generation was no longer tractable for Aleph. In the first set of experiments only term subsumption was used in our system. It can be seen that the number of expanded nodes is decreased very significantly. In the second set of experiments, predicate subsumption was used in our system as well. Results of these experiments can be seen in Figures 7.4 and 7.5. Fig. 7.4 shows results of using sorted refinement with and without predicate subsumption. Fig. 7.5 shows results of both our approaches compared to Aleph. The time taken for evaluation roughly doubles w.r.t. experiments using term subsumption only. The number of explored nodes decreases, however the decrease is not very significant. This is due to the fact that the subproperty relation hierarchy that was used has only two levels and includes around 10 predicates. Our system can be used for pattern sizes, which are intractable in Aleph. This is important, because it has been discovered that patterns with length less than 7 do not provide information sufficient for classification.

Classification based on propositional features

For the data set containing 160 design drawings their classification was provided. Examples were classified into 4 proper classes describing families of designs and 57 examples that did not belong to any of the 4 classes were classified as 'other'. By consultation with the users it was found out that the first feature used is important and also relative order of the features is important. Therefore properties describing the order of CAD features were added to background knowledge and to annotations e.g. `next(+cADFeature,-cADFeature)`,

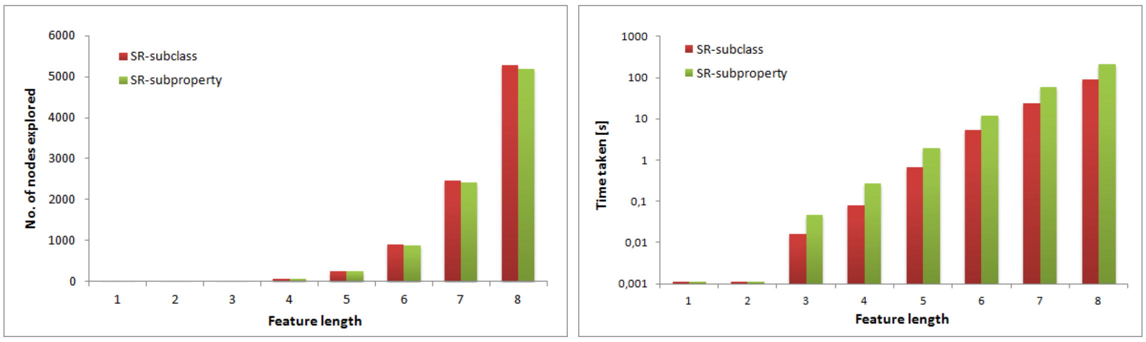


Figure 7.4: Comparison of sorted refinement with and without using taxonomy on predicates. Left: Number of nodes explored Right: Time taken.

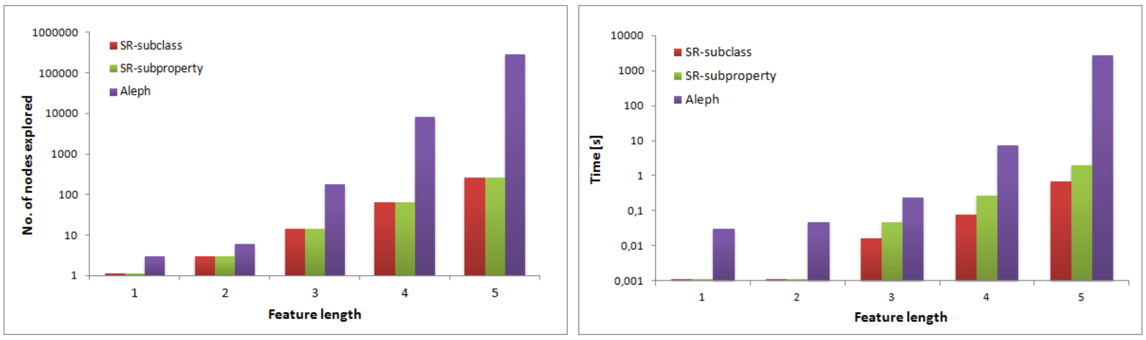


Figure 7.5: Comparison of sorted refinement and Aleph. Left: Nodes explored Right: Time taken.

`sequenceStart` and `firstFeature(+body,-cADFeature)`. The following relations were also added to the background knowledge:
`subpropertyOf(firstFeature,hasFeature)`,
`subpropertyOf(hasFeature,sequenceStart)`. Special treatment of relations that are subproperties of `next` and `sequenceStart`. Subproperties of `sequenceStart` can occur only once in a pattern and for subproperties of `next` order on the level of arguments is not checked.

Our system was used to generate a set of features of length 7. The generated features set was pruned by excluding features covering all examples. Also in case a feature covered the same examples as some of its children, the feature was excluded. Propositional algorithm J48 implemented in WEKA [108] was then used for classification using generated features as attributes. For testing 10 fold cross validation was used. Results of the classification are summarized in Table 7.1.2.

The prevailing error type indicated in the confusion matrix in Table 7.1.2 is that some items of class itemFamilyStdPlate were incorrectly classified as itemFamilySlottedPlate. These two classes are both subclasses of class itemFamilyPlate and they are more similar to each other than any other pair of classes. More detailed information or longer features would be necessary to distinguish between these two classes more accurately. Other

Table 7.2: Results of classification using the J48 algorithm.

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
itemFamilyTT	0.826	0.036	0.792	0.826	0.809	0.9
itemFamilyLiner	0.895	0.068	0.879	0.895	0.887	0.927
itemFamilyStdPlate	0.5	0.02	0.571	0.5	0.533	0.834
itemFamilySlottedPlate	0.8	0.02	0.727	0.8	0.762	0.883
other	0.855	0.071	0.883	0.855	0.869	0.897

Table 7.3: Confusion matrix of classification using J48 algorithm.

a	b	c	d	e	⟵ classified as
53	2	5	1	1	a = other
2	19	2	0	0	b = itemFamilyTT
2	3	51	1	0	c = itemFamilyLine
2	0	0	4	2	d = itemFamilyStdPlate
1	0	0	1	8	e = itemFamilySlottedPlate

errors were mostly confusions between one of the proper classes and class 'other'.

Discussion of experiments

In this section we have described semantic virtual engineering for product design in engineering environments, which integrates information from heterogeneous sources by means of a semantic layer, and identified the role of relational data mining in this application. As a case study, semantic annotation and RDM on CAD designs was chosen, since CAD designs are challenging from the ILP point of view due to the various length and structure of the description of each example combined with taxonomical background knowledge. We have proposed a baseline approach for integrating taxonomical background knowledge into an ILP system by implementing sorted refinement operator and extending it to include taxonomies on predicates.

The efficiency of our approach was demonstrated by comparing it to the standard ILP system Aleph without any support for integration of hierarchical background knowledge. The results were strongly convincing in favor of the former. In terms of the volume of search spaced traversed to find a set of frequent patterns, the 'hierarchy-blind' search conducted by Aleph maintains a roughly exponential overhead with respect to the ontology-aware refinement, as the maximum pattern size is being increased. This has a strong consequence in this application domain: working in spaces of patterns of length greater than 7 literals becomes intractable for Aleph, while such and longer patterns are important for capturing common design sequences as exemplified earlier in the text.

Features generated by our system were also used for classification of CAD designs. Generally speaking, the accuracies obtained through cross-validation were surprisingly high, which can be ascribed both to the noise-free character of the data and to the sufficient expressivity of the features our system constructed. Analyzing the prevailing classification

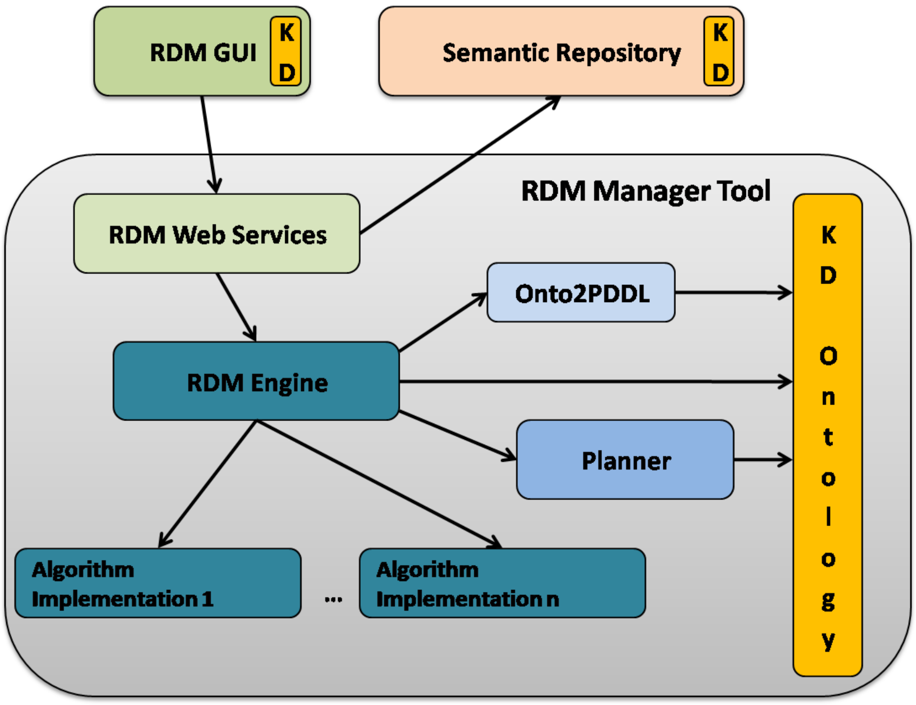


Figure 7.6: An overview of the RDM Manager architecture.

error type, it was discovered that the order of CAD design features was important for classification, and thus predicates and rules describing the order of predicates were established.

Conclusions

The conducted experiments demonstrated that the chosen approach to integration of hierarchies of concepts and predicates into a refinement operator leads to a significant increase of efficiency of relational rule learning and feature generation, enabling use of larger patterns. These results motivate further exploration of refinement operators for more expressive languages.

7.1.3 Workflow creation, execution and management

Relational knowledge discovery tasks described in the previous sections required a combination of transformation, inductive and visualization algorithms. These can be assembled manually by a computer scientist familiar with these types of algorithms and their incorporability into a workflow suitable for a particular knowledge discovery task. However it cannot be expected from the end users, in case of SEVENPRO project product engineers. Therefore in order to make relational knowledge discovery techniques accessible to these users, the workflows should be assembled automatically based on the input and output specifications of the particular knowledge discovery task and the results of the knowledge discovery task should be stored in a form that can be easily queried by the user.

Our automatic workflow construction methodology presented in Chapter 6 was implemented in the RDM Manager (shown in Fig. 7.6), which was developed as a part of the SEVENPRO software infrastructure [97]. The RDM Manager provides functionalities for workflow management and execution. The central component of the RDM Manager is the RDM Engine responsible for designing, storing, retrieving and executing workflows. For this sake, the RDM Engine has access to the KD ontology, and can launch the planner, the ontology based constructor of PDDL files (Onto2PDDL box in Fig. 7.6) as well as all the various algorithms appearing in workflows. The RDM Engine is equipped with a web service interface allowing a standardized access. A graphical user interface (RDM GUI) has been developed enabling an intuitive specification of the knowledge discovery task and passing on the specification to the RDM Engine web services. The Semantic Repository box also shown in Fig. 7.6 is a central storage point of the SEVENPRO software platform. The RDM Manager Tools stores all results of knowledge discovery processes, including the constructed workflows into the Semantic Repository for later retrieval by itself or by other software components of the SEVENPRO platform. Conversely, the Semantic Repository also holds all data onto which knowledge discovery workflows are applied.

The general workflow maintenance scenario is shown in Fig. 7.7. The user formulates a knowledge discovery task using the RDM GUI, which formalizes the task specification into a SPARQL query, passed to the RDM Engine. The RDM Engine queries the Semantic Repository for an existing workflow (possibly a single algorithm) solving the task. If such a workflow is found, it is presented to the user who can set or update its parameters. Otherwise, the RDM Engine calls the Planner. If the PDDLPlanner is used, the Onto2PDDL component is called first to produce the suitable PDDL file.

A plan generated by the Planner is a directed acyclic graph with nodes representing `Algorithm` instances, which do not contain any values of algorithm parameters specified by simple datatypes (e.g. `MinNo` - the minimal number examples covered by one rule). Therefore in the next stage it is necessary to convert the plan actions into a sequence of instances of `AlgorithmExecution`.

A SPARQL-DL query is used to search for the instances of `AlgorithmExecution` used by the actions in the plan. In the current version of our system the user has three options: to use default configurations for all the algorithms, to choose from among previously used configurations or to set all parameters manually.

When all the actions of the plan have been instantiated, they are combined into an abstract workflow represented by an instance of the `Workflow` class, which is stored in the Semantic Repository. Since the current version of the RDM Engine does not provide any means for parallelization, the actions of the workflow are converted to a sequence. The RDM Engine then generates a query for execution of each algorithm configuration in the sequence.

The data are then retrieved from the Semantic Repository using a SPARQL query. Then, for each algorithm in the sequence, the RDM Engine extracts the algorithm's class from its `AlgorithmExecution`. Knowing the class, it then launches the algorithm's wrapper passing the retrieved the data and parameters to it. When the algorithm terminates, the RDM Engine passes its results to the wrapper of the next algorithm in the sequence.

The algorithms available in the RDM Engine include specialized algorithms for

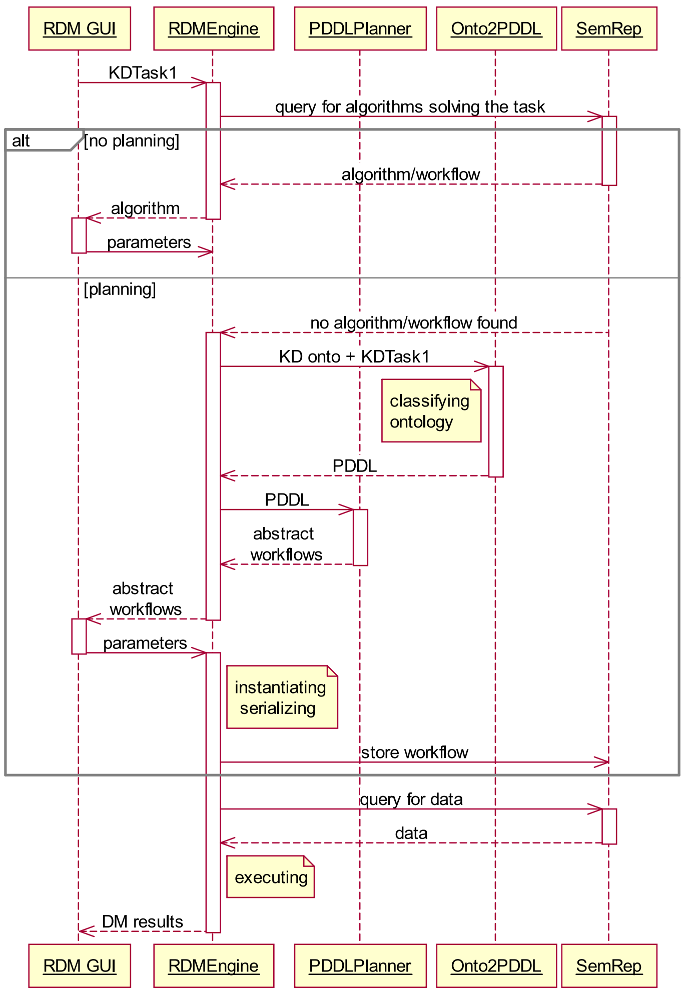


Figure 7.7: Sequence diagram showing a typical scenario for PDDLPlanner.

relational learning through propositionalization [59] and subsequent propositional search described in [104], a collection of algorithms from Weka data mining platform [108] including the JRip rule learner, the J48 decision tree induction algorithm and Apriori algorithm. In addition, algorithms for data preprocessing and format conversions are also available within the RDM Engine. New algorithms can be easily added to the RDM Engine by developing a wrapper for the particular algorithm and possibly also an API for accessing the results in case the particular result type is not yet included in the RDM Engine.

7.2 Orange4WS knowledge discovery platform

Fast-growing volumes of complex and geographically dispersed information and knowledge sources publicly available on the web present new opportunities and challenges for knowledge discovery systems. Principled fusion and mining of distributed, highly heterogeneous data and knowledge sources requires the interplay of diverse data processing and mining algorithms, resulting in elaborate data mining workflows. If such data mining workflows were built on top of a service-oriented architecture, the processing of workflow components (e.g. data mining algorithms) can be distributed between the users computer and remote computer systems. Therefore, as the use of data mining algorithms (implemented as services) is no longer limited to any particular data mining environment, platform or scenario, this can greatly expand the domains where data mining and knowledge discovery algorithms can be employed.

Orange4WS (Orange for Web Services) knowledge discovery platform , has been conceived as an extension of the existing data mining platform Orange [27]. In comparison with the current publicly available data mining platforms (best known examples being Weka [108], KNIME [10], RapidMiner [74] and Orange [27]), the Orange4WS platform provides the following new functionalities: (a) userfriendly composition of data mining workflows from local and distributed data processing/mining algorithms applied to a combination of local and distributed data/knowledge sources, (b) simplified creation of new web services from existing data processing/mining algorithms, (c) a knowledge discovery ontology of knowledge types, data mining algorithms and tasks and (d) automated construction of data mining workflows based on the specification of data mining tasks, using the data mining ontology through an algorithm that combines planning and ontological reasoning.

7.2.1 Text mining use case

This section presents a motivating use case for developing and using a service-oriented knowledge discovery platform, including a user-friendly workflow editor. The use case is built upon text mining web services, available from LATINO [1] text mining library, which provides a range of data mining and machine learning algorithms, with the emphasis on text mining, link analysis and data visualization.

The goal of this use case is to produce a compact and understandable graph of terms, which could potentially give insights into relations between biological, medical and

[1]http://sourceforge.net/projects/latino

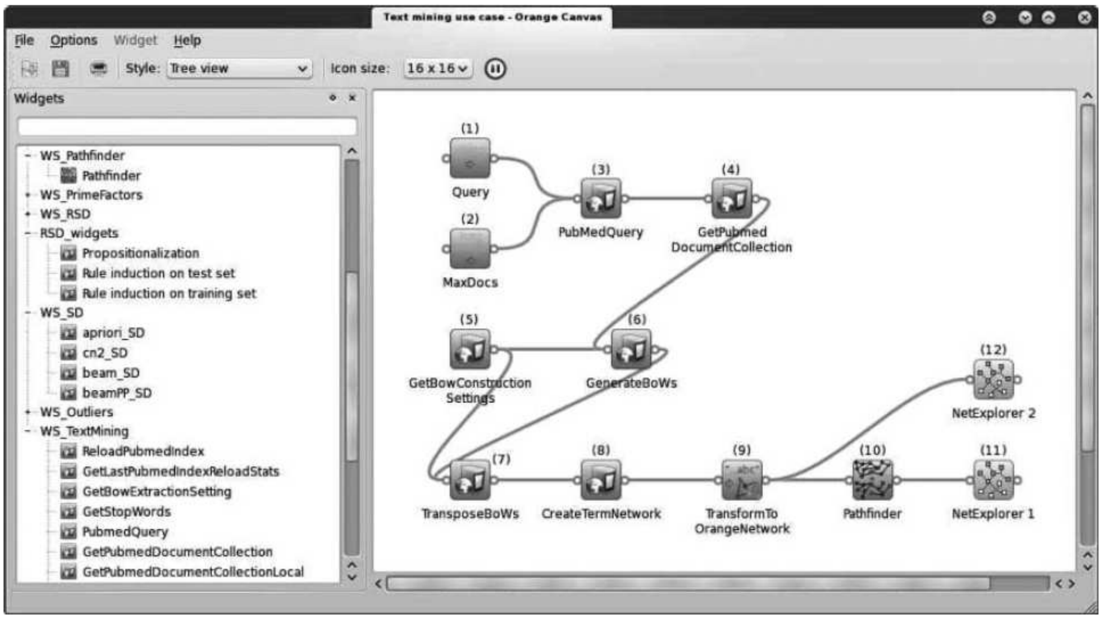


Figure 7.8: An Orange4WS workflow of text mining services in the Orange workflow execution environment. Components numbered 3, 4, 5, 6, 7, 8 and 10 are web services; components 1, 2 and 9 are Orange4WS supporting widgets; components 11 and 12 are instances of the native Orange graph visualizer.

chemical terms, relevant to the subject of a user-defined query.A manually constructed Orange4WSworkflowof processing components is shown in Fig. 7.8.

The use case demonstrates the need for a service oriented platform able to combine publicly available data repositories (PubMed) with third-party data analysis tools (LATINO), specialized algorithms (Pathfinder) and powerful local visualization components (Orange graph visualizer).

PubMed search web services is queried with a user-defined query string and a parameter defining the maximal number of documents returned (components 1, 2 and 3). It returns a collection of IDs of relevant documents. Then, the obtained IDs are used to collect titles, abstracts and keyword of these documents (component 4). Next, bag-of-words (BoW) sparse vectors are created from the collection of words (component 6). To simplify the setting of parameters for unexperienced users, there is a service providing a suitable set of default values that can be used as an input to the web service that constructs BoW vectors (component 5). BoW vectors are then transposed (component 7) and a network of words/terms is created (component 8) in the .net format of the well-known Pajek social network analysis tool [2]. The resulting graph of terms in the .net format is then transformed into Oranges native data structure for representing graphs (component 9), and simplified using a sparse variant of the Pathfinder algorithm that is implemented as a web service

[2]User manual of the Pajek software tool for the analysis and visualization of large social networks is available at http://pajek.imfm.si/doku.php.

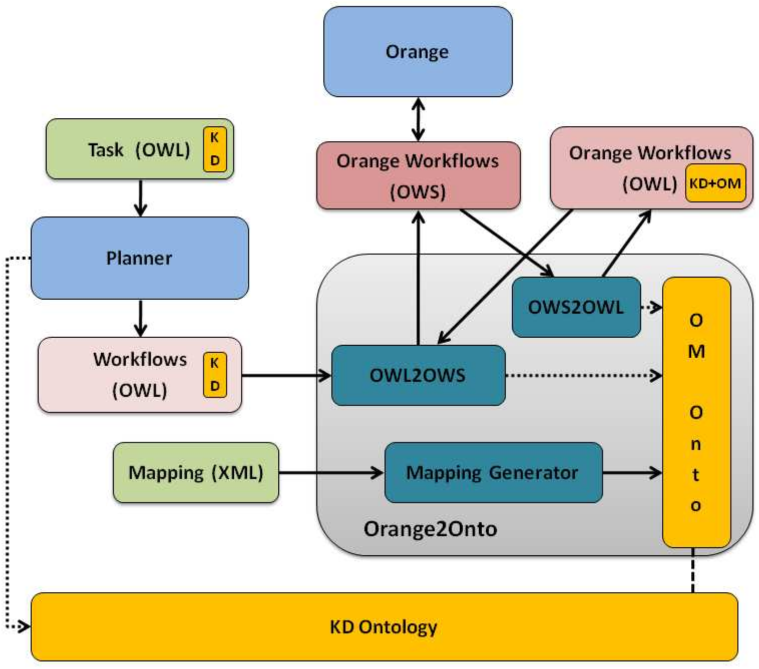


Figure 7.9: An overview of the framework for integration annotations and planning into Orange.

(component 10). Finally, the original and pruned graph are visualized using the Oranges native Network explorer (components 11 and 12).

This Orange4WS workflow, implementing a complex text mining scenario, was designed and constructed manually in the Oranges user-friendly workflow editor. In Section 7.2.2, we will demonstrate how this workflow can be constructed automatically using our workflow planner and KD ontology.

7.2.2 Integrating annotations and planning into Orange4WS

We have developed a framework for integrating our methodology into the Orange4WS data mining platform, so that workflows, which were constructed manually using the Orange4WS GUI, can be automatically annotated using the KD ontology. The annotated workflows can then be used for querying and reasoning. All the information required for the Orange4WS representation is preserved in the annotation; therefore Orange4WS workflows can be recreated from the annotations and executed again in the Orange4WS toolkit. On the other hand, workflows generated by the planner using KD annotations of Orange4WS algorithms can be converted to the Orange4WS representation and executed in Orange4WS. An overview of the framework is shown in Fig. 7.9. The module Orange2Onto, which acts as an interface between Orange4WS and ontology representation does not work directly with internal representation of Orange4WS, but it works with the OWS format used in the standard Orange distribution to store workflows in XML format.

In order to capture formally the mapping between the internal Orange4WS representation and the representation of algorithms using the KD ontology, the Orange-Map (OM) ontology was developed defining templates for mapping of algorithms, data and parameters. The template for a parameter represented using a set of radio buttons in Orange is shown below:

$$\begin{aligned}\texttt{OrangeRadioParamMapping} &\sqsubseteq \exists\texttt{parameter}\cdot\{\texttt{kd:AlgorithmParameter}\}\\ &\sqcap \exists\texttt{radioValue}\cdot\texttt{OrangeRadioValue}\\ &\sqcap \exists\texttt{orangeAlias}\cdot\texttt{string}\end{aligned}$$

$$\begin{aligned}\texttt{OrangeRadioValue} &\sqsubseteq \exists\texttt{paramURI}\cdot\{\texttt{anyURI}\}\\ &\sqcap \exists\texttt{rbNumber}\cdot\texttt{int}\end{aligned}$$

Currently in the Orange format for storing the workflows, a parameter value is represented only by the number of the selected radio button. It is specified in the mapping using the property `rbNumber`. In the KD ontology it is mapped into an instance with URI specified using `paramURI`.

The OM ontology is then used for converting the automatically generated workflows into the Orange4WS representation. In order to facilitate the creation of the mapping for new algorithms, the mapping can be specified using an XML file. The corresponding instances in the ontology are then generated automatically.

Annotation of a new algorithm available in Orange4WS thus requires the following steps:

1. create instances of `AlgorithmParameter` for all inputs and outputs
2. create an instance of `NamedAlgorithm`
3. for each instance of `AlgorithmParameter` create a class defining its range (if not yet defined, add the necessary subclasses of `Knowledge` - this should be required only when a new type of algorithm is added)
4. create an XML file defining a mapping between the algorithm representation in Orange and in the KD ontology
5. run a script for generating a mapping using the OM ontology

Annotations of Orange4WS workflows containing algorithms not annotated using the KD ontology can also be created automatically. The missing information about input/output types of the algorithms is then either deduced from the links with annotated algorithms or considered to be some `Knowledge` expressed as string. The annotations of such workflows can therefore be used for some querying and repeating of experiments, however the generated annotation of the unknown algorithm is not suitable for planning.

The procedures for converting Orange4WS representation to OWL and vice versa are implemented in Python using JPype[3] to call the Jena[4] ontology API implemented in Java.

[3]http://jpype.sourceforge.net/

[4]http://jena.sourceforge.net/

Task	No. of	Planner		HierarchyPlanner	
	algs	Prep.	Plan	Prep.	Plan
LATINO	85	144	0.38	225	0.08
LATINO	113	211	0.55	363	0.08
ASSOC	71	64	47.3	116	44.8
ASSOC	99	183	84.4	261	46.1

Table 7.4: Planner performance results, with respect to the task, the number of algorithms available. The time for preprocessing (Prep.) and planning (Plan) is shown in seconds.

Text mining use case annotation and workflow construction

The KD ontology was used to annotate the components of LATINO using the mapping described in guidelines for mapping Java classes to OWL classes [51]. The annotation was done manually and was based on LATINO WDSL specification. As an example we present a definition of the `GenerateBows` algorithm. `GenerateBows` is defined as an algorithm that can be applied to a collection of documents and produces a bag of words representation of these documents. The settings are quite complex, therefore in the current version they are provided as a single input object. The annotation serves rather as a proof of concept for including algorithms available as web services. Developing a branch of the KD ontology describing the algorithms and types of data used for text mining is out of scope of this work. The definition of the `GenerateBows` algorithm in the description logic notation using the extended ABox syntax [5] is shown below:

$$\begin{aligned}
\{\texttt{GenerateBows}\} \sqsubseteq\ & \texttt{NamedAlgorithm} \\
\sqcap\ & \exists\,\texttt{output}\cdot\{\texttt{GenerateBows-O-Bows}\} \\
\sqcap\ & \exists\,\texttt{input}\cdot\{\texttt{GenerateBows-I-Docs}\} \\
\sqcap\ & \exists\,\texttt{input}\cdot\{\texttt{GenerateBows-I-Settings}\}
\end{aligned}$$

$$\begin{aligned}
\{\texttt{GenerateBows-I-Docs-Range}\} \equiv\ & \texttt{isRangeOf}\cdot \\
& \{\texttt{GenerateBows-I-Docs}\} \\
\equiv\ & \texttt{DocumentCollection}
\end{aligned}$$

$$\begin{aligned}
\{\texttt{GenerateBows-O-Bows-Range}\} \equiv\ & \texttt{isRangeOf}\cdot \\
& \{\texttt{GenerateBows-O-Bows}\} \\
\equiv\ & \texttt{BowSpace}
\end{aligned}$$

The text mining use case described in Section 7.2.1 and our planner algorithms were used to automatically generate suitable workflow, which was then loaded into the Orange4WS platform. A schema of the generated workflow and its executable instantiation in the Orange4WS environment is shown in Fig. 7.10.

We also performed experiments comparing the enhanced planner exploiting the algorithm hierarchy with the original classical planner for the text mining use case. The

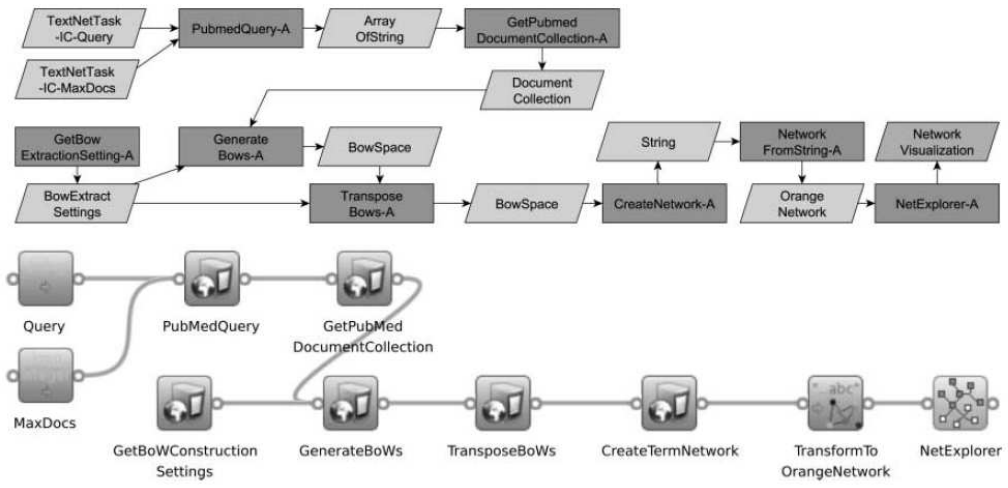


Figure 7.10: A schema of automatically generated abstract workflow and its executable instantiation in the Orange4WS environment. The underlying knowledge discovery task is a text-mining scenario of Section 7.2.1 for the analysis of a graphs of terms, obtained by querying the PubMed database using a publicly accessible web service.

results are shown in Table 7.4. The results show the same trends as the previous experiments presented in Chapter 6.

7.2.3 Discussion

We have developed a proof of concept integration of our methodology for automatic workflow construction in the Orange4WS knowledge discovery platform. We demonstrated its applicability for a text mining, which required using a combination of algorithms available locally and algorithms available as web services. In future work, we will explore adding means of semi-automatic annotation of web services. The planner will also be a subject of future improvements as we aim to incorporate the ability of satisfying user-defined constraints and preferences.

Chapter 8

Conclusions

Knowledge discovery tasks in science and engineering require mining heterogeneous and structured data and knowledge sources. It has been recognized for some years that an important factor in successfully addressing the challenges of these tasks is the ability to efficiently utilize the knowledge mastered by experts in the domain of discovery. There has been a lot of effort to formalize the current domain knowledge by creating domain ontologies. One of the key challenges for data mining is therefore to enable the data mining algorithms to efficiently incorporate this domain knowledge.

The first part of this work is therefore devoted to developing a framework for relational data mining with background knowledge including domain knowledge available in form of ontologies. Standard inductive logic programming offers means for dealing with structured data and background knowledge and produces results in form of rules, which are easily interpretable by humans and also in a formal representation and thus be easily added to the existing body of knowledge. Therefore ILP framework is a suitable platform for learning with ontologies.

There exists a wide range of algorithms for learning from propositional (attribute-value) representation. To be able to reuse the these algorithms, we adopted the approach of relational data mining through propositionalization and method of relational features construction. We enhanced both steps of the propositionalization procedure. We have developed a special refinement operator exploiting taxonomies on terms (concepts) and predicates (relations), which leads to an order of magnitude speedup during feature generation. Since the generated features also form a taxonomy, we have adapted a propositional rule learning algorithm to take this taxonomy into account achieving a significant speedup and more compact rules.

Testing and deployment of the developed algorithms lead to identification of the following challenges: adding results to the existing body of knowledge for reasoning and retrieval, efficient reuse of specialized third party algorithms, flexible modifications of the framework for different data and result types. This motivated an investigation of the possibilities to formalize the main ingredients of the knowledge discovery task: task formulation, input data and knowledge, algorithms and procedures and results.

Since none of the ontologies available at the time fulfilled the requirements of encompassing learning from diverse highly structured data and provided support for planning and reuse of workflows, we developed the Knowledge Discovery ontology (KD ontology). The developed KD ontology was used to annotate algorithms for relational data mining available within the RDM Manager tool and algorithms available in the Orange4WS data mining platform. We proposed a subontology for representing data mining workflows in such a way that they can be considered as algorithms and thus allow encapsulating often occurring workflows and constructing hierarchical workflows. The workflows subontology is compatible with OWL-S standard for annotation of web services.

As has been already stated, the present day knowledge discovery tasks require a combination of diverse specialized algorithms in complex workflows, which are difficult to assemble manually for someone not expert in data mining. Once all the important components of the workflows have been formalized using the knowledge discovery ontology, planning and reasoning can be used for automatic workflow construction.

We have developed a methodology for automatic composition of abstract workflows based on task formulation using the KD ontology, which are proposed to the user and can be instantiated interactively. Our methodology focuses on workflows for complex knowledge discovery tasks dealing with structured data and background knowledge, while the previous studies deal only with classical propositional data mining tasks or are specialized for one domain only.

Automatic workflow composition was addressed as a planning task. Three versions of the planning algorithm were developed examining the possibilities of integration of planning and ontological reasoning. The baseline approach consists of converting the task and algorithms description from ontology to standard planning formalism PDDL and subsequent use of classical planning algorithm. The second approach implements planning with obtaining suitable next step by querying the KD ontology using SPARQL query language and Pellet reasoner. The last version goes one step beyond classical planning by exploiting the taxonomy of algorithms provided by the KD ontology. It significantly improves the scalability of the planner with respect to number of available algorithms. Moreover, the taxonomy of algorithms can also be exploited in the presentation of the workflows to the user.

The proposed methodology for constructing workflows was successfully applied in two domains (bioinformatics and product engineering). The workflows generated by our algorithm were complex, but reasonable in that there was no apparent way of simplifying them while maintaining the desired functionality. The developed methodology was also integrated into the Orange4WS knowledge discovery platform.

In future work we plan to extend the ontology by descriptions of available computational resources. This will enable us to produce workflows optimized for execution in a given computing environment as a step towards future automated generation of workflows of data mining services available on the web. We also want to extend the modeling of constraints on the algorithms and workflows and to align the ontology to a top-level ontology. Furthermore, we want to introduce more complex heuristics for evaluating the workflows and metrics for workflow similarity and focus on planners more tightly integrating the planner with a reasoner.

Bibliography

[1] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M.-T. Schmidt, A. Sheth, and K. Verma. Web service semantics - WSDL-S. W3C Member Submission 7 November 2005, 2005.

[2] A. S. Ali, O. Rana, and I. Taylor. Web services composition for distributed data mining. In *Proc. of the 2005 IEEE International Conference on Parallel Processing Workshops, ICPPW'05*, 2005.

[3] M. A. Alsam, S. Auer, and J. Shen. From BPEL4WS process model to full OWL-S ontology. In *Demos and Posters of the 3rd European Semantic Web Conf.*, 2006.

[4] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, I. Trickovic, and S. Weerawarana. Business process execution language for web services version 1.1. Specification, 2003.

[5] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook, Theory, Implementation and Applications*. Cambridge University Press, 2003.

[6] L. Badea and S.-W. Neinhuys-Cheng. A refinement operator for descriptionn logics. In *Inductive Logic Programming*, LNAI 1866, pages 40–59, 2000.

[7] S. Battle, A. Bernstein, H. Boley, B. Grosof, M. Gruninger, R. Hull, M. Kifer, D. Martin, S. McIlraith, D. McGuinness, J. Su, and S. Tabet. Semantic web services language (SWSL). W3C Member Submission 9 September 2005, 2005.

[8] A. Bernstein and M. Deanzer. The NExT system: Towards true dynamic adaptions of semantic web service compositions (system description). In *Proc. of the 4th European Semantic Web Conference (ESWC'07))*, volume 4519 of *LNCS*, pages 739–748. Springer, 2007.

[9] A. Bernstein, F. Provost, and S. Hill. Towards intelligent assistance for a data mining process: An ontology-based approach for cost-sensitive classification. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):503–518, 2005.

[10] M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kotter, T. Meinl, P. Ohl, C. Sieb, K. Thiel, and B. Wiswedel. KNIME: The Konstanz Information Miner. In *Studies in Classification, Data Analysis, and Knowledge Organization*, pages 319–326. Springer, 2007.

[11] A. Blum and M. Furst. Fast planning through planning graph analysis. *Artificial intelligence*, 90:281–300, 1997.

[12] B. Bordbar, G. Howells, M. Evans, and A. Staikopoulos. Model transformation from OWL-S to BPEL via SiTra. In *Proc. of ECMDA 07*, 2007.

[13] R. J. Brachman, V. P. Gilbert, and H. J. Levesque. An essential hybrid reasoning system: Knowledge and symbol level accounts of KRYPTON. In *Proceedings of IJCAI-85*, 1985.

[14] P. Brezany, I. Janciak, and A. M. Tjoa. Ontology-based construction of grid data mining workflows. In *Data Mining with Ontologies: Implementations, Findings and Frameworks*. IGI Global, 2007.

[15] D. Brickley and R. V. Guha (Eds.). RDF vocabulary description language 1.0: RDF schema. W3C Recommendation, 2004.

[16] M. Buchheit, F. M. Donini, and A. Schaerf. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*, 1:109–138, 1993.

[17] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. Springer, 1990.

[18] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, and R. Wirth. CRISP-DM 1.0 step-by-step data mining guide, 2000.

[19] W. Chen, M. Kifer, and D. S. Warren. Hilog: A foundation for higher-order logic programming. *Journal of Logic Programming*, 15(3):187–230, 1993.

[20] W. Cheung, X.-F. Zhang, Z.-W. Luo, and F. Tong. Service-oriented distributed data mining. *IEEE Internet Computing*, 10(4):4454, 2006.

[21] W. Cohen and H. Hirsh. Learning the classic description logic: Theoretical and experimental results. In *Proceedings of teh 4th International Conference on Principles of Knowledge Representation and Reasoning*, pages 121–133, 1994.

[22] A. Congiusta, D. Talia, and P. Trunfio. Distributed data mining services leveraging WSRF. *Future Generation Computer Systems*, 23(1):34–41, 2007.

[23] The Gene Ontology Consortium. Gene ontology: tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.

[24] The OBI Consortium. The ontology for biomedical investigations, 2009.

[25] L. de Raedt, editor. *Logical and Relational Learning*. Cognitive Technologies. Springer, 2008.

[26] E. Deelman, J. Blythe, G. Yolanda, C. Kesselman, S. Koranda, A. Lazzarini, G. Mehta, M. A. Papa, and K. Vahi. Pegasus and the pulsar search: From metadata to execution on the grid. In *Parallel Processing and Applied Mathematics*, 2004.

[27] J. Demsar, B. Zupan, and G. Leban. Orange: From experimental machine learning to interactive data mining. White Paper, 2004.

[28] C. Diamantini, D. Potena, and M. Panti. Developing and open knowledge discovery support system for a networked environment. In *Proc. of the 2005 Int. Symp. on Collaborative Technologies and Systems*, pages 274–281, 2005.

[29] C. Diamantini, D. Potena, and E. Storti. KDDONTO: An ontology for discovery and composition of kdd algorithms. In *In SoKD: ECML/PKDD 2009 Workshop on Third Generation Data Mining: Towards Service-oriented Knowledge Discovery*, pages 13–24, 2009.

[30] C. Diamantini, D. Potena, and E. Storti. Ontology-driven KDD process composition. In *IDA 2009*, volume 5772 of *LNCS*, pages 285–296, 2009.

[31] B. Dolšak, I. Bratko, and A. Jezernik. Finite element mesh design: An engineering domain for ILP application. In *Proc. of ILP 1994*, GMD-Studien 237, pages 305–320, 1994.

[32] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In *Proceedings of KR-91*, 1991.

[33] F. M. Donini, M. Lenzerini, D. Nardi, A. Schaerf, and W. Nutt. An epistemic operator for description logics. *Artificial Intelligence*, 100 (1-2):1998, 225-274.

[34] S. Džeroski. Towards a general framework for data mining. In *Knowledge Discovery in Inductive Databases - 5th Int. Workshop, KDID'06*, volume 4747 of *LNCS*, pages 259–300. Springer, 2007.

[35] D. Martin (ed.). OWL-S: Semantic markup for web services. W3C Member Submission, 2004.

[36] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. Knowledge discovery and data mining: Towards a unifying framework. In *Proc. of KDD-96*, pages 82–88, 1996.

[37] M. Fernandez, A. Perez, and N. Juristo. METHONTOLOGY: from ontological art towards ontological engineering. In *Proc. of the AAAI97 Spring Symposium Series on Ontological Engineering*, pages 33–40, 1997.

[38] R. Fikes and N. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[39] A. Frisch. Sorted downward refinement: Building background knowledge into a refinement operator for ILP. In *Proc. of ILP 1999*, LNAI 1634, page 104115, 1999.

[40] A. Gangemi, S. Borgo, C. Catenacci, and J. Lehmann. Task taxonomies for knowledge content. METOKIS Deliverable D07, 2005.

[41] A. Gomez-Perez, M. Fernandez-Lopez, and O. Corcho. Methodologies and methods for building ontologies. In *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*, pages 107–197. Springer, 2004.

[42] A. Gomez-Perez, M. Fernandez-Lopez, and O. Corcho. *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*. Springer, 2004.

[43] P. Grenon. BFO in a nutshell: A bi-categorial axiomatization of BFO and comparison with DOLCE. IFOMIS reports, June 2003.

[44] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: combining logic programs with description logic. In *Proc. of the 12th Int. Conf. on World Wide Web*, pages 48–57. ACM Press New York, NY, USA, 2003.

[45] T. R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43(4-5):907–928, 1995.

[46] D. Guedes, W. J. Meira, and R. Ferreira. Anteater: A service-oriented architecture for high-performance data mining. *IEEE Internet Computing*, 10(4):36–43, 2006.

[47] M. Hilario, A. Kalousis, P. Nguyen, and A. Woznica. A data mining ontology for algorithm selection and meta-mining. In *In SoKD: ECML/PKDD 2009 Workshop on Third Generation Data Mining: Towards Service-oriented Knowledge Discovery*, pages 76–87, 2009.

[48] J. Hoffmann. Towards efficient belief update for planning-based web service composition. In *Proc.of ECAI 2008*, pages 558–562, 2008.

[49] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence research*, 14:2001, 253-302.

[50] T. Hofweber. Logic and ontology. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy (Fall 2011 Edition)*. 2011.
http://plato.stanford.edu/archives/fall2011/entries/logic-ontology/.

[51] A. Kalyanpur, D. Jiménez Pastor, S. Battle, and J. A. Padget. Automatic mapping of OWL ontologies into Java. In *Proc. of SEKE 2004*, pages 98–103, 2004.

[52] N. L. Khac, M. T. Kechadi, and J. Carthy. Admire framework: Distributed data mining on data grid platforms. In *Procs. of First Int. Conf. on Software and Data Technologies*, volume 2, pages 67–72, 2006.

[53] J.-U. Kietz. Learnability of description logic programs. In *Inductive Logic Programming: 12th International Conference*, LNCS Volume 2583, 2002.

[54] J.-U. Kietz and S. Džeroski. Inductive logic programming and learnability. *SIGART Bulletin*, 5(1):22–32, 1994.

[55] J.-U. Kietz, F. Serban, A. Bernstein, and S. Fischer. Towards cooperative planning of data mining workflows. In *In SoKD: ECML/PKDD 2009 Workshop on Third Generation Data Mining: Towards Service-oriented Knowledge Discovery*, pages 1–12, 2009.

[56] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of ACM*, 42:741–843, 1995.

[57] R. D. King, K. E. Whelan, F. M. Jones, P. K. G. Reiser, C. H. Bryant, S. H. Muggleton, D. B. Kell, and S. G. Oliver. Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, 427:247–252, 2004.

[58] M. Klusch, A. Gerber, and M. Schmidt. Semantic web service composition planning with OWLS-Xplan. In *Procs of 1st Intl. AAAI Fall Symposium on Agents and the Semantic Web*, 2005.

[59] M.-A. Krogel, S. Rawles, P. A. Flach, N. Lavrač, and S. Wrobel. Comparative evaluation of approaches to propositionalization. In *Inductive Logic Programming: 13th International Conference, ILP 2003*, volume 2835 of *LNAI*, pages 197–214, 2003.

[60] A. Kumar, M. Kantardzic, P. Ramaswamy, and P. Sadeghian. An extensible service oriented distributed data mining framework. In *Proc. IEEE/ACM Intl. Conf. on Machine Learning and Applications*, 2004.

[61] H. Lausen, A. Polleres, and D. Roman (eds.). Web service modeling ontology (WSMO). W3C Member Submission 3 June 2005, 2005.

[62] N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood, 1994.

[63] F. Lécué, A. Delteil, and A. Léger. Applying abduction in semantic web service composition. In *Procs of the IEEE International Conference on Web Services (ICWS 2007)*, pages 94–101, 2007.

[64] A. Levy and M.-C. Rousset. The limits on combining recursive horn rules with description logics. In *Proc. of the 13th Nat. Conf. on AI and 8th Innovative Applications of AI Conf., AAAI 96, IAAI 96*, pages 577–584, 1996.

[65] A. Levy and M.-C. Rousset. Combining Horn rules and description logics in CARIN. *Artificial Intelligence*, 104:165209, 1998.

[66] Y. Li and Z. Lu. Ontology-based universal knowledge grid: Enabling knowledge discovery and integration on the grid. In *Procs of the 2004 IEEE International Conference on Services Computing (SCC'04)*, 2004.

[67] F. A. Lisi and D. Malerba. Ideal refinement of descriptions in AL-Log. In *Proc. of ILP 2003*, LNCS 2835, pages 215–232, 2003.

[68] Z. Liu, A. Ranganathan, and A. Riabov. A planning approach for message-oriented semantic web service composition. In *Proc of the Nat. Conf. on AI*, volume 5(2), pages 1389–1394, 2007.

[69] R. M. MacGregor. A description classifier for predicate calculus. In *Proceedings of AAAI-94*, 1994.

[70] O. Maimon and L. Rokach. *The Data Mining and Knowledge Discovery Handbook*. Springer, 2005.

[71] D. Martin, M. Paolucci, and M. Wagner. Toward semantic annotations of web services: OWL-S from the SAWSDL perspective. In *Proc. of OWL-S: Experiences and Directions - a workshop at ESWC 2007*, 2007.

[72] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, and A. Oltramari. Ontology library. WonderWeb Deliverable D18, 2003.

[73] R. S. Michalski. Understanding the nature of learning: Issues and research directions. In *Machine Learning: An Artificial Intelligence Approach*, pages 3–25. Morgan Kaufmann, Los Altos, CA, 1986.

[74] I. Mierswa, M. Scholz, R. Klinkenberg, M. Wurst, and T. Euler. Yale: Rapid prototyping for complex data mining tasks. In *KDD*, pages 935–940. ACM Press, 2006.

[75] P. Mika, D. Oberle, A. Gangemi, and M. Sabou. Foundations for service ontologies: Aligning OWL-S to DOLCE. In *The 13th International World Wide Web Conference Proceedings*, pages 563–572, 2004.

[76] K. Morik and M. Scholz. The MiningMart approach to knowledge discovery in databases. In *Intelligent Technologies for Information Analysis*, pages 47–65, 2004.

[77] B. Motik. On the properties of metamodeling in OWL. In *Proc. of International Semantic Web Conference 2005*, pages 548–562, 2005.

[78] S. Muggleton. Inverse entailment and Progol. *New Generation Computing Journal*, 13:245–286, 1995.

[79] S-H. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*. LNAI 1228. Springer, 1997.

[80] A. Oberle. *Semantic Management of Middleware*. Semantic Web and Beyond. Springer, 2006.

[81] A. Paes, F. Železný, G. Zaverucha, D. Page, and A. Srinivasan. ILP through propositionalization and k-term DNF learning. In *Proc. of the 16th Conference on ILP*. Springer, 2007.

[82] P. Panov and S. Džeroski, 2009. Personal communication.

[83] P. Panov, S. Džeroski, and L. N. Soldatova. Ontodm: An ontology of data mining. In *IEEE ICDM Workshops 2008*, pages 752–760, 2008.

[84] P. Patel-Schneider, P. Hayes, and I. Horrocks. OWL web ontology language semantics and abstract syntax. W3C Recommendation, 2004.

[85] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, and P. Traverso. Planning and monitoring web service composition. In *Proc. of AIMSA 2004*, pages 106–115, 2004.

[86] V. Podpečan, M. Zemenová, and N. Lavrač. Orange4WS environment for service-oriented data mining. *The Computer Journal*, 55(1):82–98, 2012.

[87] A. Rowe, D. Kalaitzopoulos, M. Osmond, M. Ghanem, and Y. Guo. The Discovery Net system for high throughput bioinformatics. *Bioinformatics*, 19:225–231, 2003.

[88] U. Ruckert and S. Kramer. Stochastic local search in k-term DNF learning. In *Proc. of the 20th ICML*, pages 648–655, 2003.

[89] E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artif. Intell.*, 5(2):115–135, 1974.

[90] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):2007, 51-53.

[91] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau. HTN planning for web service composition using SHOP2. *Journal of Web Semantics*, 1(4):377–396, 2004.

[92] D. Smith and D. Weld. Temporal planning with mutual exclusion reasoning. In *Proc. of the 1999 International Joint Conference on Artificial Intelligence (IJCAI-1999)*, pages 326–333, 1999.

[93] J. F. Sowa. *Knowledge representation: logical, philosophical and computational foundations.* Brooks/Cole Publishing Co., 2000.

[94] A. Srinivasan. The Aleph manual version 4, 2003.

[95] S. Staab, R. Studer, H. Schnurr, and Y. Sure. Knowledge processes and ontologies. *IEEE Intelligent Systems*, 16(1):26–34, 2001.

[96] V. Stankovski, M. Swain, V. Kravtsov, T. Niessen, D. Wegener, J. Kindermann, and W. Dubitzky. Grid-enabling data mining applications with DataMiningGrid: An architectural perspective. *Future Generation Computer Systems*, 24(4):259–279, 2008.

[97] M. Strauchmann, T. Haase, E. Jamin, H. Cherfi, M. Renteria, and C. Masia-Tissot. Coaction of semantic technology and virtual reality in an integrated engineering environment. In *KCAP Workshop on Knowledge Management and Semantic Web for Engineering Design*, 2007.

[98] A. Suyama, N. Negishi, and T. Yamagchi. Composing inductive applications using ontologies for machine learning. In *Proc. of the First International Conference on Discovery Science*, pages 429–431, 1998.

[99] I. Taylor, M. Shields, I. Wang, and A. Harrison. The Triana workflow environment: Architecture and applications. In I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, editors, *Workflows for e-Science*, pages 320–339. Springer, 2007.

[100] I.J. Taylor, E. Deelman, D.B. Gannon, and M. Shields, editors. *Workflows for e-Science, Scientific Workflows for Grids.* Springer, 2007.

[101] I. Trajkovski, F. Železný, N. Lavrač, and J. Tolar. Learning relational descriptions of differentially expressed gene groups. *IEEE Trans. Sys Man Cyb C*, 38(1):16–25, 2008.

[102] I. Trajkovski, F. Železný, J. Tolar, and N. Lavrač. Relational subgroup discovery for descriptive analysis of microarray data. In *Proc. of CompLife 06.* Springer, 2006.

[103] M. Žáková, P. Křemen, F. Železný, and N. Lavrač. Automating knowledge discovery workflow composition through ontology-based planning. *IEEE T. Automation Science and Engineering*, 8(2):253–264, 2011.

[104] M. Žáková and F. Železný. Exploiting term, predicate, and feature taxonomies in propositionalization and propositional rule learning. In *ECML 2007: 18th European Conference on Machine Learning*, 2007.

[105] M. Žáková, F. Železný, J. A. Garcia-Sedano, C. Massia-Tissot, N. Lavrač, P. Křemen, and J. Molina. Relational data mining applied to virtual engineering of product designs. In *Proc. of the 16th Int. Conference on Inductive Logic Programming*, pages 439–453, 2006.

[106] F. Železný and N. Lavrač. Propositionalization-based relational subgroup discovery with RSD. *Machine Learning*, 62(1-2):33–63, 2006.

[107] R. Wirth, C. Shearer, U. Grimmer, T. P. Reinartz, J. Schloesser, C. Breitner, R. Engels, and G. Lindner. Towards process-oriented tool support for knowledge discovery in databases. In *Proc. of the First European Symposium on Prin- ciples of Data Mining and Knowledge Discovery*, volume 1263, pages 243–253, 1997.

[108] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann, San Francisco, 2005.

[109] Q. Yang and X. Wu. 10 challenging problems in data mining research. *Intl. Jrnl. of Inf. Technology & Decision Making*, 5(4):597–604, 2006.

[110] Y. Zou, T. Finin, and H. Chen. F-OWL: An inference engine for the semantic web. In *Formal Approaches to Agent-Based Systems*, LNCS 3228, pages 238–248, 2004.