UNIVERSITÉ PARIS-SUD 11

Thèse de doctorat en informatique

Comprendre le Web caché
Understanding the Hidden Web

Pierre Senellart

12 décembre 2007

Comprendre le Web caché
Understanding the Hidden Web

Pierre SENELLART

Résumé

Le Web caché (également appelé Web profond ou Web invisible), c'est-à-dire la partie du Web qui n'est pas directement accessible par des hyperliens, mais à travers des formulaires HTML ou des services Web, est d'une grande valeur, mais difficile à exploiter. Nous présentons un processus pour la découverte, l'analyse syntaxique et sémantique, et l'interrogation des services du Web caché, le tout de manière entièrement automatique. Nous proposons une architecture générale se basant sur un entrepôt semi-structuré de contenu imprécis (probabiliste). Nous fournissons une analyse détaillée de la complexité du modèle d'arbre probabiliste sous-jacent. Nous décrivons comment une combinaison d'heuristiques et de sondages du Web peut être utilisée pour comprendre la structure d'un formulaire HTML. Nous présentons une utilisation originale des champs aléatoires conditionnels (une méthode d'apprentissage supervisé) de manière non supervisée, sur une annotation automatique, imparfaite et imprécise, basée sur la connaissance du domaine, afin d'extraire l'information pertinente de pages de résultat HTML. Afin d'obtenir des relations sémantiques entre entrées et sorties d'un service du Web caché, nous étudions la complexité de l'obtention d'une correspondance de schémas à partir d'instances de bases de données, en se basant uniquement sur la présence des constantes dans ces deux instances. Nous décrivons enfin un modèle de représentation sémantique et d'indexation en compréhension de sources du Web caché, et débattons de la manière de traiter des requêtes de haut niveau à l'aide de telles descriptions.

Abstract

The hidden Web (also known as deep or invisible Web), that is, the part of the Web not directly accessible through hyperlinks, but through HTML forms or Web services, is of great value, but difficult to exploit. We discuss a process for the fully automatic discovery, syntactic and semantic analysis, and querying of hidden-Web services. We propose first a general architecture that relies on a semi-structured warehouse of imprecise (probabilistic) content. We provide a detailed complexity analysis of the underlying probabilistic tree model. We describe how we can use a combination of heuristics and probing to understand the structure of an HTML form. We present an original use of a supervised machine-learning method, namely conditional random fields, in an unsupervised manner, on an automatic, imperfect, and imprecise, annotation based on domain knowledge, in order to extract relevant information from HTML result pages. So as to obtain semantic relations between inputs and outputs of a hidden-Web service, we investigate the complexity of deriving a schema mapping between database instances, solely relying on the presence of constants in the two instances. We finally describe a model for the semantic representation and intensional indexing of hidden-Web sources, and discuss how to process a user's high-level query using such descriptions.

Mots clefs : Web caché, Web profond, bases de données, extraction d'informations, complexité

Keywords: hidden Web, deep Web, databases, information extraction, complexity

À l'exception de l'annexe A, qui propose une traduction en français de l'introduction, du chapitre 1 et de la conclusion, cette thèse est rédigée en anglais.

With the exception of Appendix A, a translation into French of the introduction, Chapter 1, and conclusion, this thesis is written in English.

Cette thèse est rédigée à l'aide du système de composition de documents LaTeX, avec les polices de caractères Adobe Garamond Pro, Math Design, Myriad Pro et Bitstream Vera Sans Mono. Les bandes dessinées introduisant chaque chapitre sont extraites de *The Complete Peanuts*, de Charles M. Schultz, une collection publiée depuis 2004 par Fantagraphics Books. Ils sont reproduits ici en vertu du droit de citation.

This thesis is written with the help of the typesetting system LaTeX, in the font families Adobe Garamond Pro, Math Design, Myriad Pro, and Bitstream Vera Sans Mono. The comic strips that introduce each chapter are from *The Complete Peanuts*, by Charles M. Schultz, a series published since 2004 by Fantagraphics Books. They are reprinted here under fair use.

Contents

List of Algorithms

List of Figures

List of Tables

Turing believes machines think.
Turing lies with men.
Therefore machines do not think.

Alan Mathison Turing

Acknowledgments

ALL RIGHT, MOTHER! I'LL DO IT!
I SAID I'D DO IT, AND I'LL DO IT!
YOU DON'T HAVE TO KEEP REMINDING ME! I'LL DO IT!
YOU DON'T HAVE TO PESTER ME! I SAID I'D DO IT, AND I'LL DO IT!!
Dear Grandma, Thank you for the Christmas present.
2-25
SCHULZ

My first thoughts in writing these acknowledgments go to my advisor, Serge Abiteboul. I met him six years ago, as he was giving a presentation talk to first-year students from École normale supérieure at INRIA Rocquencourt. His talk, about the Web graph and its butterfly shape, was fascinating and sparked my interest for the Web as a research topic. I later followed his course on semi-structured data and XML, and Serge (along with the other lecturers, who came from his group) managed to transform my initial skepticism about XML into a true passion for all technologies related to it. I then had the chance to do my master's thesis research under his supervision on a somewhat esoteric topic (identification of logical Web sites), in such conditions that made me eager to embark on a longer adventure with my PhD. Serge was to me an ideal advisor during the last three years, giving me a lot of free rein while always remaining available despite a busy schedule, supporting me when I tended to spread my activities while remaining just as directing as required, providing me with many opportunities of collaboration with other people. He has made me share his taste for both theory and practical applications, and I admire his proficiency in both. I shall also remember the numerous times where I felt exhilarated after a productive meeting with him, happy at actually working *with* him and benefiting of his experience. I cannot thank him enough for all of this, and my only pang at finishing my thesis is to stop being his student.

I would also like to thank the members of my thesis jury, and especially Patrick Gallinari and Val Tannen, who accepted to take the time to review my thesis.

A number of works presented in this thesis have resulted from collaborations; let me acknowledge them in chronological order. I had many interactions with researchers from the INRIA Mostrare group that led to fruitful work in common, particularly with Rémi Gilleron, Florent Jousse, Patrick Marty, Daniel Muschick, and Marc Tommasi. Daniel Muschick actually did his master's thesis on this collaboration, and I have been quite impressed at his production and motivation. It is an innocent-looking question of Rémy's at a seminar I gave in Lille who led me to consider probabilistic XML models, for which I thank him much. I had the honor of spending three months in Oxford, working under the supervision of Georg Gottlob. Oxford is a very nice city, and Georg's quick mind and skill at theoretical problems is equal to his impressive fluency in languages. It was a pleasure working with him and I learned a lot about theory of databases there. I finally had the chance to welcome Avin Mittal, from IIT Bombay, on a three-month internship in our group. I am really pleased with the work he did, not only following my directions but also studying autonomously relevant literature and coming up with productive ideas. He produced a system for probing the hidden Web that does work, which is an impressive feat in the short time he was there.

I had fruitful discussions about my research with many other people: Joachim Niehren and Fabien Torre from INRIA Mostrare; Michael Benedikt from University of Oxford; Nada Abdallah, Andrei Arion, Pierre Bourhis, Bogdan Cautis, François Goasdoué, Sandrine-Dominique Gouraud, Ioana Manolescu, Bogdan Marinoiu, Benjamin Nguyen, Nathalie Pernelle, Neoklis Polyzotis, Nicoleta Preda, Philippe Rigaux, Marie-Christine Rousset, Luc Segoufin, Michalis Vazirgiannis, and Victor Vianu from (or not far from) INRIA Gemo. Please forgive in advance all omissions that I am bound to have made in such a list. Yet another innocent discussion with Nathalie Pernelle was at the origin of the idea of combining structural machine learning and domain knowledge, that led to the collaboration with INRIA Mostrare, and I would like to thank her specially for this.

Let me also thank my friends and family, that have supported me these last few years and who helped me overcome some of the most stressful moments. I shall not try to give an exhaustive list, but let me at least mention Guillaume, Marc, and Sandra, my brother Jean who helped me discover the industrial world, and my sister Martine who has always been there for me. Her expertise in *Peanuts*-o-logy made possible the introductory comic strips that open each chapter. I shall also be forever grateful to N., for his continuous support and his belief in me.

Closest to my heart is my companion, Yann. Not only was he incredibly supportive, unceasingly repeating that I was not as terrible in what I do as I feel I am, and did he help me relax from work on many an occasion, but discussions with him were also one of my main sources of inspiration in my research. It is great to be able to explain to one's partner the details of one's work, and to be understood by him. I owe Yann much in my personal and professional life, and thank him enormously. I just hope to be able to provide him with as much as he gives me.

Let me finally acknowledge Donald Knuth, Leslie Lamport, Till Tantau and Bram Moolenaar for the amazing software they respectively produced, which eased and even made fun at times the writing of this thesis.

Introduction

" THIS IS MY REPORT ON MOUNTAINS..."
"MOUNTAINS ARE SO YOU CAN CLIMB TO THE TOP AND SEE WHERE YOU'VE BEEN"
5-2
© 1991 United Feature Syndicate, Inc.
DO YOU THINK I WENT INTO TOO MUCH DETAIL?
NO, I'M SURE THE TEACHER WILL APPRECIATE YOUR RESEARCH
SIX MINUTES IS ENOUGH TIME TO SPEND ON ANY PAPER..

Access to Web information today primarily relies on keyword-based search engines. These search engines deal with the *surface Web*, the set of Web pages directly accessible through hyperlinks, mostly ignoring the vast amount of information hidden behind forms, that composes the *hidden Web* (also known as *deep Web* or *invisible Web*). The topic of this thesis is the automated exploitation of hidden-Web resources, and more precisely the discovery, analysis, understanding, and querying of such resources. Clearly, this goes way beyond the scope of a single PhD thesis and we shall not provide a complete answer. We present a general framework and propose solutions to some particular issues raised by the exploitation of the hidden Web.

In Chapter 1, we introduce a general framework for understanding the hidden Web. Our focus is on *fully automatic* systems, not requiring any human intervention. As the problem of understanding hidden-Web resources is without doubt AI-complete, we limit our interest to a specific *application domain* relying on available *domain-specific knowledge*. Our approach is *content-centric* in the sense that the core of the system consists of a content warehouse of hidden-Web services, with independent modules enriching our knowledge of these services so they can be better exploited. For instance, a module may be responsible for discovering new relevant services (e.g., URLs of forms); another one of analyzing the structure of forms; etc. These different "agents" are then combined into a workflow* that is described in Chapter 1. Specific aspects of these agents are described in subsequent chapters. An important functionality, that of discovering relevant services, is only briefly discussed in Section 1.4.1 of Chapter 1. Such services can be obtained, for example, by querying search engines or by focused crawling.

Most of the ideas presented in this thesis have been implemented in prototype systems. To validate the approach and test the systems, we used the research publication domain. We use here this same application to illustrate our work. It should be stressed that the ideas and software can be used in any application domain, assuming that some domain knowledge is available in a standard form we shall discuss.

The data that we handle is typically rather irregular and often tree-structured. It is thus natural to use a semi-structured data model; we use XML since it is a Web standard. The different agents that cooperate to build the content warehouse are inherently imprecise (since they typically involve either machine-learning techniques or heuristics, both of which prone to imprecision). Thus we use a *probabilistic* XML content warehouse that is queried and probabilistically updated by the different agents. The probabilistic XML model is described in Chapter 2. We introduce *prob-trees* as regular

*Although the workflow is presented as sequential, one can typically improve the quality of the analysis of services by using more complex workflows. To simplify, this will be ignored here.

trees annotated with conjunctions of independent random variables (and their negation). We show how to evaluate efficiently queries and updates in this model and describe an implementation that supports it. We also investigate a number of important theoretical issues about prob-trees and discuss variants of the model. Note that although development of this prob-tree model was motivated by our hidden-Web setting, it is actually quite general and can be applied in a different context.

Consider a service of the hidden Web, say an HTML form, that is relevant to the particular application domain. To understand its semantics, a first step is the analysis of the form, i.e., the structure of the service inputs. This is considered in Chapter 3. We use some heuristics to associate domain concepts to form fields, and then *probe* these fields with domain instances to confirm or infirm these guesses. The core of the validation consists in techniques for distinguishing between *result pages* and *error pages*. We show quite satisfactory experimental results on search forms for research publications.

The next step is to extract information from the results of the form, HTML pages. We describe in Chapter 4 how supervised information-extraction techniques such as *conditional random fields* can be used in an unsupervised manner, with the help of domain knowledge. A *gazetteer* annotates result pages with domain concepts (based on available domain instances that are recognized). This annotation, that is both imprecise and incomplete, is used for bootstrapping the machine learning process.

As a result of these two steps, we understand the structure of the inputs and that of the outputs of the form (with some imprecision of course). It is then easy to wrap the form as a standard Web service described in WSDL.

It is then necessary to understand the semantic relations that exist between the inputs and outputs of a service. This issue is considered in Chapter 5. To simplify, the focus is on a relational setting. We introduce a theoretical framework for discovering relationships between two database instances over distinct and unknown schemata. This framework is grounded in the context of *data exchange*. We formalize the problem of understanding the relationship between two instances as that of obtaining a schema mapping (a set of sentences in some logical language) so that a *minimum repair* of this mapping provides a perfect description of the target instance given the source instance. We show that this definition yields "intuitive" results when applied on database instances derived from each other by basic operations (selections, projections, joins, and so forth). We study the complexity of decision problems related to this optimality notion in the context of different logical languages (e.g., full acyclic tuple-generating dependencies). For this particular problem, our contribution is strictly theoretical; practical tools based on these ideas are left for future work.

At the end of the analysis phase, we have obtained a number of Web services whose semantics is explained in terms of a global schema that is application-specific. In Chapter 6, we discuss the semantic model and the definition of services using a logical Datalog-like notation that takes into account the types of inputs and outputs, relations between them and nestedness of outputs of a service. We show how to answer queries using hidden-Web services. This leads us to studying the problem of answering queries using views in a LAV (*Local As View*) setting, with *binding patterns* restrictions of source accesses.

To summarize, we highlight the three main contributions of our thesis:

1. A general framework for understanding the hidden Web in a fully automatic, unsupervised, manner, and in particular ways to discover the structure of a form and of result pages, and to represent semantically analyzed services of the hidden Web (see Chapters 1, 3, 4, and 6).

2. A probabilistic tree model with query and update capabilities, with a theoretical study and an

implementation (see Chapter 2).

3. A theoretical framework for discovering schema mappings from database instances, along with a detailed complexity analysis (see Chapter 5).

Appendix A is a translation into French of the introduction, Chapter 1 and conclusion of this thesis. It includes an English-to-French lexicon of technical words and phrases used in this thesis. Finally, we mention in Appendix B other works we performed during the time of the thesis that have no direct relation with the hidden Web.

Chapter 1

General Framework

I'M A COWBOY..
HUH...SOME COWBOY! YOU DON'T HAVE A GUN OR ANY BOOTS OR EVEN A HAT!
SURFACE QUALITIES!
2-4
SCHULZ

Since its creation in 1991, the World Wide Web has considerably grown, with today a size of several billions of freely accessible Web pages. But this number only covers the *surface Web*, i.e., the Web pages accessible by following hyperlinks. A large part of the information on the Web may also be found in the *hidden Web*, also known as *deep Web* or *invisible Web*, that provides entry points to databases accessible via HTML forms or Web services. (In the remaining of the thesis, we use the term *service (of the hidden Web)* generically for both.) A 2001 study [Bri00] estimated the data in the hidden Web to be about 500 times that of the surface Web. If such measures are debatable, it is on the other hand unquestionable that with information of the best quality (e.g. *Yellow Pages* services, U.S. *Census Bureau*), the hidden Web is an invaluable source of information.

We shall describe, in this chapter, a general, unsupervised, *fully automatic*, framework for understanding the services of the hidden Web. This approach relies on (i) modules that accomplish various tasks, from the discovery of relevant sources to the analysis of the semantics of such sources and their indexing; (ii) a content-centric architecture, where modules interact with a probabilistic XML content warehouse. We consider here an *intensional* approach, rather than an *extensional* one. By this, we mean that we do not aim at retrieving and storing all the information lying in services of the hidden Web, but at indexing the services themselves so as to answer a user's high-level queries by forwarding the queries to the relevant services, translating them to the expected inputs of each service and integrating the result of the services.

To illustrate, let us consider a Web user interested in available options for a particular car. A traditional search engine would typically return a set of HTML Web pages, e.g., from the maker's Web site. Our goal is a system that would discover (ahead of time) that a particular form provides such information and, at query time, would fill in the form and retrieve the precise answer. Such a semantic interpretation of the hidden Web requires the discovery, analysis and indexing of hidden-Web resources. It requires combining techniques from a number of fields and in particular, machine learning, databases and linguistics.

This is obviously a very broad and difficult problem, especially since we look for an unsupervised approach. An important assumption is thus that we are interested in services relevant to a given *domain of interest*, for which we have some *domain knowledge*, in a specific form that we describe in Section 1.3. Clearly, with human providing feedback, supervised techniques can go further toward a better understanding of the Web. But the kind of unsupervised approach we propose (i) is useful at least as a first step, before human intervention; (ii) is often the only one available for applications when human resources cannot be used; (iii) is essential if we want the system to adapt to the scale and diversity of the Web, as well as its dynamism.

This chapter focuses on the general framework, while subsequent chapters will describe in more detail some of its components. We first discuss relevant work on the hidden Web itself and its analysis in Section 1.1. Our framework is then presented in Section 1.2. We introduce the kind of domain knowledge we consider in Section 1.3, before describing the different components of our architecture in Section 1.4, especially those that will not be presented in a subsequent chapter.

1.1 Related Work

1.1.1 The Hidden Web

The terms *hidden Web* [RGM01], *deep Web* [Bri00], *invisible Web* [RH05] have been used in the literature to describe more or less the same thing: the part of the World Wide Web which is not accessible through hyperlinks, and that is typically constructed from databases. Both terms *hidden* and *invisible* insist on the inaccessibility of this information to search engines, while *deep* insists on the fact that the information lies in databases behind forms, and requires *deeper* crawls than the usual *surface* crawls of search engines. [Bri00] actually makes the analogy between finding information on the Web and sea fishing: while one can harvest some fish by remaining at the surface with shallow fishing, a lot more fish is available in the sea depths. Strictly speaking, it would perhaps be more appropriate to use the phrase *deep Web* in this thesis, rather than *hidden Web*, since we typically focus on databases accessible through forms, whose content may or may not be available on the surface Web—most commercial databases of products, for instance, present their content in a fully browsable form, in order to bring search engine visitors to their Web site. We chose, however, to use *hidden Web* to highlight the fact that understanding the hidden Web in such a way is overcoming the limitation of classical search engines. Besides, *hidden Web* is a much more mysterious phrase that is bound to cause interrogations from neophytes, and, for some reason, *Web caché* sounds much nicer than *Web profond* in French.

A 2001 study from the company BrightPlanet [Bri00] has had much impact on the development of research about the hidden Web. This study uses correlation analysis between search results of different search engines to estimate the size of the hidden Web; they find that it contains between 43,000 and 96,000 Web databases, with around 500 times more content than the surface Web. In other words, the largest part by far of the content present on the Web is not exploitable by classical search engines! Although this kind of estimation is by nature quite imprecise, other more recent works [CHL+04, HPZC07] confirm this order of magnitude with another estimation approach (random sampling of IP addresses) and come up with a number of around 400,000 databases (this takes into account the growth of the Web between the times of the two studies). Moreover, even directories that are specializing in listings databases on the Web—a large list of these is given in [RH05]*—have poor coverage of the services of the hidden Web (15 % at best, cf. [HPZC07]). This is a clear motivation for systems that discover, understand and integrate hidden-Web services.

*Such directories only list databases, they do not provide ways to query them from a common interface or to integrate their results.

1.1.2 Systems for Exploiting Information of the Hidden Web

A survey of a number of systems that deal with services of the hidden Web can be found in [RH05]. Two different approaches exist: extensional (retrieving information from the hidden Web and storing it locally to process it), or intensional (analyzing services to understand their structure, store this description, and use it to forward users' queries to the services). The framework presented in this thesis is an instance of this second kind. We discuss next both approaches. Only complete systems for dealing with the hidden Web are presented; more specific works are discussed in further sections and chapters when relevant.

One of the first practical works on the hidden Web [RGM01] follows the extensional strategy: HiWE is a general system for representing forms and mapping their fields to concepts (relying heavily on human annotation), so as to retrieve result pages that are cached in a local index. It is interesting to note that this extensional approach is favored by researchers from the leading search engine Google [Goo] in [MHC+06] (the authors call it a *surfacing* approach) for domain-independence, effectiveness and efficiency issues, even though most semantics are lost in the process, and this puts a heavy load on the source that is entirely siphoned.

Most active research about the intensional indexing of the hidden Web [HC03, HMYW04, WYDM04, ZHC04, CHZ05, ZHC05, WDYM05, WDY06, CCZ07] comes from a group at University of Illinois at Urbana-Champaign, with some external collaborations, especially with Binghamton University. Two systems that have been developed in this context are MetaQuerier and WISE-Integrator. The aim is, as in our case, to discover and integrate sources of the hidden Web, to query them in a uniform way. There is a strong focus in these works on schema matching between different sources. MetaQuerier [HC03, CHZ05] uses a *holistic* approach, that is, matching a number of different sources at one, rather than one source with another. In particular, the global schema is typically not predefined but derived from a clustering of the fields from different sources. These works essentially focus on the analysis of the syntax of the form and the discovery of concepts in its different fields (some recent works [WDYM05, CCZ07] also deal with the analysis of the response pages). It is actually quite complicated to understand the relations between all previously cited papers, but [CHZ05] comes the closest to a high-level view of the entire system. We shall discuss some of these works further in subsequent chapters.

1.2 Framework

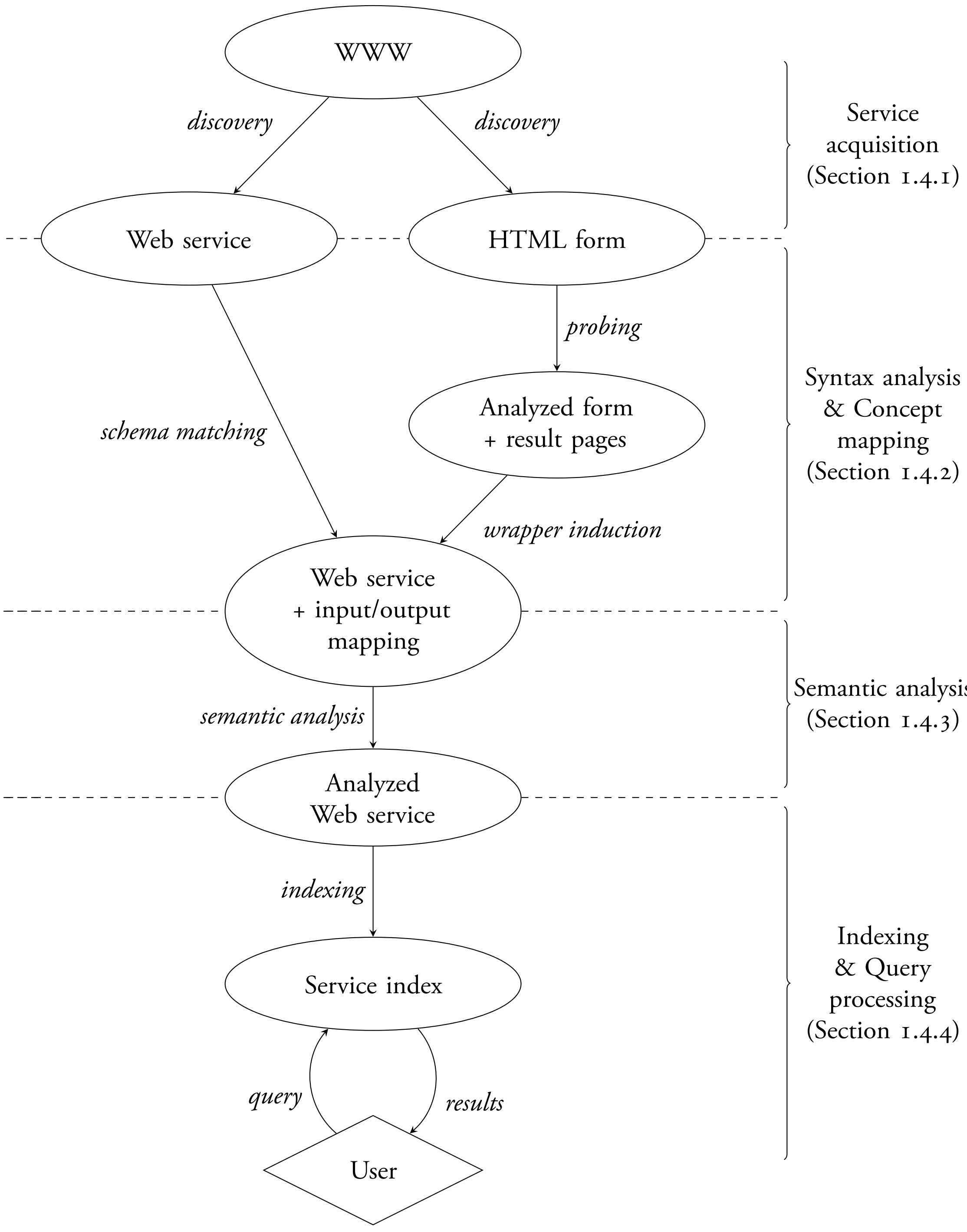
We describe our general framework. A simplified functional architecture is shown in Figure 1.1. The components will be detailed in Section 1.4. The different steps are:

1. Services of interest are first acquired from the hidden Web (cf. Section 1.4.1).

2. The syntax of these services is analyzed. This also implies a mapping between inputs and outputs of services on one hand, and concepts of the domain of interest on the other hand (cf. Section 1.4.2).

3. The semantics of relations between inputs and outputs of hidden-Web services is then investigated (cf. Section 1.4.3).

4. Once the semantics of a service is fully understood, it is indexed. This index serves to answer high-level queries that are asked directly in the language of the domain ontology (cf. Section 1.4.4).



WWW
discovery
discovery
Web service
HTML form
probing
Analyzed form + result pages
schema matching
wrapper induction
Web service + input/output mapping
semantic analysis
Analyzed Web service
indexing
Service index
query
results
User
Service acquisition (Section 1.4.1)
Syntax analysis & Concept mapping (Section 1.4.2)
Semantic analysis (Section 1.4.3)
Indexing & Query processing (Section 1.4.4)

Figure 1.1: Process for understanding the hidden Web

We have implemented a system along these lines. Some of the components are still being developed. We shall briefly discuss how these components work together in the conclusion of this thesis.

The general process described in Figure 1.1 can be seen as the collaboration of independent agents such as crawlers or information extractors manipulating (inserting, modifying) information about the hidden Web in a common warehouse. The information obtained by most of these modules is imprecise. For instance, the probing module may determine that some field in a form represents a person's first name, with some confidence level. Also, the provenance of the information is often important, for example to explain how a particular piece of data has been obtained. Finally, note that the process is shown in Figure 1.1 as rather sequential for presentation purposes, whereas, in reality, it is not. For instance, information obtained by natural language processing techniques during semantic analysis may turn useful to improve the quality of wrappers induced in a previous step.

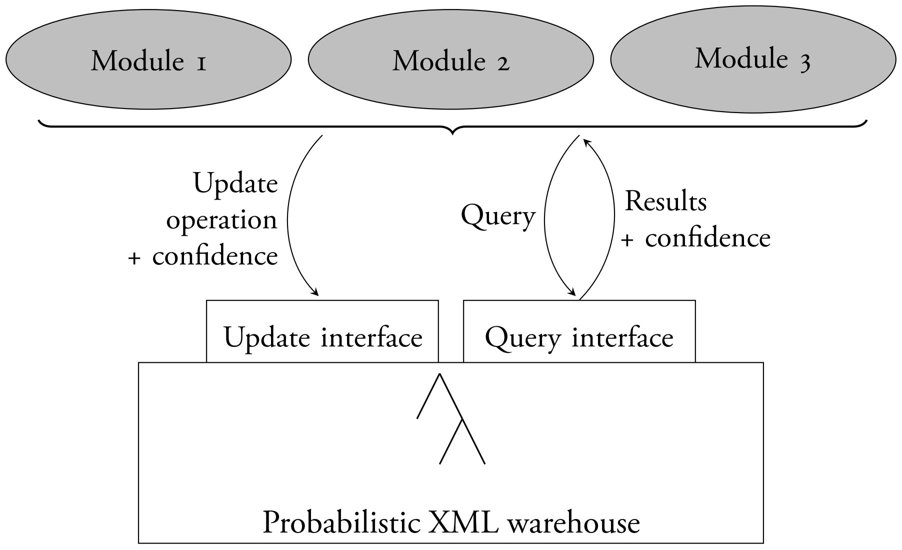


Figure 1.2: Warehouse of imprecise data

One could use a "workflow" approach to manage the collaboration of the various agents. Since their numbers and their roles are not fixed and their sequencing quite complex, we prefer to use a *content-centric* system, as shown on Figure 1.2. So we follow an approach in the style of [ANR05]. More precisely, the system is built on top of a content (semi-structured) warehouse, with querying and updating capabilities supporting imprecise information and provenance. Probabilistic information is stored in the warehouse and confidence tracking is directly provided through the query and update interfaces. Each query result and update operation comes with confidence information.

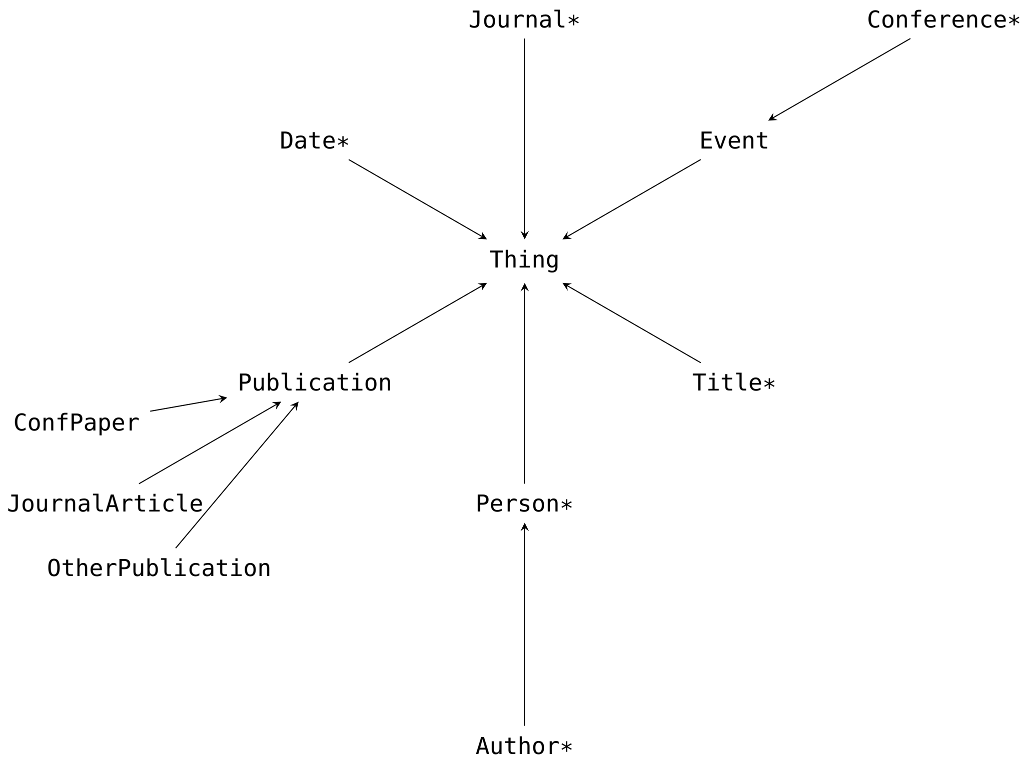
The model supporting this imprecise semi-structured warehouse, *prob-trees*, is described in Chapter 2. It is based on the use of probabilistic event variables associated to nodes of the data tree. The prob-tree model is *complete* (all combinations of possible worlds can be represented as a prob-tree), *concise* (for instance, the size of the representation grows linearly when imprecise nodes are inserted) and supports a powerful query language (tree-pattern queries with join), as well as arbitrary sequences of insertions and deletions.

1.3 Domain Knowledge

We present in this section the domain knowledge that we need to describe a specific domain of interest. We stress that all techniques described in the thesis, although they are illustrated with the example of the research publication domain, can be applied on any domain, provided that the necessary knowledge described here is given.

The needed domain knowledge can be decomposed into two different parts: a *domain ontology* and *domain instances*.

Domain ontology. The domain ontology is formed of a set of *concept names*, organized in a directed acyclic graph (DAG) of *IsA* relations, and a set of *relation names*, along with their arity and type (the type of an n-ary relation R is an n-ary tuple of concept names). More formal definitions are provided in Chapter 6. A subset of the set of concept names is distinguished as the set of *concrete* concepts (concrete concepts are the ones for which *instances* exist, cf. the following).



```
HasTitle          (Paper         , Title     )
PublishedIn       (Paper         , Date      )
WrittenBy         (Paper         , Author    )
PublishedInJournal(JournalArticle, Journal   )
PublishedInConf   (ConfPaper     , Conference)
```

Figure 1.3: Simple ontology for the research publication domain

To illustrate, consider our ontology for the research publication domain given in Figure 1.3.

IsA relations can be read as follows: a ConfPaper is a Publication, that is a Thing. Neither of these are concrete concepts, contrarily to Title (asterisks denote concrete concepts). Besides, the relation name HasTitle defines a binary relation between an instance of the Paper concept (or of one of its subsumed concepts) and an instance of the Title concept.

Such ontologies are simple examples of ontologies that are for instance used in the Semantic Web. They are typically expressed in standard languages such as RDF Schema [W3C04a] or OWL-Lite [W3C04b]*.

Domain instances. We deal here with concrete representations (as strings) of concrete concepts. Observe for instance that the strings *June 2007* and *07/06* may both stand for the same instance of a Date concept. Domain instances are character strings that stand for instances of the concrete domain concepts. More specifically, for each concrete concept c, we need the following:

- words used in the concept instances, with an approximate frequency distribution;
- an approximative probabilistic model of instances of the concept, that is, a way to assign some probability that a given string represents an instance of c.

At the minimum, we need a list of strings that represent concept instances for each concept. The most basic way to get these two pieces of knowledge is then to do the following:

- take a representative set of words appearing in these strings, with their corresponding frequency, as the approximate frequency distribution;
- for each given string s, if s may stand for an instance of concepts $c_1 \dots c_n$, assign $1/n$ as the probability that s stands for an instance of each concept c_i.

Let us insist on the fact that we only need approximations of these frequency distribution and probabilistic model. For some concepts, we may provide a more elaborate description (see below for concepts of our example ontology), but it is not expected, in any case, to provide a perfect description of concept instances. We discuss in Chapter 3 how to use word frequencies to understand the structure of an HTML form, and in Chapter 4 how to use the probabilistic model to extract information from result pages to a form.

Let us describe the domain instances that we use in the case of the research publication domain. We downloaded the content of the DBLP [Ley] database, as an XML file, from http://dblp.uni-trier.de/xml/ and we used it to generate our domain instances:

- For the concepts Title, Journal, Conference, we used the basic technique described above.
- For the Date concept, we provide a specific entity recognizer (in the form of a set of regular expressions describing monthly or yearly dates).
- For the Author concept, we extracted first and last names from DBLP person names with some heuristics, and use regular expressions describing the ways to recombine first names and last names (forming this way *Abiteboul*, *Pierre Abiteboul*, *Abiteboul Pierre*, *Abiteboul Pierre Paul* or *Abiteboul, Pierre* with various probabilities, from the last name *Abiteboul* and the first names *Pierre* and *Paul*).

*Both RDF Schema and OWL only consider binary relations, whereas we deal with relations of arbitrary arity, though binary relations are the most commonly occurring ones. Note that n-ary relations can be encoded into binary relations using *reification* if need be.

For the last two cases, probabilities associated with a regular expression are chosen in a quite *ad hoc* way. Ideally, they should come from statistical evaluation on large corpora.

This form of knowledge (domain ontologies, with a concept DAG and typed relations, and domain instances, with a frequency distribution of words and a probabilistic model for each concrete concept) is all that is used by the different modules described in this thesis. It is possible, though, that some steps of the process that we do not provide a system for (especially, service acquisition and semantic analysis) require additional knowledge (for instance, keywords related to the domain of interest or linguistic description of relation names).

1.4 Modules of our Framework

We describe here the different modules shown in Figure 1.1. Some of them are the topic of separate chapters.

1.4.1 Service Acquisition

The first phase of the process is to acquire information. The acquisition of information comes first from publication by users. The system also acquires information using search engines or by crawling the Web. The system is primarily interested in:

- HTML forms [W3C99]; these are extremely used and millions may be found on the Web.
- Web services [W3Ca]; one finds them typically in UDDI registry entries. Their description is usually given in WSDL [W3C01]. More and more are found on the Web.

The system is also interested in *extensional* resources, such as XML and HTML documents containing useful information. Because the focus is on the hidden Web, extensional information is not of primary concern here. Still, it should be possible to import such sources into the system. Furthermore, as we shall see in Chapter 6, documents are useful for *bootstrapping* the use of services.

It is important to note that we are interested in services providing information such as the DBLP Web site and not concerned with services with side effects such as booking services or mailing-list management interfaces. In particular, the Internet Standard on HTTP 1.1 [IET99] states that the HTTP GET method *should* be side-effect free, and this rule is in general followed. So forms with the GET method are typically acceptable. On the other hand, HTML forms using the POST method are often used for side effects. Note, however, that this is not always the case. Therefore, when we crawl the Web to discover services, we have to detect services with side effects to exclude them and it is not easy to do so. We shall have to rely on heuristics such as avoiding services requiring the input of an email-address, a credit-card number, or a password (unless this service has been explicitly declared by a user of the system).

As already mentioned, we are only interested in services relevant to some domain of interest. If, for instance, we crawl the Web to discover services, we must “focus” the crawl to the domain of interest; this kind of *focused crawling* of the World Wide Web has been studied in [CvdBD99, DCL+00]. An interesting approach to the specific problem of focused crawling for discovering forms is presented in [BF05]. The authors combine classical focused-crawling techniques, with a page classifier for the domain of interest, with two sources of feedback that help controlling it: a form classifier that checks whether a form is of interest, and a link classifier that considers features of the history of the links followed to get to the current page (the assumption is that there are distinguishing features in the path followed from, say, a Web site root page, to a database search form). The authors show that this allows to discover hidden-Web databases with less crawling resources than either full crawls or generic focused-crawling techniques. Note, however, that all three classifiers must be trained with annotated examples of pages, forms, and link paths.

Services of interest may also be obtained by querying general search engines with domain keywords. For instance, the queries "publication database" or "publications advanced search" already return services highly relevant to the research publication domain. Directories of online databases may also be of use, though we already noted that their coverage is quite low.

The acquisition of sources of information relevant to a domain of interest from the hidden Web is not one of the step of the process that we specifically worked on. So the topic will not be detailed in this thesis.

1.4.2 Syntax Analysis and Concept Mapping

Once services are identified, their structure have to be understood, in order to be able to effectively use them. By *structure*, we mean the kinds of inputs and outputs expected and produced by each service, the way to provide the service with these inputs, as well as the way to retrieve the outputs of a service. A related issue is that of mapping inputs and outputs to the concepts of our ontology. Indeed, as we shall see, we need to perform both steps at the same time in the case of HTML forms.

Let us first consider Web services, where the problem is easier. A WSDL description of a Web service is a formal description of the kinds of inputs expected by the service, and the outputs it returns. The names of inputs and outputs, however, are arbitrary, and may not have direct relations with the domain concepts. Mapping concepts to these names is an example of a *schema matching* problem; we do not bring any direct contribution to this topic, and we refer the reader to [RB01] for a survey of automatic schema matching techniques. Another source of information is the concrete (XML Schema) types of the inputs: a "`\(\d{3}\) \d{3}-\d{4}`" regular expression probably represents a telephone number, whereas an `xs:date` simple type represents a date.

Consider next HTML forms. Here, the structure of both the form itself and of result pages (which have to be generated by *probing* the form, i.e., submitting it with some filled-in values) must be understood. Note that it is thus necessary, in order to obtain the result pages, to have some understanding of what concepts the fields of the form map to. In other words, structural analysis and concept mapping must be performed at the same time. We present a two-step process for solving this problem:

1. Chapter 3 shows how to use some heuristics to map concepts of the domain ontology to the fields of a form, and to confirm these annotations by probing these fields.

2. Chapter 4 presents an approach to the extraction of information from result pages; a *gazetteer* annotates the page with instances of the domain concepts, and this imprecise and incomplete annotation is generalized as the input of a machine-learning–based wrapper-induction system.

1.4.3 Semantic Analysis

A service, with its syntactic description, must be semantically analyzed in order to understand the semantic relations between its inputs and outputs; the aim is here is to express the semantics of the service in a formal way.

We present in Chapter 6 a semantic model for describing services. The way to get this semantic description from services, however, is quite complex. In some domains, there are few ambiguities:

for instance, in the domain of research publications, if we have the concepts `Title`, `Person`, `Journal` and `Date`, it is most likely that the title `Title` is that of an article written by the person `Person` in a journal `Journal` published at the date `Date`. This is not necessary however: the date could be the date of last change of the publication, and the person could be the editor of the journal. Other domains may cause even more elaborate issues. Consider for instance a genealogy database. Here, it is very important to know if the relation `FatherOf` is between P and P' or between P' and P. Tools at our disposal to solve these issues are basic natural language processing techniques or keyword selection in the description of the service, for instance in the form page itself or in pages pointing to this page.

Our contribution to this problem lies in the theoretical framework described in Chapter 5. We present there how the notion of *optimal* relationship between database instances can be formalized. This part of our work did not lead to any direct implementation.

1.4.4 Indexing and Query Processing

Assuming that we have a set of services that are semantically described, the final step of our process is to provide a user with the possibility of using these services to answer high-level queries, in the language of the domain ontology. This means indexing the services and translating queries over the domain ontology to queries over relevant services. This problem is highly dependent on the representation of the semantics of a service, and the discussion will be therefore deferred to Chapter 6, where we present the semantic model.

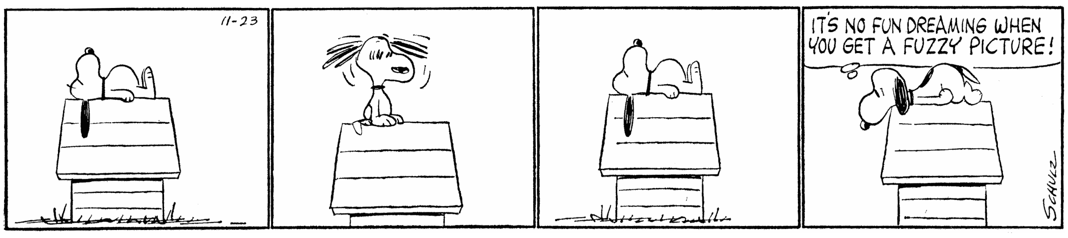
Conclusion

In this chapter, we presented a general and fully automatic process for the semantic interpretation of the hidden Web. This process is based on independent agents carrying out different tasks such as information acquisition and enrichment, and participating to the construction of a content-centric semi-structured probabilistic warehouse. We briefly described the different steps of the process. For some of them, only prospectives ideas are given, as we do not claim a complete solution to the problem of exploiting the information of the hidden Web.

The following chapters will further detail some components, starting with the underlying probabilistic data model of the content warehouse.

Chapter 2

A Probabilistic XML Database Model



This chapter takes up and expands on two previously published papers, one of which is system-oriented [5], and the other is a complexity study [4]. The presentation and proofs are more detailed here, and some content is new.

A large number of automatic tasks on real-world data generate *imprecise* results, e.g., information extraction, natural language processing, data mining. Moreover, in many of these tasks, information is represented in a semi-structured way, either due to an inherent tree-like structure of the original data, or because it is natural to represent derived information or knowledge in a hierarchical manner. This is in particular the case in our process for understanding the hidden Web, as argued in Section 1.2. We need to manage imprecise tree information gathered by a system during its entire lifetime, and in particular evaluate queries and imprecise updates over such data. We present here an original probabilistic tree model for managing imprecise tree data, the *prob-tree* model[*]. We discuss in detail expressiveness results, efficient ways to perform queries and updates, and a number of theoretical and implementation issues.

The purpose is to design a model for storing imprecise information, that is both *expressive* and *concise*. Prob-trees are unordered trees whose nodes are annotated by conjunctions of (possibly negated) *event variables*, in the style of conditions in [IL84]. Each event variable is assigned a probability value. In particular, every probabilistic update introduces a new event variable (independent from the previous ones) that captures the belief the system has in this particular update. The semantics of such a prob-tree is described in terms of sets of *possible worlds*. We identify a large class of queries for which efficient querying and updating algorithms, directly performed over the prob-tree, compute the correct answer, by using evaluation algorithms developed for precise data. A large class of queries can be evaluated in **PTIME**. For updates, deletion may be intractable. (Observe that in settings we are interested in, based on tools gathering knowledge, deletions are rare.) We also propose a theoretical foundation for the prob-tree model. In particular, we discuss the notion of equivalence of prob-trees and its complexity. We also study the issue of removing less probable possible worlds, and that of validating a prob-tree against a DTD. We show that these

[*]In [5], we referred to it as the *fuzzy tree* model; we changed the terminology in subsequent works, in order to avoid confusion with works on *fuzzy databases*.

two problems are intractable in the most general case. We discuss variants of the prob-tree model, and how it compares to other models for representing semi-structured probabilistic information. We finally describe the FuzzyXML system, an implementation of the prob-tree model relying on generic query and update processors.

This chapter is organized as follows. We first present related work on probabilistic databases in Section 2.1. Section 2.2 introduces the prob-tree model, while Section 2.3 discusses the querying and updating of prob-trees. In Section 2.4, a notion of equivalence between prob-trees is introduced, and its complexity is investigated. Other complexity issues are considered in Section 2.5. Variants of the prob-tree model are discussed in Section 2.6 and other probabilistic models are compared to the prob-tree model in Section 2.7. Finally, we describe our implementation in Section 2.8.

2.1 Related Work

The topic of probabilistic databases has been intensively studied, see for instance [dR95, CP87, BGMP92, FR97], and [DS04, Wid05] for more recent works. A recent tutorial surveying a number of probabilistic database models can be found in [DS07]. The idea of associating probabilistic formulas to data elements comes from the *conditional tables* of [IL84]. Conditional tables are used for representing incomplete information, but their use for probabilistic databases is quite straightforward, as discussed in [GT06]. A work close in spirit to this one, but in the context of relational databases, is [AKG91]. The tree structure and multi-set semantics we use have for consequence that the complexity results on tables of [AKG91] do not apply to our model.

A relatively small number of works have dealt with the representation of probabilistic semi-structured data. In [DGH01], a semi-structured database is used to store complex probability distributions of data that is essentially relational. Works closer to ours are [NJ02, HGS03, vKdKA05, LSC06]. Nierman et al. [NJ02] describe ProTDB, a model based on the use of *mutually exclusive* and *independent* probability distributions assigned to virtual nodes in a tree. We present in more detail this model in Section 2.7.2, and show how it can be seen as a particular case of the prob-tree model, following observations from [KS07]; this latter work discusses advanced querying issues (projection and boolean queries, incomplete answers) in the ProTDB setting. In [HGS03], a complex model, based on directed acyclic graphs, is developed, along with an algebraic query language. Keulen et al. [vKdKA05] present an approach to data integration using probabilistic trees; their model is based on possible-worlds semantics, and allows both extensive descriptions of the possible worlds and node-based factorization. Querying and the way to present data integration results on this model are also shown. Finally, Li et al. observe in [LSC06] that query efficiency can be improved on probabilistic trees when *absolute* probabilities are stored on nodes of the trees, instead of conditional probabilities, at the cost of more expensive changes in the probability values. None of those works touch upon the question of updates, which is of major interest to us in this chapter, since the prob-tree model is intended to be used in a probabilistic data warehouse that different modules query and update.

2.2 The Prob-Tree Model

In this section, we present the basics of the prob-tree model. We first introduce a tree data model and, next, the prob-tree model, along with its possible-worlds semantics.

2.2.1 Data Model

We assume some countably infinite sets of *node identifiers* $\mathcal{N}$ (distinct nodes of a tree have distinct identifiers), *labels* $\mathcal{L}$ (say, the set of character strings), *values* $\mathcal{V}$ (say, the set of character strings, with ε as the empty string). In the following, nodes will be denoted n, n', s, s'...; labels A, B, C, D...; and a `proportional` font will be used for values.

Definition 2.1. A *data tree t* is a 5-tuple $t=(N,E,r,\varphi,\nu)$ where $N\subset\mathcal{N}$ is a finite set of nodes, $E\subseteq N^2$ a tree rooted in $r\in N$, $\varphi:N\to\mathcal{L}$ associates a label to each node in N and ν associates a value in $\mathcal{V}$ to each leaf of t.

Let $t=(N,E,r,\varphi,\nu)$ and $t'=(N',E',r',\varphi',\nu')$ be two data trees. We say that t and t' are *isomorphic* (denoted $t\sim t'$) if there is a bijection $\psi:N\to N'$ such that:

(i) For $s_1,s_2\in N$, $(s_1,s_2)\in E\Leftrightarrow(\psi(s_1),\psi(s_2))\in E'$;
(ii) $\psi(r)=r'$;
(iii) $\forall s\in N$, $\varphi'(\psi(s))=\varphi(s)$;
(iv) $\forall s\in N$, if s is a leaf then $\nu'(\psi(s))=\nu(s)$;

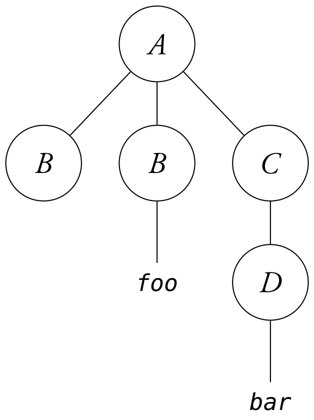


Figure 2.1: Example data tree

An example of data tree is given in Figure 2.1. Observe that ε values are omitted in the figure. Indeed, we shall often consider trees without data values. The simple data model we use is inspired by XML but ignores a number of XML features such as the ordering, the distinction between attributes and elements, or mixed content. Some of them could be considered as “cosmetic” additions to the model; one is critical, namely the ordering of nodes, as briefly touched upon in Section 2.6.4. It should also be observed that this data model adopts a multi-set semantics. To see that, consider for instance a data tree with a root node and two children with the same label. We see it essentially as different from a data tree with a root node and a single child with the same label. A model based on a pure set semantics is briefly considered in Section 2.6.1.

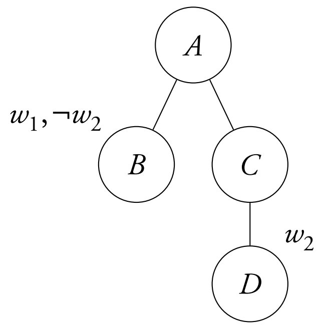
2.2.2 Syntax of Prob-Trees

We next present the *prob-tree model* for representing probabilistic semi-structured information, that is based on annotating the nodes of a tree with probabilistic conditions in the style of the conditions in [IL84].

We assume the existence of a countable set $\mathcal{W}$ of *event variables*. Let W be a finite set of event variables. A *condition* over W is a (possibly empty) set of *atomic conditions* of the form w or $\neg w$ (for

w in W). This condition is interpreted as the conjunction of the atomic conditions. A *probability distribution* π for W assigns probabilities, i.e., values in $]0;1]$, to the different event variables in W. We choose not to allow zero probabilities so that, in particular, updates with a zero probability will not be performed at all. But this is only a convention and could be changed without altering the results presented here. Formally, we have:

Definition 2.2. A *prob-tree* (short for *probabilistic tree*) T is a 4-tuple (t, W, π, γ) where $t = (N, E, r, \varphi, \nu)$ is a data tree, $W \subseteq \mathcal{W}$ is finite, π is some probability distribution over W, and γ assigns conditions over W to nodes in $N - \{r\}$.



Event	Proba.
w_1	0.8
w_2	0.7

Figure 2.2: Example prob-tree

An example of prob-tree is shown in Figure 2.2. The node labeled by B is annotated with $\{w_1, \neg w_2\}$, the D node with $\{w_2\}$, and the C node with the empty condition. We now define the *semantics* of a prob-tree, introducing to do that the notion of *possible-worlds set*.

2.2.3 Semantics of Prob-Trees

The real world with some uncertainty is modeled by a set of possible worlds, each with its own probability. More precisely, a *possible-worlds (PW) set* S is a finite set of pairs (t_i, p_i) where (i) the t_i are data trees with the same root label, and (ii) each p_i is a positive real number with $\sum_{i=1}^{n} p_i = 1$. An example of a PW set is shown in Figure 2.3.

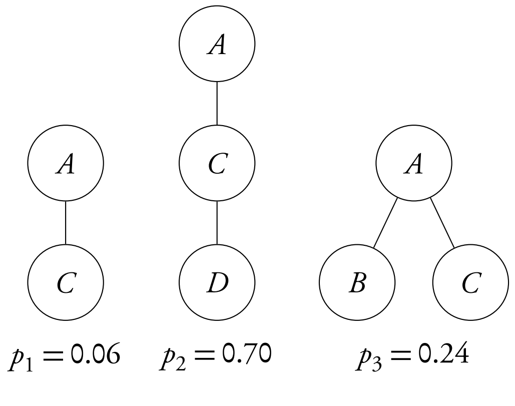


Figure 2.3: Example of possible-worlds set

As different PW sets may represent the same abstract possible worlds, we need the notion of *isomorphism* between possible-worlds sets. Let $S = \{(t_1, p_1) \dots (t_n, p_n)\}$ and $S' = \left\{(t'_1, p'_1) \dots (t'_m, p'_m)\right\}$

be two possible-worlds sets. We say that S and S' are *isomorphic* (denoted $S \sim S'$) if, for each data tree t appearing either in S or in S':

$$\sum_{\substack{1 \leqslant i \leqslant n \\ t_i \sim t}} p_i = \sum_{\substack{1 \leqslant j \leqslant m \\ t'_j \sim t}} p'_j.$$

This allows defining the notion of *normalization* of PW sets: a PW is *normalized* if it does not contain two possible worlds with isomorphic data trees. Every PW set can be normalized by assigning as the probability of each possible world the sum of the probabilities of possible worlds with isomorphic data trees.

Note that the possible-worlds model could be used itself as a model for representing probabilistic trees, but this is typically verbose, so unpractical.

We are now ready to define the semantics of prob-trees in terms of possible worlds:

Definition 2.3. Let $T = (t, W, \pi, \gamma)$ be a prob-tree. For $V \subseteq W$, the *value of T in the world V*, denoted $V(T)$, is the subtree of t where all nodes conditioned by a '$\neg w$' atom for $w \in V$ or by a 'w' atom for $w \notin V$ have been removed (as well as their descendants). The *possible-worlds semantics* of T, denoted $[\![T]\!]$, is the PW set defined by:

$$[\![T]\!] = \bigcup_{V \subseteq W} \left\{ \left(V(T), \prod_{w \in V} \pi(w) \prod_{w \in W - V} (1 - \pi(w)) \right) \right\}.$$

Observe that an underlying assumption of this particular semantics is that the probabilistic events of a prob-tree are supposed to be independent. As an example, the PW set shown in Figure 2.3 is (up to isomorphism) the semantics of the prob-tree of Figure 2.2. An important result is that the prob-tree model has the same expressive power as the possible-worlds model:

Theorem 2.4. *For each PW set S, there exists a prob-tree T such that $S \sim [\![T]\!]$.*

Proof. Let $S = \{(t_1,p_1)\ldots(t_n,p_n)\}$ be a PW set. Assume without loss of generality that all t_i use different nodes, except for the root r that is common to all of them. Let $w_1 \ldots w_{n-1}$ be $n-1$ event variables. We define the prob-tree $T = \left(t, \left\{w_1 \ldots w_{n-1}\right\}, \pi, \gamma\right)$ as follows:

- the root of t is the common root r of all t_i;
- $\forall i,\ \pi(w_i) = \dfrac{p_i}{1 - \sum_{1 \leqslant j < i} p_j}$;
- for each i, all children of the root of t_i are added as children of the root of t, along with their descendant subtrees, conditioned by:

$$\begin{cases} w_1 & \text{if } i = 1, \\ \neg w_1, \ldots, \neg w_{i-1}, w_i & \text{if } 2 \leqslant i \leqslant n-1, \\ \neg w_1, \ldots, \neg w_{n-1} & \text{if } i = n. \end{cases}$$

We have:

$$\begin{aligned}
[\![T]\!] &= \bigcup_{V\subseteq\{w_1\dots w_{n-1}\}} \left\{\left(V(T), \prod_{w\in V}\pi(w)\prod_{w\in W-V}(1-\pi(w))\right)\right\}\\
&\sim \left\{\left(t_1, \sum_{\substack{V\subseteq\{w_1\dots w_{n-1}\}\\ w_1\in V}}\prod_{w\in V}\pi(w)\prod_{w\in W-V}(1-\pi(w))\right)\right\}\\
&\quad\cup\left\{\left(t_2, \sum_{\substack{V\subseteq\{w_1\dots w_{n-1}\}\\ w_1\notin V\wedge w_2\in V}}\prod_{w\in V}\pi(w)\prod_{w\in W-V}(1-\pi(w))\right)\right\}\\
&\quad\cup\dots\\
&\quad\cup\left\{\left(t_n, \sum_{\substack{V\subseteq\{w_1\dots w_{n-1}\}\\ w_1\notin V\wedge\dots\wedge w_{n-1}\notin V}}\prod_{w\in V}\pi(w)\prod_{w\in W-V}(1-\pi(w))\right)\right\}\\
&= \{(t_1,\pi(w_1))\}\cup\{(t_2,(1-\pi(w_1))\pi(w_2))\}\cup\dots\cup\\
&\quad\{(t_n,(1-\pi(w_1))\dots(1-\pi(w_{n-1})))\}\\
&= \{(t_1,p_1)\}\cup\{(t_2,p_2)\}\cup\dots\cup\{(t_n,p_n)\}\\
&= S
\end{aligned}$$

since

$$\prod_{j=1}^{i}(1-\pi(w_j)) = \prod_{j=1}^{i}\frac{1-\sum_{1\leqslant k<j+1}p_k}{1-\sum_{1\leqslant k<j}p_j} = 1-\sum_{1\leqslant k<i}p_i. \qquad \square$$

Note that this construction uses a number of event variables linear in the number of possible worlds in S. Thus, the size of the resulting prob-tree is essentially the size of the original PW set. One could clearly hope to find more compact representations.

In order to guarantee conciseness of the prob-tree model, we may want to have a polynomial bound on the size of prob-trees whose semantics only involve data trees of bounded size (and with probabilities of bounded precision). A model with such a polynomial bound, the so-called *simple probabilistic* model, is presented in Section 2.7.1, but we shall see that it is less expressive than the PW model. Actually, the following result shows that neither the prob-tree model, nor any other model as expressive as the PW model, can guarantee such a bound:

Proposition 2.5. *Let μ be a one-to-one mapping sending every normalized possible world set (with probabilities of bounded precision) to some integer (say, a binary representation of an element of a model). Then, the average size of $\mu(S)$ (that is, $\log\mu(S)$) for PW sets S in which every possible world has at most n nodes is at least exponential in n.*

Proof. This results from a simple counting of the number of possible-worlds sets involving only possible worlds with at most n nodes. Let us call this number σ_n. If we forget about the values of the probabilities and the labels of the nodes, we get that σ_n must be greater than the number of sets of unordered, unlabeled, rooted trees with at most n nodes. We have the following equality about the number a_n of unordered, unlabeled, rooted trees with exactly n nodes [Ott48, Knu97]:

$$a_n = \frac{\alpha^{n-1}}{n}\sqrt{\beta/2\pi n} + O(n^{-5/2}\alpha^n)$$

where $\alpha > 2$ and β are two constants. We have therefore:

$$\sigma_n \geqslant 2^{\sum_{i=1}^{n} a_n} \geqslant 2^{a_n} = \Omega(2^{2^n}).$$

Since σ_n is doubly exponential in n, an element of $\mu(S)$ cannot be identified on average with less than $\Omega(2^n)$ bits. □

2.3 Queries and Updates

We next consider how to perform queries and updates directly on prob-trees.

2.3.1 Querying Prob-Trees

We first precisely define the queries we consider. The goal is to be able to evaluate efficiently queries. Indeed, in practice, one would ideally like to rely on a standard query processor to do most of the work. We show that we can use this approach for a very large class of queries, namely *locally monotone* queries. To define this class, we use the auxiliary notion of sub-datatree. Note that we only consider subtrees that have the same root as the original trees, obtained by pruning some of its branches.

Definition 2.6. Let $t = (N,E,r,\varphi,\nu)$, $t' = (N',E',r',\varphi',\nu')$ be two data trees. Then t' is a *sub-datatree* of t (denoted $t' \leqslant t$) if: (i) $N' \subseteq N$; (ii) if $n_1 \in N'$ and $(n_2,n_1) \in E$, $n_2 \in N'$; (iii) $E' = E \cap N'^2$; (iv) $r' = r$; (v) $\varphi' = \varphi_{|N'}$; (vi) $\nu' = \nu_{|N'}$. The set of all sub-datatrees of a data tree t is denoted $\mathsf{Sub}(t)$.

The sub-datatree relation is clearly a partial order, which justifies the notation $t' \leqslant t$.

The queries we consider return subtrees of the data tree. This greatly simplifies the management of probabilities. Intuitively, we return pieces of the original tree, but always keep the path from these pieces to the root. This notion is defined formally next, together with the large class of queries for which we are able to obtain an efficient query evaluation algorithm.

Definition 2.7. A *query* Q is a function over the set of data trees, such that for each data tree t, $Q(t)$ is a (possibly empty) set of sub-datatrees of t. The tree t is said to be *matched* by Q if $Q(t) \neq \varnothing$.

A query Q is *locally monotone* if either of the following two equivalent conditions holds:

(i) for any three data trees $u \leqslant t' \leqslant t$, $u \in Q(t) \iff u \in Q(t')$;

(ii) for any two data trees $t' \leqslant t$, $Q(t') = Q(t) \cap \mathsf{Sub}(t')$.

Proof of the equivalence.

(i) ⇒ (ii). Let t and t' be two data trees such that $t' \leqslant t$. Let $u \in Q(t')$. By definition of queries, $u \in \mathsf{Sub}(t')$, which means that $u \leqslant t'$. By (i), $u \in Q(t)$. This shows that $Q(t') \subseteq Q(t) \cap \mathsf{Sub}(t')$.

Let now $u \in Q(t) \cap \mathsf{Sub}(t')$. By (i), $u \in Q(t')$. This shows that $Q(t') \supseteq Q(t) \cap \mathsf{Sub}(t')$ and concludes the proof of the implication.

(ii) ⇒ (i). Let $u \leqslant t' \leqslant t$ three data trees. Suppose first that $u \in Q(t)$. As $u \in \mathsf{Sub}(t')$, by (ii), $u \in Q(t')$.

Suppose now that $u \in Q(t')$. By (ii), $u \in Q(t)$. □

Locally monotone queries is a class of queries for which, *given an algorithm to compute the query on data trees*, we can compute easily the answers on a prob-tree. To illustrate, let us consider a particular class of queries, *tree-pattern queries with joins*, that we show to be locally monotone.

Definition 2.8. A *tree-pattern query with joins* (referred to in the following as a *TPWJ query*) is defined as a 4-uple (t, D, J, J') where $t = (N, E, r, \varphi, \nu)$ is a data tree, $D \subseteq E$ is a set of *descendant edges* (the other edges are interpreted as *child edges*) and $J \subseteq N^2$ (respectively, $J' \subseteq N^2$) is a set of *positive* (respectively, *negative*) *join conditions*, such that, for all (n, n') in J, n and n' are two leafs of t and $n \neq n'$.

Let $Q = (t, D, J)$ with $t = (N, E, r, \varphi, \nu)$ be a TPWJ query and $t' = (N', E', r', \varphi', \nu')$ a data tree. Then a *valuation* μ (from Q in t') is a mapping from N to N' verifying:

(i) $\mu(r) = r'$;
(ii) $\forall n \in N$, either $\varphi(n)$ is the special symbol '$*$', or $\varphi(\mu(n)) = \varphi(n)$;
(iii) $\forall (n_1, n_2) \in E$, if $(n_1, n_2) \in D$, $\mu(n_2)$ is a descendant of $\mu(n_1)$, otherwise it is a child of $\mu(n_1)$;
(iv) for each leaf n of t with $\nu(n) \neq \varepsilon$, $\mu(n)$ is a leaf of t' and $\nu'(\mu(n)) = \nu(n)$;
(v) for each $(n_1, n_2) \in J$ (respectively, $(n_1, n_2) \in J'$), both $\mu(n_1)$ and $\mu(n_2)$ are leaves of t' and $\nu'(\mu(n_1)) = \nu'(\mu(n_2))$ (respectively, $\nu'(\mu(n_1)) \neq \nu'(\mu(n_2))$).

The set of all answers to $Q(t')$ is the set of minimal sub-datatrees of t' containing $\mu(N)$ for every possible valuation μ.

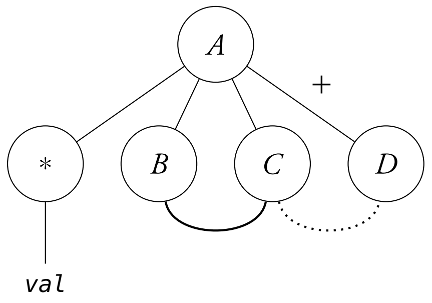


Figure 2.4: Example of tree-pattern query with joins

Such a TPWJ query can be represented as a graph as in Figure 2.4. Note the positive join condition between the nodes labeled B and C, and the negative join condition between the nodes labeled C and D (these are *value joins*). Intuitively, this query will be matched by a tree having a child of its root A with any label (with constant value `val`), children B and C (with identical values) and a descendant labeled D, with a different value from that of B and C nodes. A more concrete example of a TPWJ query is given in Section 2.8. We now show that this class of queries is indeed locally monotone.

Proposition 2.9. *TPWJ queries are locally monotone.*

Proof. We shall prove (ii) of Definition 2.7. Let $Q = (t_Q, D, J, J')$ be a TPWJ query with $t_Q = (N_Q, E_Q, r_Q, \varphi_Q, \nu_Q)$. Let $t = (N, E, r, \varphi, \nu)$ and $t' = (N', E', r', \varphi', \nu')$ be two data trees with $t' \leqslant t$.

- Let $u \in Q(t')$. By definition, $u \in \mathsf{Sub}(t')$. Let us show that $u \in Q(t)$. There is a valuation $\mu : N_Q \to N'$ such that u is the minimal sub-datatree of t' containing $\mu(N_Q)$. Just observe that μ can also be seen as a valuation from Q in t (every condition of what defines a valuation remains valid), and u is still the minimal sub-datatree of t containing $\mu(N_Q)$. Therefore, $u \in Q(t)$.

- Let $u \in Q(t) \cap \mathsf{Sub}(t')$. There is a valuation $\mu : N_Q \to N$ such that u is the minimal sub-datatree of t containing $\mu(N_Q)$. Since u is a sub-datatree of t', $\mu(N_Q) \subseteq N'$. We conclude by observing that μ is still a valuation from Q in t (every condition of what defines a valuation remains valid), and u is the minimal sub-datatree of t' containing $\mu(N_Q)$. □

As we have just seen, TPWJ queries are locally monotone because the answer to a TPWJ query contains the full context needed to decide whether the query matches or not; in particular, TPWJ queries are *positive* queries (despite the negative joints), and do not imply any existential or universal quantification. It is easy to see that the query "Return the root if all its children are labeled by A" or even "Return the root if one of its children is labeled by A" are not locally monotone (in the former case, because the universal quantifier involves some form of negation; in the latter, because the full context of the match does not appear in the query result).

We now consider how queries are applied to PW sets and prob-trees.

Definition 2.10. Let Q be a query and $S = \{(t_i, p_i)\}$ a PW set. The *result* of Q on S, denoted $Q(S)$, is

$$\bigcup_{(t_i,p_i)\in S}\ \bigcup_{t\in Q(t_i)} \{(t,p_i)\}.$$

Observe that the answer to a query is not strictly speaking a possible-worlds set, since the probabilities do not have to sum to 1.

Definition 2.11. Let Q be a locally monotone query. The result of Q on a prob-tree $T = (t, W, \pi, \gamma)$, denoted $Q(T)$, is

$$\bigcup_{u\in Q(t)} \left\{ \left(u, \mathsf{eval}\left(\bigcup_{n \text{ node of } u} \gamma(n) \right) \right) \right\}$$

where $\mathsf{eval}(\mathit{cond})$ returns 0 if there is an event w such that both 'w' and '$\neg w$' are in cond, and is otherwise defined as:

$$\prod_{w\in \mathit{cond}} \pi(w) \prod_{\neg w\in \mathit{cond}} (1-\pi(w)).$$

In other words, the result of a locally monotone query on a prob-tree can be computed as described in Algorithm 2.1, given an algorithm to compute query results on data trees.

The following result states the consistence between the way queries are performed on prob-trees and the possible-worlds semantics.

Theorem 2.12. *Let T be a prob-tree and Q be a locally monotone query. Then, with a little abuse of notation since the probabilities in $Q(T)$ and $Q(\llbracket T \rrbracket)$ do not sum to 1, we have $Q(T) \sim Q(\llbracket T \rrbracket)$.*

```
Algorithm 2.1 Result of a locally monotone query on a prob-tree
INPUT: Locally monotone query $Q$, prob-tree $T=(t,W,\pi,\gamma)$.
OUTPUT: Set of pairs $\{(t_i,p_i)\}$ with $t_i$ a data tree and $p_i$ a probability.
(a) Compute the set $R$ of the results of $Q$ on $t$.
(b) For each $r\in R$, browse $r$ to gather all conditions $cond_r$ of nodes in $r$.
(c) Return $\{(r,\mathsf{eval}(cond_r))\mid r\in R\}$, with eval as in Definition 2.11.
```

Proof. Let $T=(t,W,\pi,\gamma)$, $Q(T)=\{(u_1,p_1)\ldots(u_n,p_n)\}$ and $Q(\llbracket T\rrbracket)=\{(u'_1,p'_1)\ldots(u'_m,p'_m)\}$. Let u be a data tree appearing either in $Q(T)$ or in $Q(\llbracket T\rrbracket)$. We denote:

$$p=\sum_{\substack{1\leqslant i\leqslant n\\ u_i\sim u}} p_i; \qquad\qquad p'=\sum_{\substack{1\leqslant j\leqslant m\\ u'_j\sim u}} p'_j.$$

We need to show that $p=p'$.

1. Let us first prove that $p\leqslant p'$. If $p=0$, this is trivial; suppose $p>0$. Then there is a data tree (or more than one) isomorph to u in $Q(T)$ (with a non-zero probability); let U be the set of all such data trees. For $v\in U$, let $cond_v$ be the union of all conditions annotating v in the prob-tree T. We have:

$$p=\sum_{v\in U}\mathsf{eval}(cond_v)=\sum_{v\in U}\prod_{w\in cond_v}\pi(w)\prod_{\neg w\in cond_v}(1-\pi(w)).$$

   For $v\in U$, let $\mathcal{V}_v$ be the subset of 2^W of all subsets of W that are *compatible* with $cond_v$, that is, such that if $V\in\mathcal{V}_v$, then all w with $w\in cond_v$ are in V, and all w with $\neg w\in cond_v$ are not in V. Then, for all $V\in\mathcal{V}_v$, we have $V(T)\leqslant t$ and $v\leqslant V(T)$. Since Q is locally monotone, $v\in Q(V(T))$. This yields:

$$\begin{aligned}p'&\geqslant\sum_{v\in U}\sum_{V\in\mathcal{V}_v}\prod_{w\in V}\pi(w)\prod_{w\notin V}(1-\pi-w)\\&=\sum_{v\in U}\prod_{w\in cond_v}\pi(w)\prod_{\neg w\in cond_v}(1-\pi(w))\underbrace{\sum_{V\in\mathcal{V}_v}\prod_{\substack{w\in V\\ w\notin cond_v}}\pi(w)\prod_{\substack{w\notin V\\ \neg w\notin cond_v}}(1-\pi(w))}_{=1}\\&=p.\end{aligned}$$

2. Now, let us prove that $p'\leqslant p$. If $p'=0$, this is trivial; suppose $p'>0$. Then there is a data tree (or more than one) isomorph to u in $Q(\llbracket T\rrbracket)$. Let U be the set of all such data trees. For $v\in U$, let $\mathcal{V}_v$ be the subset of 2^W of all subsets V of W such that $v\in Q(V(T))$. Observe that, since Q is locally monotone, $V\in\mathcal{V}_v$ if and only if $v\leqslant V(T)$. Let $cond_v$ be the union of all conditions annotating nodes of v in T. We have:

$$\begin{aligned}p'&=\sum_{v\in U}\sum_{V\in\mathcal{V}_v}\prod_{w\in V}\pi(w)\prod_{w\notin V}(1-\pi(w))\\&=\sum_{v\in U}\prod_{w\in cond_v}\pi(w)\prod_{\neg w\in cond_v}(1-\pi(w))\end{aligned}$$

using the same identity as above.

For $v \in U$ and $V \in \mathcal{V}_v$, we know that $V(T) \leqslant t$ and $v \in Q(V(T))$. Thanks to the local monotonicity of Q, $v \in Q(t)$. Then:

$$\begin{aligned} p &\geqslant \sum_{v \in U} \mathsf{eval}(cond_v) \\ &= \sum_{v \in U} \prod_{w \in cond_v} \pi(w) \prod_{\neg w \in cond_v} (1 - \pi(w)) \\ &= p'. \end{aligned}$$

□

2.3.2 Updating Prob-Trees

In this section, we present the class of updates we consider, their semantics, and show how to apply updates directly on prob-trees.

We assume that a query Q defines, for each data tree t, and for each $t' \in Q(t)$, a mapping $\mu^Q_{t'}$ from some finite set N_Q to the nodes of t' (they can be seen as nodes of t, since t' is a sub-datatree of t). For TPWJ queries, defined in the previous section, for instance, a natural mapping is the valuation from Definition 2.8: N_Q is the set of nodes of the query tree, and $\mu^Q_{t'}$ maps a node of the query tree to the corresponding node in t'. From now on, we assume that the query language defines, for each query, its associated mapping.

We define next update operations, whose basic components are *insertions* and *deletions*. Let $t = (N, E, r, \varphi, \nu)$ and $t' = (N', E', r', \varphi', \nu')$ be two data trees. Assume without loss of generality that they use different identifiers, i.e., $N \cap N' = \varnothing$. Let Q be a query, with its associated mapping. An *insertion* is an expression $i(n, t')$ where $n \in N_Q$ stands for the node where the data tree t' is to be inserted. A *deletion* is an expression $d(n)$ where $n \in N_Q$. The mapping of node n is removed as well as all its descendants. Insertions and deletions are *elementary* updates that are used to define *update operations*. Typically, one wants to perform a number of update operations based on the result of a query. This motivates the following definition.

Definition 2.13. An *update operation* is a pair $\tau = (Q, U)$ where Q is a locally monotone query and U is a non-empty set of:

1. *insertions* on N_Q, that is, expressions $i(n, t')$ where $n \in N_Q$ and t' is a tree to insert (as a child of the node mapped by n);
2. *deletions* on N_Q, that is, expressions $d(n)$ where $n \in N_Q$ (indicating the node to delete).

If $|U| = 1$, τ is called an *elementary* update operation.

Queries are used to select the nodes of the trees where insertions or deletions are made. Intuitively, when one applies an update operation (say, a single deletion) on a data tree t, it results in the deletion of a sub-datatree for each matching of Q. More formally:

Definition 2.14. Let $\tau = (Q, U)$ be an update operation. Let t be a tree matched by Q. For $o \in U$, let n_o be the node of N_Q appearing in o. The result of the operation τ on t, denoted $\tau(t)$, is the result of applying successively, for each $o \in U$:

1. if $o = i(n, t')$, the insertion, in t, of t' as a child of all $\mu^Q_u(n)$ for $u \in Q(t)$ (possibly inserting t' multiple times at the same place);

2. if $o = d(n)$, the deletion, in t, of $\mu_u^Q(n)$ for all $u \in Q(t)$.

Note that although the order of the operations above is not specified, there is no ambiguity here, if we chose the natural convention that a node inserted (or deleted) as a descendant of a node which is also deleted will not be present in the result tree.

A *probabilistic update operation* is a pair (τ, c) where τ is an update operation and $c \in]0;1]$ is the *confidence* we have in the operation. This confidence, defines, intuitively, the probability the update operation is carried out. This leads to the following definition for probabilistic updates on PW sets:

Definition 2.15. Let $S = \{(t_i,p_i)\}$ be a PW set, (τ, c) a probabilistic update operation, $\tau = (Q, U)$. The result of (τ, c) on S, denoted $(\tau, c)(S)$, is the PW set:

$$\begin{aligned} &\{(t,p) \in T \mid t \text{ is not matched by } Q\} \\ \cup &\{(\tau(t), p \cdot c) \mid t \text{ is matched by } Q\} \\ \cup &\{(t, p \cdot (1-c)) \mid t \text{ is matched by } Q\}. \end{aligned}$$

We can now define the same for prob-trees:

Definition 2.16. The result of a probabilistic update operation (τ, c) on a prob-tree T, denoted $(\tau, c)(T)$ is the result of applying Algorithm 2.2 to T and (τ, c).

This algorithm, in particular the deletion part, is quite complex. Note however, that in a number of common cases, some simplifications occur:

- If there are only insertions to perform, we just insert as many nodes as there are matchings, with appropriate conditions.
- If a node to be deleted is a leaf, the deep copy amounts to just the copy of the node and its value.

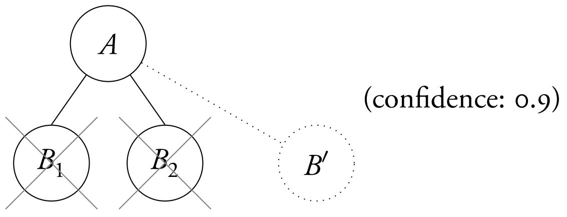


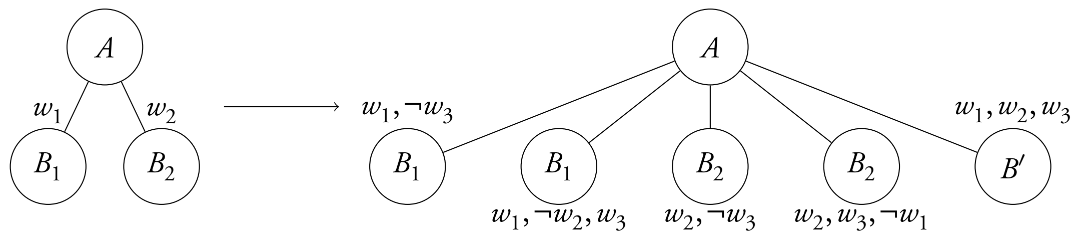
Figure 2.5: Example of probabilistic update

For a more elaborate case, let us consider the probabilistic update, described by a simple-tree pattern query, shown in Figure 2.5. In this graphical representation of an update, deletions appear as crossed-out nodes of the query tree, and insertions as subtrees added with dotted lines. This update can be expressed in natural language as follows: If there are two nodes labeled respectively by B_1 and B_2 that appear as children of the root, replace both with a single node labeled by B' with a confidence of 0.9. This probabilistic update is applied to an example prob-tree in Figure 2.6. Observe that, in the resulting tree, the update is not performed, and, hence, the node B_1 is not deleted, if the update hypothesis is not true $(w_1, \neg w_3)$ or if the update hypothesis is true but the node B_2 does not exist $(w_1, \neg w_2, w_3)$. A similar condition holds for the node B_2. This simple example shows that the information contained in a prob-tree can become quite complex, especially in the presence of deletions.

```
Algorithm 2.2 Result of a probabilistic update on a prob-tree
INPUT: A prob-tree $T=(t,W,\pi,\gamma)$, a probabilistic update operation $(\tau,c)$ with $\tau=(Q,U)$.
OUTPUT: $(\tau,c)(T)$ (by definition).
(a) Compute the set $R$ of the results of $Q$ on $T$, keeping conditions $cond_r$ for each $r \in R$ (cf.
    Algorithm 2.1).
(b) Let $w$ be a fresh event variable, with $\pi(w)=c$.
(c) For each node $n$ in $T$, gather all elementary operations that should apply to $n$, along with the
    corresponding conditions (i.e., gather, for each $r \in R$ and $n \in N_Q$, the operations that should
    apply to all $\mu_r^Q(n)$ with the corresponding $cond_r$), to build a set $\mathcal{O}_n$ of pairs $(o,c)$ where $o$ is
    either an insertion (with the corresponding data tree) or a deletion, and $c$ is a conjunction of
    atom literals.
(d) Browse the nodes of $T$, from bottom up; for each node $n$ such that $\mathcal{O}_n \neq \varnothing$, perform the
    following (in that order):
    (i) For each insertion operation of a data tree $t'$ with condition $c$, insert $t'$ below $n$, the root
        of $t'$ being conditioned by $c \cup \{w\}$. Redundant literals (that are implied by the conditions
        on $n$ or its ancestors) are removed. The same $t'$ might be inserted multiple times if it
        results from multiple matchings.
    (ii) Gather all conditions of deletions into a DNF $\psi$. Let $\psi'$ be the formula

             $\psi' = \bigwedge \gamma(n) \wedge (\neg w \vee \neg \psi).$

         Transform $\psi'$ into a DNF $\psi''$ that is logically equivalent to $\psi'$, and such that only one
         disjunct of $\psi''$ may be true at a time (just enumerate all valuations of the variables in $\psi'$
         that make $\psi'$ true). Let $k$ be the number of disjuncts in $\psi''$. Replace $n$ by $k$ deep copies of
         $n$ and its children, each one being annotated with one of the disjuncts of $\psi''$. Redundant
         literals (that are implied by the conditions on the ancestors of $n$) are removed.
(e) Return the updated prob-tree.
```



Event	Proba.
w_1	0.8
w_2	0.7

Event	Proba.
w_1	0.8
w_2	0.7
w_3	0.9

Figure 2.6: Result of applying the update from Figure 2.5 to an example prob-tree

We now state that our definition of updates on PW sets and prob-trees are consistent with each other:

Theorem 2.17. *Let T be a prob-tree and (τ, c) a probabilistic update operation. Then: $[\![(\tau, c)(T)]\!] \sim (\tau, c)([\![T]\!])$. In other words, the following diagram commutes:*

$$\begin{tikzcd}
\textit{prob-tree} \arrow[r, "(\tau,c)(\cdot)"] \arrow[d, "[\![\cdot]\!]"] & \textit{prob-tree} \arrow[d, "[\![\cdot]\!]"] \\
\textit{PW set} \arrow[r, "(\tau,c)(\cdot)"] & \textit{PW set}
\end{tikzcd}$$

Proof. Let $T = (t, W, \pi, \gamma)$, $\tau = (Q, U)$.

The proof is by recursion on the cardinality of U. If $U = \varnothing$, $(\tau, c)(T) = T$ and $(\tau, c)([\![T]\!]) = [\![T]\!]$, so $[\![(\tau, c)(T)]\!] = (\tau, c)([\![T]\!])$.

Suppose now that we know that for all U of cardinality $n-1$ ($n \geqslant 1$), $[\![(\tau, c)(T)]\!] \sim (\tau, c)([\![T]\!])$. Let o be an arbitrary element of U, operating on $n \in N_Q$. Let w be the fresh event variable introduced in $(\tau, c)(T)$. We consider $(\tau', c)(T)$ with $\tau' = (Q, U - \{o\})$ and assume, without loss of generality that the fresh event variable introduced in $(\tau', c)(T)$ is also w. We now compare $(\tau', c)(T)$ with $(\tau, c)(T)$.

Let R be the set of results of the query Q on T, with associated conditions $cond_r$ for each $r \in R$. Then $(\tau, c)(T)$ can be obtained from $(\tau', c)(T)$ in the following way (some differences may arise this way, since redundant conditions can appear in one of the trees, but the result of the transformation is *structurally equivalent* (cf. Section 2.4) to $(\tau, c)(T)$):

1. If o is an insertion of a data tree t', by inserting t' under all (clones of) $\mu_r^Q(n)$ for each $r \in R$, conditioned by $cond_r \cup \{w\}$. We allow for clones, since it is possible than an ancestor of $\mu_r^Q(n)$ (or $\mu_r^Q(n)$ itself) has been conditionally deleted in $(\tau', c)(T)$.

2. If o is a deletion, by cloning all (clones of) $\mu_r^Q(n)$ for each $r \in R$ and annotating them by a condition stating that the node remains in the tree if, first, $\gamma(\mu_r^Q(n))$ is true, and, second, w is false or $cond_r$ is false.

We can now compare $(\tau, c)[\![T]\!]$ with $(\tau', c)[\![T]\!]$. The former is obtained from the latter by doing the following:

1. If o is an insertion of a data tree t', by inserting t' under $\mu_r^Q(n)$ in each possible world matching Q with probability c for all $r \in R$.

2. If o is a deletion, by deleting $\mu_r^Q(n)$ in each possible world matching Q with probability c for all $r \in R$.

Just observe that the possible worlds matching Q, for $r \in R$, are exactly those produced in $[\![T]\!]$ by subsets $V \subseteq W$ that are compatible with $cond_r$. By comparing the two transformations that are performed here, we see that $[\![(\tau, c)(T)]\!]$ is obtained from $[\![(\tau', c)(T)]\!]$ in the same way as $(\tau, c)([\![T]\!])$ is obtained from $(\tau', c)([\![T]\!])$. We use then the recursion hypothesis, which implies that $(\tau', c)([\![T]\!]) \sim [\![(\tau', c)(T)]\!]$. This concludes the proof. □

An interesting side benefit of using the prob-tree model is the possibility to keep *lineage* (or *provenance*) information about the data. Since every node is conditioned by event variables corresponding to update operations, we can associate meta-data to these variables to record information about the origin of the corresponding operation. Note that these variables are preserved throughout the whole process; a prob-tree system is able to deliver, along with query results and probabilities, information about the lineage associated with a piece of data (possibly updated more than once) and query results. This is especially interesting in the context of a content-centric imprecise process, as the one discussed in Chapter 1.

2.3.3 Complexity of Queries and Updates

In what follows, we assume given a locally monotone query language $\mathcal{Q}$ and an algorithm to answer queries over trees that are "lifted" to queries/updates over prob-trees (in the case of updates, we give complexity results for elementary updates). We next analyze the complexity of the algorithms for querying and updating prob-trees. Observe that in the following proposition, the complexity of the operations on prob-trees is stated in terms of the complexity of the corresponding operation on data trees. So, for instance, since the data complexity of tree-pattern queries with join is **PTIME**, an immediate consequence of the proposition is that over prob-trees, it is also **PTIME**. More precisely, let $|\cdot|$ denote the size (number of nodes, of literals) of a prob-tree (or of a set of possible worlds), and **time** denotes the time it takes to evaluate the query or operation. Then the complexity of Algorithms 2.1 and 2.2, that is, an upper bound on the complexity of querying and updating prob-trees (based on an algorithm to answer $\mathcal{Q}$ queries), is as follows:

Proposition 2.18. *Let T be a prob-tree with underlying data tree t. Let Q be a query over T, and i_Q and d_Q be respectively an insertion and deletion on T, with Q as defining query. Then:*

$$\begin{aligned}
\mathbf{time}(Q(T)) &\leqslant \mathbf{time}(Q(t)) + O(|Q(t)|\cdot|T|)\\
\mathbf{time}(i_Q(T)) &\leqslant \mathbf{time}(Q(T)) + O(|Q(t)|\cdot|T|)\\
|i_Q(T)| &\leqslant |T| + O(|Q(t)|\cdot|T|)\\
\mathbf{time}(d_Q(T)) &\leqslant \mathbf{time}(Q(T)) + O(|Q(t)|\cdot 2^{|T|})\\
|d_Q(T)| &\leqslant |T| + O(|Q(t)|\cdot 2^{|T|})
\end{aligned}$$

Proof. These are straightforward from the definitions of Algorithms 2.1 and 2.2. The combinatorial explosion of deletions happens when a query has multiple results (essentially because, in this case, we need to express the negation of a disjunction of conjunctions in terms of a disjunction of conjunctions), and, as we shall see in Theorem 2.26, this complexity is inherent to the problem of deletion in prob-trees. □

2.4 Equivalence of Prob-Trees

One defines the notion of equivalence between prob-trees directly based on data tree isomorphism. It essentially states that two prob-trees use the same event variables and that for each assignment of values to the event variables, they define the same possible world.

Definition 2.19. Let $T = (t, W, \pi, \gamma)$, $T' = (t', W, \pi, \gamma')$ be two prob-trees (over the same event variables and distribution). Then T and T' are *structurally equivalent* (denoted $T \equiv_{\text{struct}} T'$) if for each $V \subseteq W$, $V(T) \sim V(T')$.

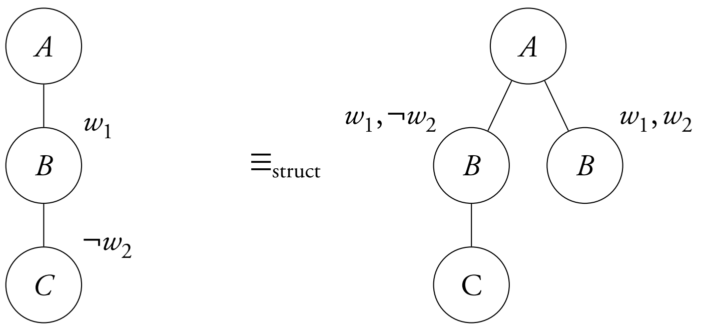


Figure 2.7: Example of structurally equivalent prob-trees

Figure 2.7 gives an example of two structurally equivalent prob-trees. Note that an alternative definition of equivalence of prob-trees, based on their possible-worlds semantics, is discussed in Section 2.6. We have a simple complexity upper bound about structural equivalence:

Proposition 2.20. *Determining if two prob-trees (over the same event variables and distribution) are structurally equivalent is* ***coNP****.*

Proof. The complement of this problem can be solved with the **NP** algorithm shown as Algorithm 2.3. □

Algorithm 2.3 Structural equivalence of prob-trees (non-deterministic)

```
INPUT: Two prob-trees $T_1$ and $T_2$ on the same event variable set $W$ and with the same probability distribution $\pi$.
OUTPUT: true if $T_1 \not\equiv_{\text{struct}} T_2$.
(a) Guess a subset $V$ of $W$.
(b) Compute $V(T_1)$ and $V(T_2)$ in linear time.
(c) If $V(T_1) \not\sim V(T_2)$, return true. (Isomorphism of labeled unordered data trees can be determined in linear time, cf. [AHU74].)
```

We stated in [4] a more precise result, that this problem is **coRP** [Pap94] (a randomized complexity class). However, we discovered since then an error in the proof: Lemma 2 from [4] does not hold. This can be seen for example on the prob-trees shown in Figure 2.7. The proof is still of interest, though, and we use it to prove a weaker result on *flat* prob-trees (that is, prob-trees of depth 1). The precise characterization of the complexity of the structural equivalence problem is open.

We use some bridge to (i) the number of disjuncts satisfied by valuations of DNF formulas and (ii) multivariate polynomials.

Definition 2.21. Let ψ and ψ' be two propositional formulas in disjunctive normal form. We say that ψ and ψ' are *count-equivalent*, denoted $\psi \overset{+}{\equiv} \psi'$, if, for any valuation ν of the variables appearing in ψ and ψ', the same number of disjuncts is satisfied by ν in ψ and in ψ'.

We note that this is a stronger notion than simple propositional formula equivalence. For instance, the formulas $X \lor (X \land Y)$ and X are equivalent but not count-equivalent. We indicate next how we can relate count-equivalence of formulas in DNF with equality of multivariate polynomials.

Definition 2.22. Let ψ be a propositional formula in disjunctive normal form, over variables $X_1 \dots X_n$. Let ψ' be a formula in DNF obtained from ψ by removing every disjunct containing incompatible atomic conditions, and by removing duplicate atomic conditions from each disjunct. The *characteristic polynomial of* ψ, denoted P_ψ, is the multivariate polynomial in $X_1 \dots X_n$ with integer coefficients, obtained from ψ' in the following manner:

(i) Positive literals X_i are left as is.
(ii) Negative literals $\neg X_i$ are replaced by $(1 - X_i)$.
(iii) Disjunction is replaced by addition.
(iv) Conjunction is replaced by multiplication.

Thus, the characteristic polynomial of $X \vee (X \wedge Y)$ is $X + X \cdot Y$. We now state that equality of characteristic polynomials is the same as count-equivalence.

Lemma 2.23. *Let ψ and ψ' be two propositional formulas in disjunctive normal form. Then, $\psi \overset{+}{\equiv} \psi'$ if and only if $P_\psi = P_{\psi'}$.*

Proof. One direction is obvious, i.e., if $P_\psi = P_{\psi'}$ then $\psi \overset{+}{\equiv} \psi'$. For that, just observe that the number of conjuncts satisfied by some valuation ν in ψ is the value of P_ψ for this valuation. Now to consider the converse, first observe that P_ψ and $P_{\psi'}$ are polynomials with degree at most 1 in every variable (this comes from the normalization of the formula in DNF used in Definition 2.22). Suppose that $\psi \overset{+}{\equiv} \psi'$. Consider the development of P_ψ:

$$P_\psi(X_1 \dots X_n) = \sum_{V \subseteq [\![1;n]\!]} \alpha_V \prod_{i \in V} X_i$$

and similarly for $P_{\psi'}$ with coefficients α'_V.

We have, for each tuple $(x_1 \dots x_n)$ of $\{0,1\}^n$,

$$P_\psi(x_1 \dots x_n) = P_{\psi'}(x_1 \dots x_n),$$

that is to say,

$$\sum_{U \subseteq \{\, i \mid x_i = 1 \,\}} \alpha_U = \sum_{U \subseteq \{\, i \mid x_i = 1 \,\}} \alpha'_U.$$

We can then prove by induction on the cardinality of V that this implies that $\forall V \subseteq [\![1;n]\!], \alpha_V = \alpha'_V$, which means that $P_\psi = P_{\psi'}$. □

There are actually deep aspects in this result, related to recent works by Green and Tannen in [GKT07]. DNF formulas and multivariate polynomials can be seen as instances of *provenance semirings*, that naturally arise in a number of contexts, especially incomplete and probabilistic databases.

The following result gives the (straightforward) relation between structural equivalence of flat prob-trees and count-equivalence.

Lemma 2.24. *Let $T = (t, W, \pi, \gamma)$ and $T' = (t', W, \pi, \gamma')$ be two prob-trees of depth 1 (over the same event variables and probability distribution).*

Let $u_1 \dots u_n$ be representative elements of the n equivalence classes implied by equality of both values and labels over the leafs (that is, the children of the root) of T and T'. For $1 \leqslant i \leqslant n$, let ψ_i be the disjunction of the conditions attached to the leafs of T with the same value and label as u_i, and let ψ'_i be the same for T'.

Then, $T \equiv_{\text{struct}} T'$ if and only if $\varphi(r) = \varphi(r')$ and, for each $1 \leqslant i \leqslant n$, $\psi_i \overset{+}{\equiv} \psi'_i$.

Proof. Just observe that the number of disjuncts of ψ_i (respectively, ψ'_i) that are made true by some valuation ν of the event variables is the number of leafs of $\nu(T)$ (respectively, $\nu(T')$) that have the same value and label as u_i. □

This leads to a direct complexity result for structural equivalence on flat prob-trees:

Proposition 2.25. *There is a PTIME algorithm, that, given two prob-trees of depth* 1*, always returns* `true` *if the prob-trees are structurally equivalent and returns* `false` *if the prob-trees are not structurally equivalent with probability at least ½ (that is, determining if two flat prob-trees are structurally equivalent is a* ***coRP*** *problem).*

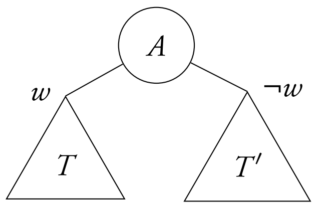
Proof. The algorithm relies on Lemmas 2.23 and 2.24. We use the Schwartz-Zippel lemma [Sch80, Zip79], which states that the probability that a multivariate polynomial of degree d is zero on a point each coordinate of which is randomly chosen in some finite set S is $d/|S|$.

Algorithm 2.4 Structural equivalence of flat prob-trees (randomized)

```
INPUT: Two flat prob-trees $T$ and $T'$, a finite set $S$ of integers, a positive integer $m$.
OUTPUT: true if $T \equiv_{\text{struct}} T'$; false if $T \not\equiv_{\text{struct}} T'$ with probability ½.
(a) Compute equivalence classes for the equality of both values and labels among leafs of $T$ and $T'$.
(b) Gather the formulas in DNF $\psi_i$ and $\psi'_i$ corresponding to the conditions on the leafs of $T$
    and $T'$ in the equivalence class $i$.
(c) For each equivalence class $i$, choose at random $m$ points of $S^p$, where $p$ is the number of variables
    of $P_{\psi_i} - P_{\psi'_i}$, and evaluate this polynomial in these points.
(d) If all these evaluations return 0 for each $i$, return true else return false.
```

The algorithm is presented as Algorithm 2.4. We have the following lower bound for the probability that it is correct when it returns `false`: $\left(1-\left(\frac{|W|}{|S|}\right)^m\right)^n$, where n is the number of nodes. This probability is greater than ½ as soon as m and S are chosen such that $|S| \geqslant \frac{|W|}{\sqrt[m]{1-(1/2)^{1/n}}}$. □

Note that determining whether a prob-tree is independent of some event variable is actually computationally as complex as deciding equivalence between prob-trees. Indeed, if T and T' are two prob-trees, determining if T is structurally equivalent to T' can be done by determining if the following tree is independent of w (a fresh variable):



2.5 Other Issues about Prob-Trees

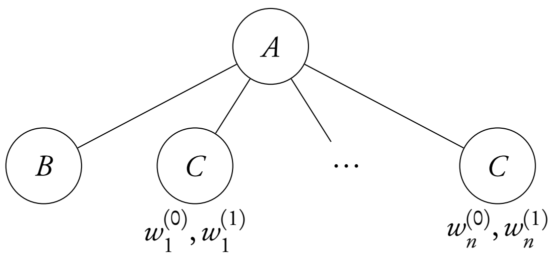
In this section, we consider three natural problems about prob-trees that all highlight some inherent complexity in dealing with imprecise data. First we show that some deletion may cause a combinatorial explosion. We then prove that similar phenomena arise when we try to restrict the possible worlds (i) to have at least a threshold probability and (ii) to be valid with respect to some DTD.

2.5.1 Deletions

First we consider deletion. In this part, we assume that our query language is expressive enough to express the following deletion: $d_0 =$ "If the root has a C child, then delete all B children of the root." (this is a reasonable assumption, since the update is rather basic).

Theorem 2.26. *For all $n \in \mathbb{N}$, there exists a prob-tree T, of size $O(n)$, such that for each prob-tree T' such that $T' \equiv_{\text{struct}} d_0(T)$, the size of T' is $\Omega(2^n)$.*

Proof. Consider the following prob-tree T, which has $n+2$ nodes and $2n$ event variables, each appearing only once (we take an arbitrary probability distribution π, say $\pi(n) = 1/2$ for all n):



Let T' be a prob-tree such that $T' \equiv_{\text{struct}} d_0(T)$; we assume that T' does not contain any inconsistent condition (corresponding nodes may safely be removed from T'). We also assume that the deletion has a confidence of 1 (that is, it does not introduce a new event variable).

T' is necessarily some prob-tree of depth 1 with root node A, and with a number of B and C children. Let Ψ be the set of conditions annotating nodes labeled by B. Observe that for all $\psi \in \Psi$, and for all $1 \leqslant k \leqslant n$, either $\neg w_k^{(0)}$ or $\neg w_k^{(1)}$ appears in ψ (otherwise, there is a possible world for T' where both B and C nodes appear, which is a contradiction with the definition of d_0).

Let now $\{b_1 \dots b_n\}$ be an arbitrary element of $\{0,1\}^n$. Let ν be the valuation of the event variables such that $\forall 1 \leqslant k \leqslant n$, $\nu(w_k^{(b_k)}) = 0$ and $\nu(w_k^{(1-b_k)}) = 1$. Then $\nu(T)$ is the subtree of T with only two nodes labeled by A and B. Therefore, $\nu(d_0(T)) = d_0(\nu(T)) = \nu(T)$. This means that there exists $\psi_{b_1 \dots b_n} \in \Psi$ such that $\nu \models \psi_{b_1 \dots b_n}$.

Assume now by contradiction there exists $b_1 \dots b_n$, $b'_1 \dots b'_n$ and $1 \leqslant k \leqslant n$, such that $\psi_{b_1 \dots b_n} = \psi_{b'_1 \dots b'_n} = \psi$ and $b_k \neq b'_k$. But we have already noted that ψ contains either $\neg w_k^{(0)}$ or $\neg w_k^{(1)}$. In the former case, we cannot have either $b_k = 0$ or $b'_k = 0$; in the latter, we cannot have $b_k = 1$ or $b'_k = 1$. This leads to a contradiction, which means that to each element of $\{0,1\}^n$ corresponds a different element of Ψ. T' has then at least 2^n different literals, which concludes the proof. □

2.5.2 Threshold Probability

We consider next what happens when some probability threshold is imposed on a prob-tree.

Given a prob-tree, one may want to eliminate the possible worlds that are too unlikely. More precisely, consider a prob-tree T with $[\![T]\!] = \{(t_1,p_1) \dots (t_n,p_n)\}$. Let us further assume that $[\![T]\!]$ is normalized, i.e., that there are no i, j distinct with $t_i \sim t_j$. Suppose we fix some p for a minimum threshold on probability. We want to consider the set of possible worlds with probability greater than the threshold:

$$\{(t_i,p_i) \in [\![T]\!] \mid p_i \geqslant p\}.$$

This set, however, is not a PW set, since the probabilities may not sum to 1. In order to use our framework, and especially to perform comparisons with the possible-worlds semantics of a prob-tree, we cumulate the probabilities that were lost (those of the trees violating the constraint) and assign them to the tree t_r consisting simply of a root (with the same label as the common root label). In other words, this comes down to interpreting the root-tree as *inconsistent*. We thus define now:

$$[\![T]\!]_{\geqslant p} = \{(t_i,p_i) \in [\![T]\!] \mid p_i \geqslant p\} \cup \left\{(t_r,p_{\text{missing}})\right\}$$

where

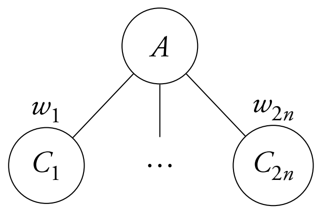
$$p_{\text{missing}} = \sum_{\substack{1\leqslant i\leqslant n\\ p_i<p}} p_i.$$

We now introduce a similar notion of restriction directly for prob-trees, since we want to deal with structural equivalence of prob-trees. A restriction of a prob-tree $T=(t,W,\pi,\gamma)$ to possible worlds with probability greater than some threshold p is some prob-tree $T'=(t',W,\pi,\gamma')$ such that for all $V\subseteq W$ with $V(T)$ appearing in $[\![T]\!]_{\geqslant p}$, $V(T')\sim V(T)$ and for all $V\subseteq W$ with $V(T)$ not appearing in $[\![T]\!]_{\geqslant p}$, $V(T')=t_r$. In other words, we want to keep the same event variables as in T, with the same signification for valid possible worlds, and with the assumption that the root tree denotes an inconsistent state. Obviously, T' is not unique; however, it is guaranteed to exist, since it is always possible to annotate children of the root with conditions describing uniquely each $V\subseteq W$.

The question is now, is there a compact prob-tree restriction of a prob-tree to some probability threshold? The answer is no:

Theorem 2.27. *For all $n\in\mathbb{N}$, there exists a prob-tree T of size $O(n)$ and a probability threshold p such that for each prob-tree T' that is a restriction of T to the threshold p, the size of T' is $\Omega(2^n)$.*

Proof. Consider the following prob-tree $T=(t,\pi,W,\gamma)$, with $2n+1$ nodes and $2n$ event variables, each appearing once; we choose a uniform probability distribution $\pi(w_i)=1/4$ and a probability threshold $p=(3/16)^n$:



The probability of a given possible world of $[\![T]\!]$ is

$$\left(\frac{1}{4}\right)^k\left(\frac{3}{4}\right)^{2n-k}$$

where k is the number of event variables set to true. This is greater than p if and only if $k\leqslant n$. Hence, the trees appearing in $[\![T]\!]_{\geqslant p}$ are exactly the subtrees of the data tree underlying T with at most n leafs; besides, distinct valuations of event variables yield distinct data trees since each node of T has a distinct label. All labels C_1, …, C_{2n} appear in $[\![T]\!]_{\geqslant p}$ since they have a symmetric role in T.

Let $T' = (t', W, \pi, \gamma')$ be a restriction of T to p. T' is necessarily a flat prob-tree with leafs of the root labeled by C_1, …, C_{2n} (there may be multiple leafs with the same label). For $1 \leqslant i \leqslant 2n$, let Ψ_i be the set of conjunctions annotating nodes labeled by C_i.

We shall prove that, for all $1 \leqslant i \leqslant 2n$, for all $K \subseteq [\![1;2n]\!]$ such that $i \in K$ and $|K| = n$, there exists $\psi \in \Psi_i$ such that $\{\neg w_k \mid 1 \leqslant k \leqslant 2n, k \notin K\} \subseteq \psi$ and all other literals of ψ are positive. Assume by contradiction that this is not the case. Then, there is a $1 \leqslant i \leqslant 2n$, a $K \subseteq [\![1;2n]\!]$ such that $i \in K$ and $|K| = n$ such that no element $\psi \in \Psi_i$ verifies that $\{\neg w_k \mid 1 \leqslant k \leqslant 2n, k \notin K\} \subseteq \psi$ with all other literals ψ positive. $\{w_k \mid k \in K\}$ is a subset of W that leads to a possible world of $[\![T]\!]_{\geqslant p}$, in which the node labeled by C_i appears; therefore, there exists a $\psi \in \Psi_i$ such that $\psi \subseteq \{w_k \mid k \in K\} \cap \{\neg w_k \mid k \notin K\}$. By hypothesis, there must be a $k \notin K$ such that $\neg w_k \notin \psi$. But then, $\{w_k \mid k \in K\} \cup w_k \models \psi$, whereas as this subset of W is of cardinality $n+1$, it should yield the root tree.

What we just proved gives the fact that there exists at least $\binom{2n-1}{n-1}$ distinct elements in each Ψ_i. But:

$$\binom{2n-1}{n-1} = \frac{(2n-1)!}{(n-1)! \cdot n!} = \frac{(2n)!}{2(n!)^2} \sim \frac{\sqrt{4\pi n}\left(\frac{2n}{\mathrm{e}}\right)^{2n}}{4\pi n\left(\frac{n}{\mathrm{e}}\right)^{2n}} = \frac{2^{2n-1}}{\sqrt{\pi n}} = \Omega(2^n)$$

using Stirling's formula. □

Note that we used here structural equivalence between prob-trees. It is still open whether this result still holds for a less restrictive notion of equivalence, i.e., when looking for a prob-tree T' such that $[\![T']\!] \sim [\![T]\!]_{\geqslant p}$ (i.e., *semantic equivalence*, as in Section 2.6.2).

2.5.3 Validation

Finally we consider validity with respect to a DTD.

A Document Type Definition for an XML document defines the constraints applying to the children of a node using a *sequence* operator (`(A,B)`), a *disjunction* operator (`(A|B)`) and *repetition* operators (`(A*)`, `(A+)`, `(A?)`). As we consider unordered trees here, we do not consider the sequence operator as such. To simplify, we use the following definition of DTDs that simply gives a lower and upper bound for the number of occurrences of nodes with a given label l' as children of some node labeled by l.

Definition 2.28. A *Document Type Definition* (*DTD*) D is a function over some finite subset L of the set of labels $\mathscr{L}$ such that for $l \in L$, $D(l)$ is a finite set of elements of $\mathscr{L} \times [\![0;+\infty[\![\times [\![1;+\infty]\!]$ and, if $(l_1, p_1, q_1) \in D(l)$ and $(l_2, p_2, q_2) \in D(l)$, either $l_1 \neq l_2$ or $(l_1, p_1, q_1) = (l_2, p_2, q_2)$.

We use the following notation, for $l \in L$: $D_-(l)(l')$ and $D_+(l)(l')$ are respectively the unique p and q such that $(l', p, q) \in D(l)$ if such p and q exist; otherwise, we denote $D_-(l)(l') = 0$ and $D_+(l)(l') = 0$.

Definition 2.29. Let D be a DTD and $t = (N, E, r, \varphi, \nu)$ a data tree. Let L be the domain of D. We say that *t satisfies D* (denoted $t \models D$) if, for each $s \in N$ such that $\varphi(s) \in L$, and for each $l' \in L$:

$$D_-(\varphi(s))(l') \leqslant \left|\{s' \in N \mid \varphi(s') = l' \wedge (s, s') \in E\}\right|;$$
$$D_+(\varphi(s))(l') \geqslant \left|\{s' \in N \mid \varphi(s') = l' \wedge (s, s') \in E\}\right|.$$

Note that we do not impose any condition on nodes of t whose label is not in the domain of the DTD.

Given a prob-tree T and a DTD D, two natural questions arise:

1. (DTD-Satisfiability) $\{(t,p) \in [\![T]\!] \mid t \models D\} \overset{?}{\neq} \emptyset$
2. (DTD-Validity) $\{(t,p) \in [\![T]\!] \mid t \models D\} \overset{?}{\sim} [\![T]\!]$

We have the following complexity result:

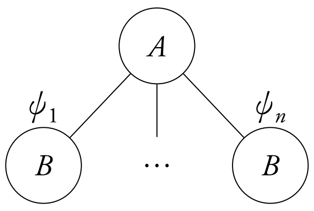
Theorem 2.30.

1. DTD-Satisfiability *is **NP**-complete in the number of event variables (and linear in the number of nodes in the tree).*
2. DTD-Validity *is **coNP**-complete in the number of event variables (and linear in the number of nodes in the tree).*

Proof. We use a reduction of SAT. The beginning of the construction is the same in both cases.

Let θ be a propositional logic formula, in conjunctive normal form (i.e., an input to the SAT problem). Let $\psi_1 \dots \psi_n$ be the terms of $\neg\theta$ in disjunctive normal form (the DNF of $\neg\theta$ is computed in a linear time from θ which is in CNF).

Let T be the following prob-tree:



1. Consider the DTD D: $D(A) = \{(B, 0, 0)\}$.

$$\begin{aligned} & \{(t,p) \in [\![T]\!] \mid t \models D\} \neq \emptyset \\ \iff & \psi_1 \vee \cdots \vee \psi_n \text{ not a tautology} \\ \iff & \theta \text{ is satisfiable.} \end{aligned}$$

   Since the construction of the reduction is linear in the size of θ, this proves that the DTD satisfiability problem is **NP**-hard.

   Moreover, here is an **NP** algorithm for the DTD satisfiability problem, which concludes the proof of its **NP**-completeness: Guess a valuation ν of the event variables of T, and return `true` if $\nu(T)$ satisfies the DTD (which can be checked in linear time).

2. Consider the DTD D: $D(A) = \{(B, 1, +\infty)\}$.

$$\begin{aligned} & \{(t,p) \in [\![T]\!] \mid t \models D\} \sim [\![T]\!] \\ \iff & \psi_1 \vee \cdots \vee \psi_n \text{ tautology} \\ \iff & \theta \text{ is not satisfiable.} \end{aligned}$$

Since the construction of the reduction is linear in the size of θ, this proves that the validity problem is **coNP**-hard.

Moreover, here is an **NP** algorithm for the complement of the validity problem, which concludes the proof of its **coNP**-completeness: Guess a valuation ν of the event variables of T and return `true` if $\nu(T)$ does not satisfy the DTD.

Observe that the DTDs we used in the proof are all of constant size. □

We could also consider the DTD-Restriction problem: Given a DTD and a prob-tree, can we represent concisely as a prob-tree the set of possible worlds valid against the DTD? This would require similar definitions as in Section 2.5.2. Actually, the result is, similarly, negative and the proof is done in exactly the same way, with a DTD requiring that the node A has at most n children labeled by C. The C_i nodes are replaced by C nodes with a D_i child in order to give them the same label while keeping them distinguishable.

2.6 Variants of the Prob-Tree Model

In this section, we briefly consider variants of the prob-tree model presented up to here, and discuss their complexity. Namely, we consider (i) a tree model with set semantics, instead of our *multi-set* semantics; (ii) the notion of *semantic* equivalence (in place of structural equivalence); (iii) a prob-tree model where nodes are assigned arbitrary propositional formula (and not simply conjunctions) as conditions; and (iv) ordered trees.

2.6.1 Set Semantics

In this chapter, we introduced a data model with a *multi-set* (or *bag*) semantics. One can consider instead a *set semantics*. One just has to redefine isomorphism between data trees inductively as follows. Let t, t' be two trees. They are isomorphic if their roots have the same label and if each subtree of the root of t is isomorphic to some subtree of the root of t', and symmetrically. Most definitions of this chapter can then be applied as is, relying on this new version of data tree isomorphism. The results about queries and updates remain, including the exponential complexity of deletions from Theorem 2.26 (the proofs are almost unchanged). An important difference, however, is for structural equivalence, for which there is now a simple way of proving **coNP**-completeness: Just observe that we no longer deal with *count-equivalence*, but with classical equivalence of propositional formulas.

2.6.2 Semantic Equivalence

Structural equivalence is only relevant for prob-trees that share the same event variables. If we want to compare prob-trees with different sets of events, we can define another kind of equivalence, through their possible-worlds semantics: T and T' are *semantically equivalent* (denoted $T \equiv_{\text{sem}} T'$) if $[\![T]\!] \sim [\![T']\!]$. The first natural question is that of the relation between structural and semantic equivalence.

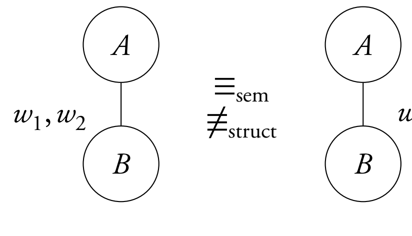
Proposition 2.31. *Let $T = (t, W, \pi, \gamma)$, $T' = (t', W', \pi', \gamma')$ be two prob-trees, with $W = W'$ and $\pi = \pi'$. Then*
(i) If $T \equiv_{\text{struct}} T'$, then $T \equiv_{\text{sem}} T'$;

(ii) $T \equiv_{\text{struct}} T'$ *if and only if, for each probability distribution* π'' *over* W*,* $(t, W, \pi'', \gamma) \equiv_{\text{sem}} (t', W, \pi'', \gamma')$*.*

Proof. (i) is obvious. Now suppose that for each π'' over W, $(t, W, \pi'', \gamma) \equiv_{\text{sem}} (t', W, \pi'', \gamma')$. To conclude the proof, it clearly suffices to show that $T \equiv_{\text{struct}} T'$. For each $V \subseteq W$, let π_V be the probability distribution that maps $w \in V$ to 1 and $w \in W - V$ to 0. Then, if T_V and T'_V denote respectively the prob-trees obtained from T and T' by exchanging the original probability distribution π with π_V, $[\![T_V]\!] = \{(V(T), 1)\}$ and $[\![T'_V]\!] = \{(V(T'), 1)\}$ and we have thus $V(T) = V(T')$.

Note that strictly speaking, we disallowed variables with zero probability. So to be precise, we should use ε instead of 0 and $1-\varepsilon$ instead of 1. If ε is chosen so that $(1-\varepsilon)^n > 2^n\varepsilon$ (which is always possible for a sufficiently low value of ε), $V(T)$ and $V(T')$ are the elements of highest probability of, respectively, $[\![T]\!]$ and $[\![T']\!]$. □

Note that $T \equiv_{\text{sem}} T'$ does not imply $T \equiv_{\text{struct}} T'$. For instance, if w_1, w_2, w_3 verify $\pi(w_3) = \pi(w_1) \cdot \pi(w_2)$, we have :



Clearly, there is an **EXPTIME** algorithm for determining if two prob-trees are semantically equivalent (just compute the possible-worlds sets, normalize them, and check if they are isomorphic, which can be decided in quadratic time in the number of possible worlds). It is open whether the problem also belongs to a lower complexity class. Similarly, it is open whether Theorem 2.26 on the complexity of deletions still holds for semantic equivalence.

2.6.3 Arbitrary Propositional Formula

In prob-trees, the conditions we use are conjunctions of literals. A natural extension is to allow any propositional formula (including disjunctions) as conditions. A question is how this is affecting the complexity. First, one can show that the evaluation of boolean queries is **NP**-complete (assuming the underlying query language over data tree is in **PTIME** and includes, say, tree pattern queries). The fact that it is **NP** is obvious, and there is a linear-time reduction of SAT to this problem. Then, the cost of an update operation is now **PTIME** (again assuming the underlying language on data trees is **PTIME**). Indeed, we can now simply annotate inserted or deleted nodes by complex formulas. In particular, Theorem 2.26 is no longer valid. So this model privileges updates (that are cheap) against queries (that are expensive). It is not adapted to the applications that motivated our work.

2.6.4 Order Semantics

By considering ordered trees, we would move closer to standard XML. The situation is more intricate and would require totally different techniques. The complexity would typically be higher because of the inherent combinatorics that is introduced. Some of the algorithms, especially for querying and updating, may still be valid if we just add an additional label to each node indicating its position among its sibling.

2.7 Other Models for Probabilistic Data in XML

We present here two alternatives to prob-trees for representing probabilistic data in trees. The first one, simple probabilistic (SP) trees, is a simple and intuitive model, that is shown to be less expressive than the possible-worlds (and hence the prob-tree) model. The second one has been studied in the literature, and can be thought as a combination of possible worlds and SP trees. We show that it can be seen as a restricted case of the prob-tree model.

2.7.1 Simple Probabilistic Model

In the spirit of probabilistic models for the relational model, we can attach a probability to each node in a data tree. The intuition is that it captures the probability of that node to be present, assuming its parent is. A limitation of this model is that the only probability dependency that is captured is between nodes in a parent/child relationship. We study now this model and highlight its limitations.

Definition 2.32. A *simple probabilistic (SP) tree* T is a pair (t,π) where $t=(N,E,r,\varphi,\nu)$ is a data tree and $\pi : N \to]0;1]$ such that $\pi(r)=1$ assigns probabilities to tree nodes.

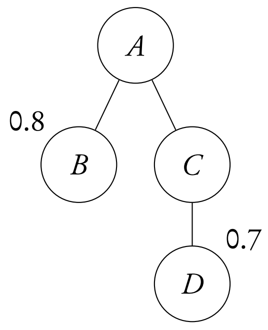


Figure 2.8: Example SP tree

Such an SP tree is represented as in Figure 2.8. Only probabilities not set to 1 are shown.

We can give a *possible-worlds semantics* to an SP tree as follows. Choose an arbitrary $X \subseteq N$. Consider t_X the tree obtained by removing from t all nodes not in X (and their descendants). We assign to this tree, the probability:

$$p_X = \prod_{n\in X} \pi(n) \prod_{n\in N-X} (1-\pi(n)).$$

The possible-worlds semantics of T, denoted $[\![T]\!]$, is defined as:

$$\{(t_X, p_X) \mid X \subseteq N\}.$$

Note that this definition correctly provides a PW set because:

$$\sum_{X\subseteq N} \left(\prod_{n\in X} \pi(n) \prod_{n\in A-X} (1-\pi(n)) \right) = 1$$

from classical probability equations.

As an example, the semantics of the SP tree from Figure 2.8 is the PW set on Figure 2.9. A natural question is then whether the SP tree model is as expressive as the PW tree model. The answer is no.

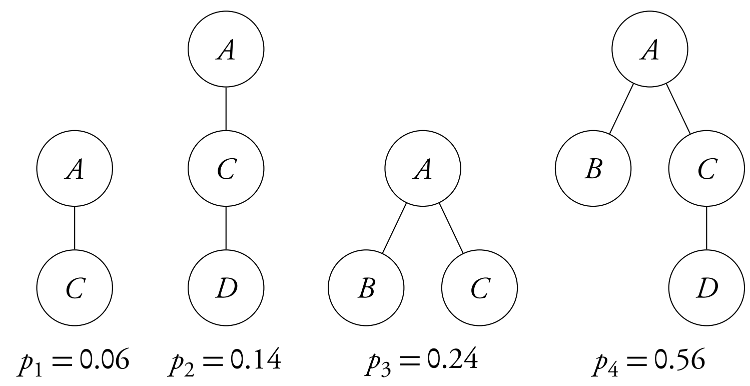


Figure 2.9: Possible-worlds semantics for the SP tree from Figure 2.8

Proposition 2.33. *There exists a PW set which is not the PW semantics of any SP tree.*

Figure 2.3 is an example of a PW set that has no equivalent SP tree. Intuitively, an equivalent SP tree would necessarily have nodes A, B, C and D, so the PW set would contain a tree with these four nodes, which it does not have, a contradiction.

This negative result is not a sufficient reason for ignoring the SP model, since one might argue that PW trees not representable in the SP model are of little practical interest. However, we next show some more fundamental shortcomings of that model. For this, consider queries and updates in the SP model.

Definition 2.34. Let Q be a locally monotone query and $T = (t, \pi)$ an SP tree. The result of Q on T, denoted $Q(T)$, is the set:

$$\bigcup_{u \in Q(t)} \left\{ \left(u, \prod_{n \text{ node of } u} \pi(n) \right) \right\}.$$

Once again, $Q(T)$ is *not* in general a PW set. The way to read $(t', \pi') \in Q(T)$ is that there is a probability π' that t' is a result to $Q(T)$. We have the following result, which shows that the definition is coherent with the PW semantics of an SP tree:

Theorem 2.35. *Let Q be a locally monotone query and T an SP tree. Then $Q(T) \sim Q([\![T]\!])$ (with the same abuse of notation as in Theorem 2.12).*

Proof. Observe that an SP tree $T = (t, \pi)$ can be transformed into a prob-tree $T' = (t, W, \pi', \gamma)$ in the following way: Each node n of t with $\pi(n) = 1$ is annotated by the empty condition, while each node n of t with $\pi(n) < 1$ is annotated by w_n, where w_n is a fresh event variable with probability $\pi'(w_n) = \pi(n)$. It is clear that T and T' share the same semantics, and that $Q(T) = Q(T')$ (just compare Definitions 2.34 and 2.11). The result comes then from Theorem 2.12. □

Now consider updates. The following result demonstrates that the SP tree model does not meet our requirements in terms of update, even when only using simple tree-pattern queries without any join.

Proposition 2.36. *There exists an SP tree T and a probabilistic update operation (τ, c) with a tree-pattern query as defining query, such that there is no SP tree whose semantics is isomorph to $(\tau, c)([\![T]\!])$.*

Proof. We choose T as the root tree, and τ the update operation that adds two nodes B and C as children of the root, with some confidence $c < 1$. $(\tau, c)(\llbracket T \rrbracket)$ is a set of two possible worlds: the root tree, with confidence $1 - c$, and a tree with three nodes, with confidence c. A candidate SP tree for this PW set needs to have three nodes; but in this case, its possible-worlds semantics has four different worlds, including worlds where B is present without C, and vice-versa. $\square$

In other words, "SP trees are not closed under update operations." This comes from the fact that dependencies between nodes are not expressible in the SP model. As seen is the proof, if an update operation adds two nodes under one common node, their presence is completely correlated, whereas the presence of siblings is independent in the SP model. Indeed, a simple *modification* that can be seen as an interdependent succession of an insertion and a deletion, cannot be represented in the SP model. Actually, the problem is even deeper: As the positions where the updates are performed are selected by a query, the update to be performed may be conditioned by the assumptions that were made to realize the query. There is no way to specify this kind of dependency in the SP model, whereas it is easy to do so with prob-trees.

2.7.2 ProTDB

ProTDB (Probabilistic Tree Data Base) is a system to manage probabilistic data by Nierman and Jagadish, presented in [NJ02]. The underlying model is also at the basis of [KS07] where Kimelfeld and Sagiv explore a number of querying issues, including incomplete queries. A remark at the end of [KS07] indicates that there are translations between the model underlying ProTDB (we call it the *p-document* model, following [KS07]) and the prob-tree model. We discuss and precise this issue in this section.

Let us first give a formal definition of p-documents:

Definition 2.37. A *p-document* is a 4-uple (t, I, M, π) where $t = (N, E, r, \varphi, \nu)$ is a data tree, $I \subseteq N$ is a set of *independent distribution* nodes, $M \subseteq N$ is a set of *mutually exclusive distribution* nodes (neither the root nor leafs can be distribution nodes, and $I \cap M = \varnothing$), and $\pi : A \rightarrow]0;1]$ is a probability distribution function that is defined for all nodes n that are children of a distribution node in t, and undefined elsewhere. An additional constraint is that for all mutually exclusive distribution node d, the following holds:

$$\sum_{n \text{ child of } d} \pi(n) \leqslant 1.$$

An example p-document is shown in Figure 2.10. Distribution nodes are shown as diamonds, while regular nodes are shown as circles as usual. The probability distribution values are displayed along the edge between distribution nodes and their children. Intuitively, independent distribution nodes behave similarly as nodes in the SP tree model, while mutually exclusive distribution nodes represent different possible worlds. We now define the semantics of documents in a more precise way.

Definition 2.38. Let $T = (t, I, M, \pi)$ be a p-document. The possible-worlds semantics of T is the set of pairs (tree, probability) obtained as the result from all the possible following choices, performed from top down (starting from a probability of 1):

(i) For each $d \in I$, we choose an arbitrary (possibly empty) subset of the set of children of d and attach them to the parent of d, removing thereby d and its other children. The probability of this choice is multiplied by the product of the probabilities of the selected children (1 when no child is selected).

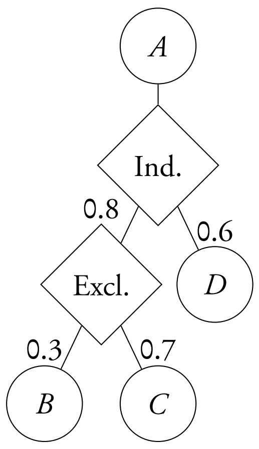


Figure 2.10: Example p-document

(ii) For each $d \in M$, we choose a single child of d, and attach it to the parent of d, removing thereby d and its other children. The probability of this choice is multiplied by the probability of the selected child. We can also choose not to select any child and remove d and all its children from the tree, the probability is then multiplied by $1 - s$ where s is the sum of the probabilities of the children of d.

It is easy to check that this is indeed a PW set. As an illustration, the possible-worlds semantics of the p-document from Figure 2.10 is given in Figure 2.11.

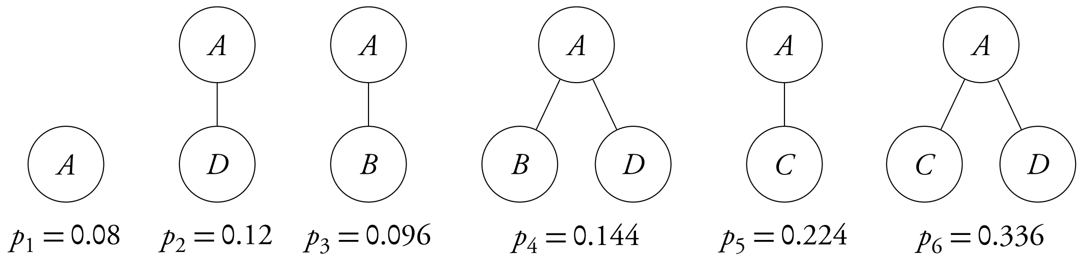


Figure 2.11: Possible-worlds semantics of the p-document from Figure 2.10

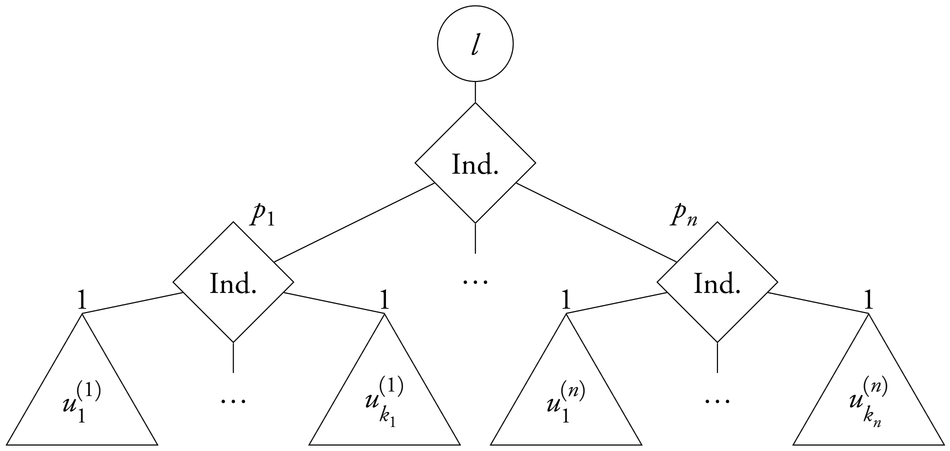
We now state the relations between the expressiveness and conciseness of p-documents on one hand, and prob-trees on the other hand: p-documents are fully expressive (as are prob-trees) but exponentially less concise than prob-trees.

Proposition 2.39.

1. *For each PW set S, there exists a p-document T such that $S \sim [\![T]\!]$.*
2. *There is a linear algorithm for transforming a p-document T into a prob-tree T' such that $[\![T]\!] \sim [\![T']\!]$.*
3. *For all $n \in \mathbb{N}$, there exists a prob-tree T of size $O(n)$ such that for each p-document T' satisfying $[\![T]\!] \sim [\![T']\!]$, the size of T' is $\Omega(2^n)$.*

Proof.

1. Let $S = \{(t_i, p_i)\}$, for $1 \leqslant i \leqslant n$. Let l be the common label of all roots of S. For each i, we denote by $u_1^{(i)}$, …, $u_{k_i}^{(i)}$ the k_i subtrees just below the root of data tree i. Then, the following p-document has for possible-worlds semantics S:



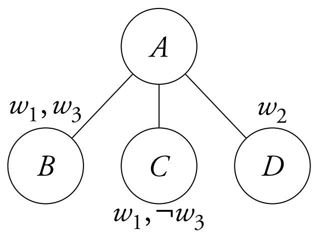
2. Refer to Algorithm 2.5, which is clearly linear. The fact that T' has the same semantics as T is straightforward. As an example, Figure 2.12 is the result of applying Algorithm 2.5 to the p-document from Figure 2.10. Obviously, this linearity in time gives also a linear bound on the size of the resulting prob-tree.

Algorithm 2.5 Transformation of a p-document into a prob-tree with the same semantics

Input: A p-document T.
Output: A prob-tree T' such that $[\![T]\!] \sim [\![T']\!]$.

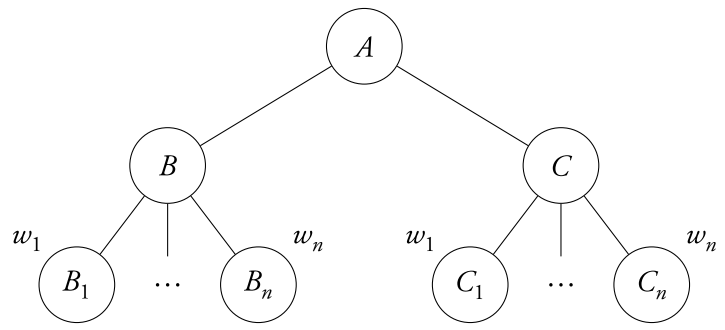
(a) Transform every independent distribution node into a node with a special label; introduce a fresh event variable w for every child of the distribution node, with associated probability the probability of the child, and add the literal w as a condition to the child.
(b) Transform every mutually exclusive distribution node into a node with a special label; introduce as many event variables as there are children of the distribution node (one less if the probabilities of the children sum to 1), with associated probability as in the proof of Theorem 2.4, and add conditions to the children as in the proof of Theorem 2.4.
(c) Remove every node with a special label, attaching their regular node closest descendants to their regular node closest ancestor, and merging event conditions on each ancestor→descendant path with the condition of the descendant.

3. Consider the following prob-tree T, with $2n+3$ nodes, n event variables, and a uniform probability distribution set to ½:



Event	Proba.
w_1	0.8
w_2	0.6
w_3	0.3

Figure 2.12: Prob-tree resulting from applying Algorithm 2.5 to the p-document from Figure 2.10



Let T' be a p-document such that $[\![T]\!] \sim [\![T']\!]$. First, let us prove that no (non-trivial, that is, with multiple choices) distribution node appears in T' below a node labeled by B or C (in other words, that all distribution nodes appear at the top of the tree, just below the root). Let us suppose by contradiction that this is not the case. Then, there exists a node labeled B_i for some i (or C_i but these play a symmetric role) that has for ancestor a distribution node which is below a B node; since the distribution node is supposed non-trivial, there must be a choice to make, and it is necessarily between the absence or presence of B_i. But then, this choice cannot be correlated with the absence or presence of C_i which is done, as C_i cannot be a descendant of B_i, in another branch. This yields a contradiction, since the absence or presence of B_i and C_i are mutually dependent.

All distribution nodes of T' appear at the top of the tree. Each subtree without any distribution node directly below these must then correspond to a possible world, and each possible world must appear that way. But there are 2^n different possible worlds; T' has thus more than 2^n nodes. □

Thus, any p-document can be transformed efficiently into a prob-tree, and the converse is false.

2.8 Implementation

This section discusses our implementation of a prob-tree system, namely FuzzyXML. Both the code and a demonstration are freely available at `http://pierre.senellart.com/software/fuzzyxml/`.

FuzzyXML supports TPWJ queries, and arbitrary probabilistic update operations. To facilitate the input of queries, a path language, similar to a subset of XPath, is introduced, along with a language for expressing joins. We do not present it in detail, but let us consider the example shown in Figure 2.13 The first two lines are a path expression, the third line is a corresponding join expression. The corresponding query is shown in Figure 2.14.

```
/books[-book[title$1[.="Foundations␣of␣Databases"]]/author$2]
      /book[-title$3][-author$4]/author$5
$2 = $4 and $1 != $3 and $4 != $5
```

Figure 2.13: Example path/join expression

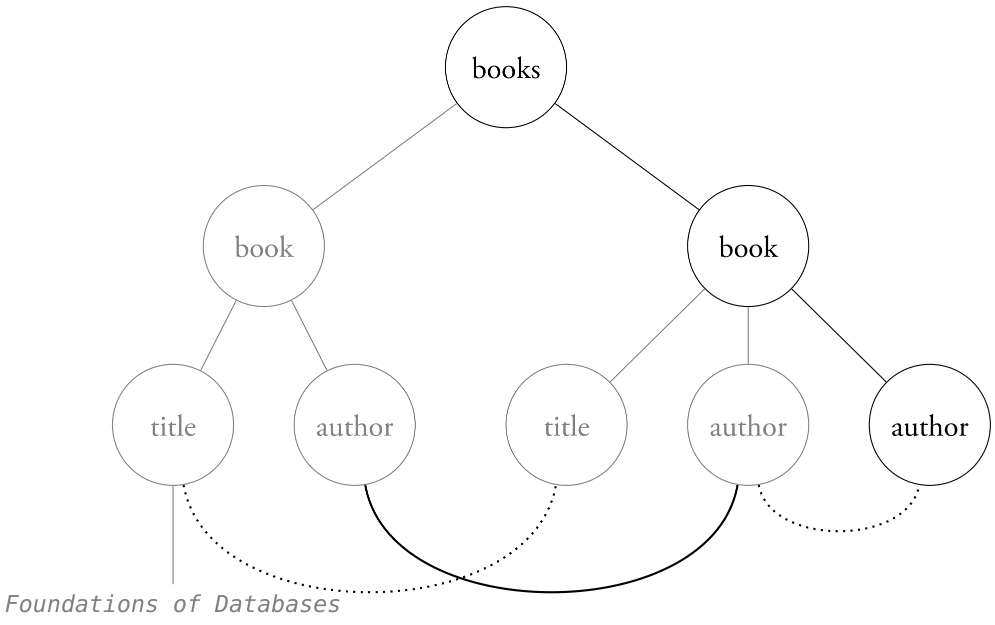


Figure 2.14: TPWJ query corresponding to the path/join expression of Figure 2.13

```
<!ELEMENT books (book*)>
<!ELEMENT book (title, author+)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT title (#PCDATA)>
```

Figure 2.15: DTD for the query of Figures 2.13 and 2.14

Join expressions are specified using '`$n`' variables attached to nodes of the path. The '-' sign preceding some element names is an indication that these nodes should be filtered out from the final query results (but they are of course used during the query evaluation). Corresponding subtrees are shown in gray in Figure 2.14.

The query is intended to be used in an XML document respecting the DTD of Figure 2.15.

The query can now be easily translated in natural language: It returns authors who are (strict) coauthor, with one of the author of *Foundations of Databases*, of a book different from this one. This example illustrates the expressiveness of TPWJ queries, along with the way they can be entered in FuzzyXML.

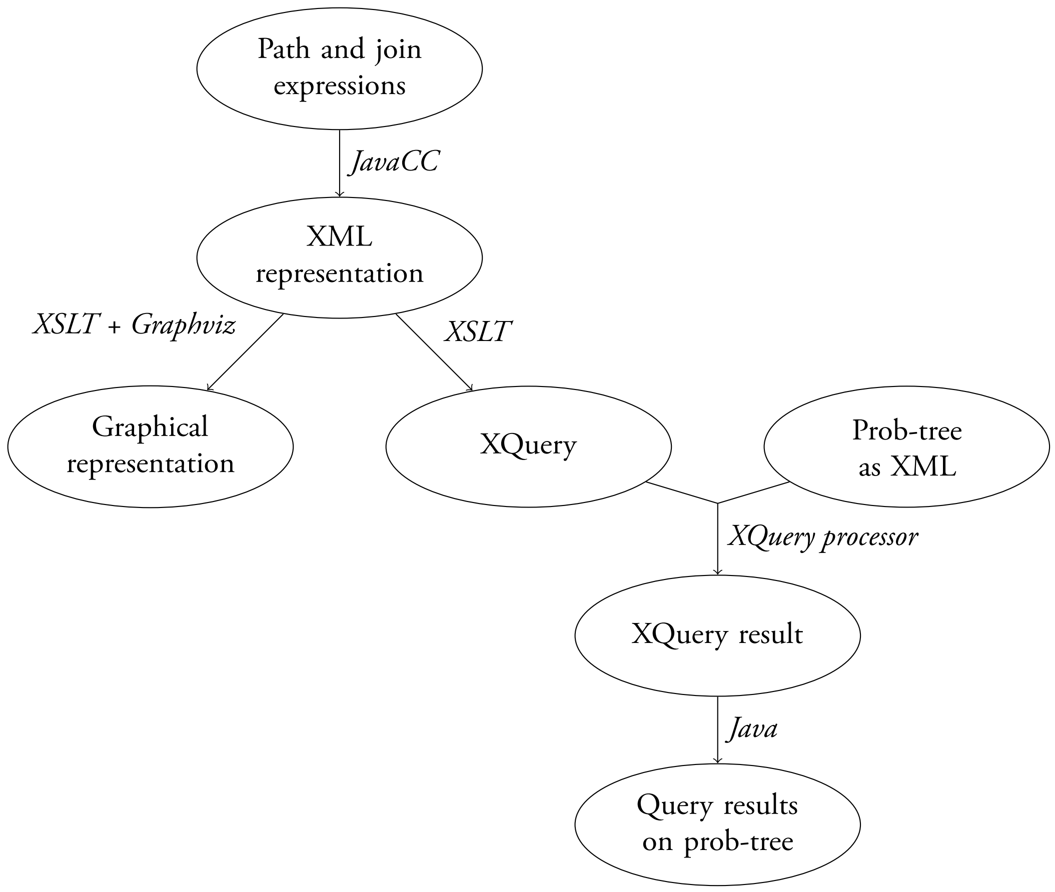


Figure 2.16: Different steps of query processing in FuzzyXML, with technologies used

Figure 2.16 represents the different steps that are used in FuzzyXML to process a query, along with the technologies involved. Path and join expressions are first parsed into an XML representation with the help of JavaCC [Jav]; these queries can be displayed to the user in a tree-like representation similar to that of Figure 2.14 by transforming this XML representation with XSLT into a Graphviz [ATT] graph description. FuzzyXML compiles TPWJ queries on prob-trees into XQuery [W3C07] programs. This compilation is performed in XSLT. The XQuery programs are then evaluated by a standard XQuery processor (currently, the Qizx/open Java engine [AXY]); a post-processing step is performed on the resulting subtrees to compute the associated probabilities and return the final results to the user.

Note that the compilation from TPWJ queries into XQuery is not entirely straightforward, mostly

```
xquery version '1.0';
declare namespace fxml="http://pierre.senellart.com/software/fuzzyxml/";

declare function local:minimal-tree
  ($root as node(),$set as node()*,$ancestors as node()*,$nodrops as node()*)
  as node()
{
  return element {name($root)} {
    $root/@fxml:id,
    for $nodrop in $nodrops[@fxml:id=$root/@fxml:id]
      where exists($nodrop intersect $set)
      return attribute fxml:nodrop { 1 },
    $root/fxml:*,
    let $children:=$root/* intersect $ancestors
      for $n in $children
      return local:minimal-tree($n,$set,$ancestors,$nodrops)
  }
}

return element fxml:result {
  let $books:=doc($document)/books
  return (
    for $book1 in $books/book
    for $book2 in $books/book
    for $title1 in $book1/title[fxml:value="Foundations␣of␣Databases"]
    for $author1 in $book1/author
    for $author21 in $book2/author
    for $title2 in $book2/title
    for $author22 in $book2/author
    where $author1/fxml:value=$author21/fxml:value and
          not($title1/fxml:value=$title2/fxml:value) and
          not($author21/fxml:value=$author22/fxml:value)
    return
      let $set:=$books union $book1 union $title1 union $author1 union
                $book2 union $author21 union $title2 union $author22
      return local:minimal-tree(
        $books,
        $set,
        $set/ancestor-or-self::*,
        $books/@fxml:id union $book2/@fxml:id union $author22/@fxml:id)
   )
}
```

Figure 2.17: XQuery compilation of the query from Figure 2.14

since we need to return sub-datatrees of the original tree, something that is not directly supported by XQuery. To illustrate, we show in Figure 2.17 the XQuery program that results from the compilation of the TPWJ query from Figure 2.14. Some simplifications and reformatting have been made in order to improve legibility. The tree-pattern query is expressed in lines 23–29 with XPath expressions for each different node in the tree, while joins are expressed in lines 30–32 with a **`where`** condition. The function `local:minimal-tree` (lines 4–18) recursively builds the minimal tree rooted at `$root` containing the nodes in the `$set` variable (that is, all nodes matched by the query). A `fxml:nodrop` attribute is added to nodes that are not filtered out in the final results (for technical reasons, this is easier than to mark out nodes to be filtered out).

Similarly, updates are compiled into XUpdate [XML00] programs, evaluated by a standard XUpdate processor (currently, an *ad hoc* processor written in XSLT). A limitation of the XUpdate language (the fact that it does not support cloning of nodes) makes it necessary in some cases (namely, when nodes which are not leafs have to be conditionally deleted) to perform additional XQuery queries on the tree. A way around this limitation would be to use a proper language for expressing updates in XML databases, like the XQuery Update Facility [W3Cb], but such a language is not yet either finalized or widely implemented.

The advantage of using standard languages (XQuery, XUpdate) for compiling queries and updates on prob-trees is the possibility to use any engine supporting these language, which includes XML native repositories such as eXist [eXi].

Observe that we need to record probabilistic meta-information in the XML documents. This is achieved by adding an element containing event conditions on every conditioned node, and maintaining external tables with the association between event names and probabilities.

To conclude this section, we briefly discuss the limitations of our work when exposed to real-life XML documents:

- The trees we consider are unordered, which is absolutely inherent to the approach (see Section 2.6.4 for a brief discussion on changes of the semantics to support ordered trees). We have to assume that the applications we support are only concerned with queries that do not rely on the order of siblings. This is a rather standard assumption.

- Mixed content (nodes with both children and textual content) is not yet supported. This can be easily fixed by adding virtual text nodes above every text fragment.

- Attribute nodes are not yet supported either. They can easily be replaced by standard nodes.

Conclusion

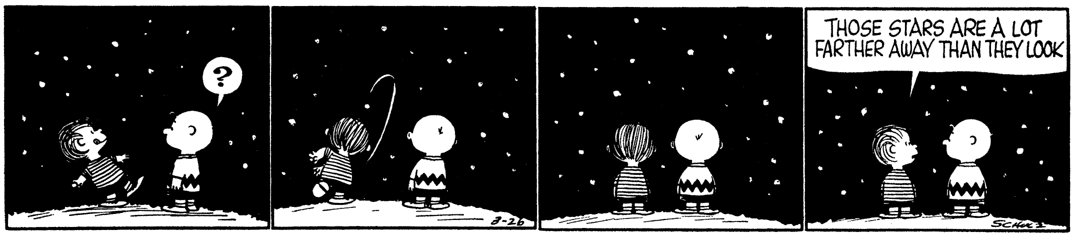
The prob-tree model is a natural, concise and powerful way to represent probabilistic XML information. Queries, and especially updates, are supported in an efficient way, with the exception of deletions, which arise more rarely in our context of a process that gather imprecise information. Complexity results have been established, and an implementation exists.

The present work may be pursued in a number of directions. A first one is prob-tree simplification. One would often like to approximate a prob-tree to get a more compact representation, perhaps ignoring less probable worlds and some of the probabilistic events (some of the provenance/history). Also, probabilities can be used to rank results. It would be useful to have algorithms obtaining the most probable results first. Finally, it would be interesting to also handle aggregate functions. We believe the use of multi-sets simplifies this last issue.

Going back to our original motivation of understanding the hidden Web, we now have a model that can be used for the probabilistic data warehouse described in Chapter 1. We now proceed, in the following chapters, to the issue of gathering (imprecise) information on the hidden Web.

Chapter 3

Probing the Hidden Web



This work has been carried out in collaboration with Avin Mittal, from the Indian Institute of Technology in Bombay, during his three-month internship under my supervision. A more detailed and technical presentation can be found in Avin's report [Mit07]. The content of this chapter and the following one, along with additional experiments, is also presented in [11].

Although some of the sources of the hidden Web are Web services [W3Ca] that are described by a WSDL [W3C01] file, these form a very small minority. Most of the services that can be found on the hidden Web are accessible through an HTML form interface, and their results are shown as HTML result pages. A first step to the understanding and indexing of these services is to understand the structure of both these form interfaces and their result pages. This chapter focuses on the former, while Chapter 4 will deal with the latter. As we shall see, the two problems are not completely independent and we shall mention some interactions between them along the way.

As discussed in Chapter 1, we rely on some domain knowledge to analyze the structure of HTML forms; we use here for this domain knowledge a list of domain concepts, and words appearing in instances of each concept along with their relative frequency. Domain concepts are compared to words that appear around a field of an HTML form to annotate fields, while domain instances are used to probe the form, in order both to obtain some feedback about our annotation, and to have a first hint about the structure of result pages.

Given the URL of an HTML form, the aim of the work presented in this chapter is to annotate each relevant field of this form with the domain concept this fields maps to, to confirm these annotations by probing the form with domain instances, and to generate the result pages that will be analyzed in the next chapter. As in the remaining of this thesis, we take the example of the publication database domain, but our approach is quite general and can be applied to any other domain for which we have sufficient domain knowledge.

We make a number of simplifying assumptions on the structure and semantics of the forms:

- We assume that each field that maps to a domain concept must represent this concept as a whole. This means that we do not manage compound concepts that are queried by their components (a name, with a field for inputting the first name, and another one for the last name). We do not consider range queries, that are common, e.g., for dates.

- We assume that the result to a form represents the conjunctive query of each filled-in field/value pair. In particular, we do not consider complex search queries based on boolean operators.
- We assume that concepts whose instances have multiple words can be queried with a single word occurring in them. In other words, forms support *partial match* queries, instead of *complete match* queries. For instance, a field for inputting the title of an article may be queried with a single word from the title; we actually never found in our experiments any form where this assumption failed to hold.
- We assume that there are no specifically required field in the form, and that each field can be either filled in or omitted. This is actually quite a strong assumption, and it would be a very interesting extension to use probes to distinguish between required and optional fields. This might be feasible in the following way: If we are sure that some tuple is in the underlying database (because it has been outputted as the result of a previous query), we can try to probe the form with all subsets of the attribute of this tuple to check whenever we get a result page. When we do not get a result page, we know that some required field has not been filled in (see Section 3.4 for techniques to distinguish between result and error pages). An alternative approach would be to make use of the potential JavaScript code that is executed when the form is submitted, and to use techniques from programming language semantics or software testing to check if the form submission is rejected by the script because of a missing value. This looks quite promising, but we unfortunately discovered that such JavaScript validation code was quite rare.

In spite of these assumptions, we shall see in Section 3.5 that we still get satisfactory results on most HTML forms that we found in our experiment. Overcoming each of these limitations is a challenging and interesting problem that should be addressed in future work.

The outline of this chapter is the following. We first discuss related work, focusing on understanding form structure and form probing, in Section 3.4, before presenting the general architecture of our form prober in Section 3.2. We then detail how the first part of the structural analysis is performed in Section 3.3. In Section 3.4, we present the probing step itself that is used to confirm annotations discovered in the previous step. Experiments are then discussed in Section 3.5. Finally, we show how we can use all of this to wrap an HTML form as a Web service, in order to abstract away the interface of the service, in Section 3.6.

3.1 Related Work

We present here related work about the analysis of the structure of forms, and probing of these forms; see Section 1.1 for more general works about the hidden Web.

An early work on crawling the hidden Web is [RGM01], where Raghavan and Garcia-Molina present a system that focuses on analyzing Web forms, automatically generating queries and extracting information from the response pages thus obtained. There are many similarities between the approach described in Section 3.3 and [RGM01], though the authors choose to use a method based on the visual layout of elements to determine the label of a field, rather than the more structural method that we use.

The MetaQuerier system processes multi-attribute forms [ZHC05] and uses predicate mapping to convert user-given queries to specific form queries, and thus fetch the required response pages hidden behind the form interface. The focus of this work is on schema mapping and query rewriting to translate queries to a given interface. The paper does not address the analysis of forms themselves, but the same authors describe in [ZHC04] a fairly elaborate approach, based on the notion of *hidden grammars* that describe the relation between structure of a query form and its visual layout.

[IG02] focus on the sampling of a source of the hidden Web, by using domain knowledge to obtain a representative subset of result documents.

[BF04] is another example of a system which extracts hidden-Web data from keyword based interfaces by querying the interface with high coverage keywords. The aim of the authors is to extract all the data from the database, and index it locally, which is quite a different goal. They exploit the same idea as we do for distinguishing between result and error pages, citing [DEW97] as their inspiration, that is, probing the form with *nonsense* keywords that we are sure do not exist in the database.

Finally, the idea of using clustering for distinguishing between result and error pages comes from [CLB04], although we do not use the same input for the clustering algorithm. In [CLB04], the authors construct the feature vector for a page by extracting the tags from HTML code and use the cosine similarity measure with a tf-idf weighting. In practice (see Section 3.5), we found out that this tag-signature–based clustering does not work very well in comparison to our scheme of clustering based on the terminal paths in the DOM tree, that is presented in Section 3.4.

The work presented here has the following particularities with respect to previous work:

- We present a complete system that integrates form syntactic analysis, probing, and wrapping the form into a Web service, exploiting and combining various ideas of the literature.
- We propose a novel way of clustering result and error pages, based on terminal paths in the DOM tree, that performs very well in practice.
- We stress the independence of our hidden-Web prober from the considered domain; the knowledge domain, as a list of concepts and a list of instances of these concepts with their frequency, is external to the system and is the only element to change to handle a different domain.

3.2 Architecture of a Hidden-Web Prober

Figure 3.1 depicts the general architecture of our hidden-Web prober. The different modules of the system, that will be described in the following sections, are shown as rectangular boxes, while external data and agents are represented as ellipses. The input to such a system is the URL of a form of the considered domain, which might be automatically generated using one of the techniques presented in Section 1.4.1. Its output is a Web service, along with its WSDL description, that wraps the original form and that the user can use to perform queries independently of the interface of the service (result pages are still raw HTML pages, see Chapter 4 for extracting information from these).

External components of our system include domain knowledge (concepts and instances), a general ontology that is used to broaden the set of concept names (we used WordNet [Pri] in the experiments, as we dealt with forms in the English language; obviously, this ontology has to be changed for handling a different language), and the World Wide Web itself (more precisely, the submission page for the form, with possibly pages linked from it).

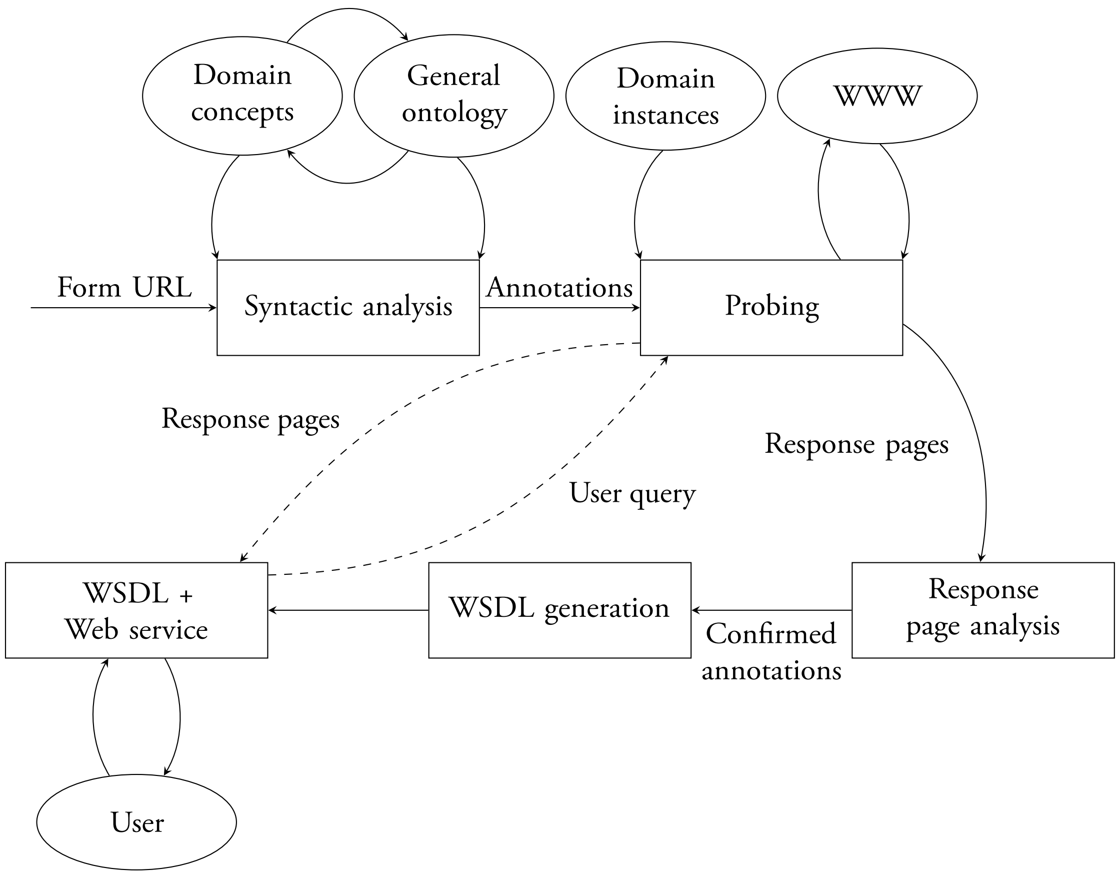


Figure 3.1: Architecture of a system for probing the hidden Web

Four different modules build up the main part of our system, while a fifth one is used to handle Web-service requests from the user. The *syntactic analyzer* processes the HTML code of the form, extracts the relevant information, and adds initial annotations; it is described in Section 3.3. The *probing* and *response page analyzer*, detailed in Section 3.4, probe the form with knowledge domain, so as to confirm or infirm these annotations. Finally, the resulting analyzed form is wrapped as a Web service as described in Section 3.6. At each step, information acquired from previous modules is kept as an XML description that is stored at the end of the process to be used when a query is performed.

3.3 Structural Analysis of an HTML Form

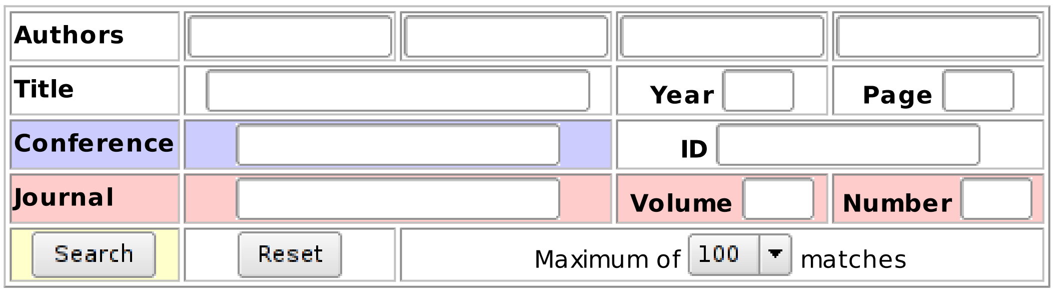


Figure 3.2: Example form

The role of the first module of our prober is to analyze the structure of the form and to find fields that are relevant to the domain concepts. Consider a form like the one that is shown in Figure 3.2. We describe in Algorithm 3.1 the different steps that are carried out, and detail them below.

Algorithm 3.1 Structural analysis of an HTML form

INPUT: URL of a form.
OUTPUT: Meta-information about the form, list of fields with probabilistic annotations.

(a) Retrieve the Web page at the given URL.
(b) Identify the different forms on the page; store meta-information (submission URL, method, encoding).
(c) For each form field:
  - (i) Gather all words of the textual context: `name` and `id` attributes, content of a corresponding `label` element, words appearing before the field in source code order. Words are annotated with a confidence value that depends on their origin.
  - (ii) Remove stop-words.
  - (iii) Stem all context words with Porter's stemming algorithm [Por80].
  - (iv) Check whether any resulting stemmed word matches a stem of words related to the concept name as given by WordNet [Pri]. The final confidence depends on the distance of the concept name to the related word.
  - (v) Annotate the fields with matching concepts, with associated confidence as the probability that this field represent this concept.

(d) Return all gathered data.

Note that the proper (semantic) way to give the label of a form field is the use of the `label`

HTML element, whose `for` attribute must correspond to the `id` attribute of a form field [W3C99]. Unfortunately, this tag is rarely used by Web developers, despite its accessibility virtues, and the fact that graphical browsers use it to allow the user to click on a field label rather than on the field itself with the same effects. We have to resort to other contextual information, such as the `name` (which identifies a field in the submission process) or `id` (which identifies a field for CSS styling or JavaScript processing) attributes, or the text appearing (in the source code) before the field. An alternative here is to use the graphical layout of the form, as in [RGM01, ZHC04], and rules that indicate where the label of a field most often lies relatively to the field.

Some standard preprocessing (stop-word removal, stemming) is then applied to words of the context, before comparing them to words related to concept names. Related words are extracted from WordNet [Pri] by following hyponymy, hyperonymy and synonymy relations (our domain-specific ontology can also be used at this point). Matches between words of the context of a field and words related to a concept name correspond to an annotation of the field with the concept name, subjected to some confidence that is computed from the origin of concept words and related words. Unfortunately, the confidence values are chosen in a quite *ad hoc* way, and the resulting values that are used as probabilities that the field represents the concept are not really well-founded. Some statistical analysis of large corpora of forms may be needed to get proper probability values.

3.4 Probing a Form

Once fields have been assigned probabilistic annotations of concepts, these annotations are confirmed using a probing of the form. Specifically, we compare what happens when we probe a field that has been annotated as concept *c* with instances of *c*, chosen representatively of the frequency distribution of instances of *c*. If the result pages that we obtain are significantly different from result pages obtained by probing the field with nonsense words (e.g., `dsqdqkhzezezaui`), we may assume that the annotation is indeed correct.

One of the important aspects of this module is to be able to distinguish between match and no-match pages (or in other words, result and error pages) resulting from the submission of the form. No-match pages indicate that some required field was not provided, that the input value is incorrect for the corresponding field, or that there are no records even though the input was correct. The distinction between match and no-match pages can be made using a number of heuristics (the size of the no-match page is smaller, there are less outgoing links, we can use the presence of keywords like “Error” or “No match”, absence of keywords like “Next” or “More”). We choose to use a much more robust approach, by performing a clustering of result and error pages. If a page is in a different cluster than an error page obtained with the submission of a nonsense word, this page is probably a result page.

We use a standard clustering approach (namely, an incremental clustering algorithm, that works well in our context, when we have a small number of documents to cluster, with significant differences between error and result pages), with a feature vector built as follows. We consider the DOM tree of an HTML document (an example of which is depicted in Figure 3.3). *Terminal paths* of the DOM tree are the set of paths from the root to a leaf of the tree. Each distinct sequence of node labels (that is, HTML element names) along a terminal path forms a dimension of the vector space that we use for clustering. Each page is then represented in this vector space, with a tf-idf (term frequency-inverse document frequency) weighting, depending on which terminal paths are present in its DOM tree. Finally, the cosine similarity measure is used to compare two vectors during clustering. The idea is that two result pages share most of their terminal paths, some of

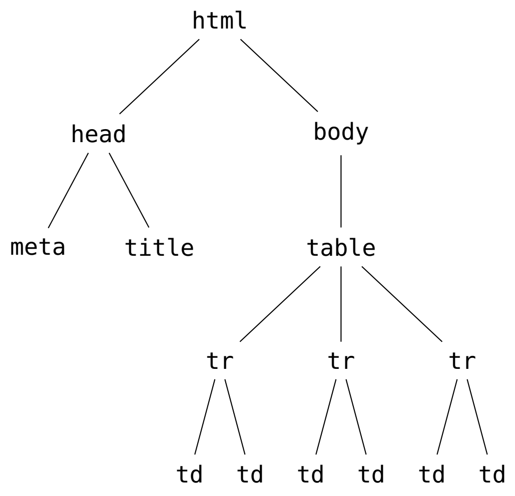


Figure 3.3: Example DOM tree of an HTML page

them may just be repeated more than other; on the other hand, a result page and an error page have significantly different structures (no list of results appears in an error page) that leads to a completely different representation in the vector space of terminal paths in the DOM tree.

We summarize the algorithm to confirm annotations of the previous module in Algorithm 3.2. An additional step that is also performed during the analysis of result pages is to look for "Next" hyperlinks that point to pages with subsequent results. This is currently done in a purely heuristic way, by looking for links that contain "Next" or similar keywords. Note also that we may get a clustering of more than two clusters, if there are significant differences between two result pages. This is the case, for instance, in the publication database DBLP, where searching for an ambiguous author name results in a different page than searching for a name that only appears once in the database. It is then important to use multiple words for probing, representative of their frequency distribution, so as (i) to generate all possible kinds of result pages; (ii) to be sure to get a result page, as long as the service probed has similar content as our domain knowledge.

Algorithm 3.2 Confirmation of field annotation with probing

```
INPUT: A given field of a form, with its probabilistic annotations.
OUTPUT: A confirmed annotation for this field, or none at all.
(a) First, identify an error page by probing the field with a nonsense word.
(b) Let $c$ be the concept of highest probability a field is annotated with.
(c) Probe the fields with a set of instance words, randomly chosen according to their frequency
    distribution, to get a corresponding set of pages.
(d) Cluster the set of pages obtained by probing with the error page, using an incremental clustering
    algorithm on terminal paths in the DOM tree of the pages.
(e) If some pages obtained by probing are different from the result page, confirm the annotation;
    otherwise, retry with the next best concept annotation.
(f) Look for "Next" links on result pages, and store the corresponding information.
```

3.5 Experimental Results

We describe here experiments that have been carried out, using a Java implementation of our system for probing the hidden Web. We look at the domain of publication databases and, as mentioned in Chapter 1, the knowledge domain comes from DBLP. We consider the following concrete concepts for annotating form fields:

- `Title`
- `Author`
- `Journal`
- `Conference`

Note that we are not looking for fields representing the `Date` concept, since those are mostly present as range-query fields, that we do not support yet.

We present in Table 3.1 some statistics about the performance of the syntactic analyzer and the probing module on a list of 14 URLs of complex publication database search forms. The number of text fields is shown for each form, with the number of fields that have been annotated with some concept by the syntactic analyzer, and the number of confirmed annotations by the probing module. In addition, we processed each of these forms by hand to find all fields that were relevant to some of our concepts. Precision and recall of the confirmed annotations with respect to the human annotations are shown in the last two columns of Table 3.1. A precision of 100 % means that all confirmed annotations are correct, while a recall of 100 % means that all fields relevant to domain concepts have been correctly annotated. Note that we do not consider here either single-field forms, which are mostly keyword-based search forms, nor forms that allow complex boolean combinations of the various fields, such as `http://scholar.lib.vt.edu:8765/index.html?ql=a`.

The first observation is that, despite the various assumptions that we made on the fields of a form, we still get quite good results; in particular, the average precision for our dataset is 90 %, while the average recall is 77 %. The other observation that can be made is that the probing and confirmation step is indeed useful, since it removes a large number of incorrect annotations. Improving the precision should perhaps be easier than improving recall: An idea is to be more cautious and less tolerant during the probing step, only probing with words that are unambiguously attached to a given concept, while requiring that most probes return result pages. This might, however, reduce quite a lot the coverage. Improving the recall may be quite hard, in the situation where the textual context of a field is not descriptive enough to get an annotation. We may try, however, in these cases, to probe a field with each concept in turn; as the number of fields and concepts are small enough, this seems feasible. Note finally that the time required for all this processing is essentially the network access times required for the probes, all other operation taking a negligible time.

As explained in Section 3.4, the feature vector that we use for clustering is the set of terminal paths in the DOM tree of the document, with tf-idf weighting. The DOM tree captures the structure of the document perfectly, and works particularly well for our experiments. For instance, the cosine similarities between the result pages from Google Scholar (`http://scholar.google.com/advanced_scholar_search`) are up at around 0.99, whereas the similarities between result and error pages are of the order of 0.01. To show that the DOM tree model is an adequate choice, we also experimented with a feature vector based simply on the occurrence of HTML tags in the document [CLB04]. We simply consider all tags that occur in the document, compute the tf-idf score based on the occurrence of tags in the collection and use the cosine similarity between these

Table 3.1: Experimental results of our probing module

URL	Text fields	Annotated	Confirmed	Recall (%)	Precision (%)
`http://www.informatik.uni-trier.de/~ley/db/indices/query.html`	12	8	7	100	100
`http://pubs.er.usgs.gov/usgspubs/index.jsp?view=adv`	5	3	3	100	100
`http://nrelpubs.nrel.gov/Webtop/ws/nich/www/public/SearchForm`	6	5	3	100	67
`http://eprints.aktors.org/perl/search/advanced`	9	6	2	67	100
`http://eprints.unifi.it/perl/search/advanced`	9	6	2	67	100
`http://caltechlib.library.caltech.edu/perl/search/advanced`	9	5	2	100	100
`http://dlist.sir.arizona.edu/perl/search/advanced`	9	6	3	67	67
`http://www.diva-portal.se/`	7	2	1	33	100
`http://eprints.rclis.org/perl/search/advanced`	11	6	2	67	100
`http://highwire.stanford.edu/cgi/search`	7	4	3	100	67
`http://www.ingentaconnect.com/search/advanced`	5	4	2	67	100
`http://eprints.cs.vt.edu/perl/search/advanced`	9	6	2	67	100
`http://scholar.google.com/advanced_scholar_search`	8	5	4	100	50
`http://archives.cs.iastate.edu/perl/advsearch`	11	5	2	50	100

vectors for clustering. It was found that this approach assigns a rather high degree of similarity between result and error pages (for Google Scholar, it was of the order of 0.5 to 0.6, for instance, which is rather high, and makes the clustering process very dependent of the considered threshold). The possible reason for this is that almost all HTML tags appear in all pages and hence the idf measure is wasted. Also, length normalization leads to the increase in similarity of the vectors between result and error pages, since the structural details such as paths, ordering of tags, etc., are lost.

3.6 Wrapping a Form as a Web Service

Our probing module can be used to provide a user with an abstract Web-service [W3Ca] interface to a given query form of the hidden Web. From our annotation of the fields of a form, we can derive a WSDL [W3C01] description of an abstract service, as well as a Web-service interface to it. Thus a user can query such services as Google Scholar or DBLP with the same interface. The user also indicates how many result pages he is interested in, in the case when we are able to follow "Next" links. This module has also been implemented as a Java Web application. Obviously, such a module is all the more valuable as it is integrated with the extraction of results from result pages, which may be presented to the user in an abstract way; we discuss this in the next chapter.

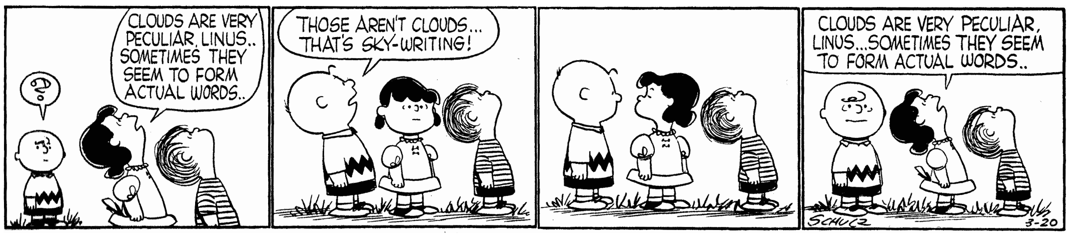
Conclusion

We presented in this chapter a system that is able to analyze the structure of a query form of the hidden Web, using probes of the form to confirm annotations of form fields with domain concepts. Result and error pages are distinguished with the use of a clustering according to the set of terminal paths in the DOM tree of the page. Without much sophistication, the system is able to understand the structure of forms of publication database services with high precision; this annotation of forms is leveraged in the presentation to the user of a source-independent interface to services of the hidden Web.

Many refinements could be applied, in particular in order to remove some of the restrictions imposed on the query forms. One of the most important improvement would be to detect in an automatic way optional vs. compulsory fields. Such a system is also to be thought in a more general context, along with a system for discovering sources (see Section 1.4.1) and a system for extracting information from result pages (see next chapter).

Chapter 4

Extracting Result Page Data


CLOUDS ARE VERY PECULIAR, LINUS.. SOMETIMES THEY SEEM TO FORM ACTUAL WORDS..
THOSE AREN'T CLOUDS... THAT'S SKY-WRITING!
CLOUDS ARE VERY PECULIAR, LINUS...SOMETIMES THEY SEEM TO FORM ACTUAL WORDS..
SCHULZ
3-20

This work has been carried out in collaboration with researchers from the INRIA Mostrare group in Lille, and has been in particular at the center of Daniel Muschick's master's thesis [Mus07]. We discuss this work briefly, and refer the reader to [Mus07] for more details. The content of this chapter and the preceding one, along with additional experiments, is also presented in [11].

The understanding of the structure of a form is not sufficient for being able to use it automatically in applications. The structure of the pages that result from the submission of a form has also to be understood. In other words, *wrappers* to extract data from result pages have to be built. These pages are typically in HTML and present one or more query results. They have been generated by some software and are organized according to some structure (e.g., list or table) that we have to discover. We describe in this chapter an original approach that relies on a supervised machine learning technique that is to build a wrapper starting from an imprecise and imperfect annotation by domain knowledge. More precisely, we first linearly browse result pages, annotating them with domain instances. This annotation is then used to train a Conditional Random Fields (CRF) wrapper, whose features only consider the structure of the page. This wrapper can then be used on other result pages that exhibit the same structure. The addition of a bootstrap loop to this process, either in the construction of a wrapper for a single source or across sources, is also considered.

We first discuss related work in Section 4.1. In particular, we introduce the machine learning framework that we use, Conditional Random Fields for XML (XCRF). We then describe our method for combining annotation with domain knowledge and supervised machine learning to induce wrappers from result pages in Section 4.2. Experimental results are presented in Section 4.3. Finally, we revisit in Section 4.4 our discussion of Section 3.6 on how to wrap a form as a Web service, now that we have a way to understand the structure of result pages.

4.1 Related Work

The task that we consider in this chapter is an *information extraction* task. We aim to extract structured data from HTML pages. A survey of information extraction methods on the Web is presented in [CKGS06]. Most techniques rely on some form of human supervision, with the exceptions of [CMM01] and [AGM03] that are purely structural information extraction techniques performing unsupervised machine learning on unannotated documents (in this case, a few result pages). ExAlg [AGM03] presents in our opinion interesting theoretical advantages over RoadRunner [CMM01] (in particular, for its handling of disjunction), but the latter is freely available and can directly be tested. They both try to discover *patterns* representing the original skeleton structure. As such approaches are purely structural, an additional step has to be performed in order to label extracted data. As a follow-up of RoadRunner, Arlotta et al. [ACMM03] propose to exploit spatial relationship between labels and data in the page to discover these labels. Naturally, this only works if there are some labels in the result page, which is not always the case, especially with bibliographical data, often formatted using domain conventions.

Two aspects of our work have some similarities with papers about the MetaQuerier system. In [WDYM05], bootstrapping a knowledge base by extracting information on result pages and injecting them back on subsequent probing is discussed. Bootstrapping across different sources, in order to benefit from their different coverage, is the topic of [CCZ07]. Our main idea of using supervised techniques on the structure of a document to generalize an annotation by a gazetteer has not been explored in either of these works.

The part of the result pages containing relevant information can be roughly identified as the part of the page that changes between two different queries. Note, however, that there may also be content (advertisements, hints, etc.) that changes between different runs of the same queries; such content has to be filtered out). [CLB04] proposes an elaborate method to identify these *pagelets*, based on the similarity between subtrees of the DOM tree of the page. As we shall see, we use instead a simpler approach based on the extraction of the least common ancestor of annotated nodes.

We briefly present now Conditional Random Fields for XML (XCRF), a supervised machine learning framework for labeling tree data, proposed in the INRIA Mostrare group. We have not participated in the development of this model, but worked in collaboration with Mostrare to use it in the context of the hidden Web. Conditional Random Fields (CRF) have been introduced by Lafferty et al. in [LMP01] as a way to represent conditional probabilities in a graphical model. They have been much used in various labeling tasks; a recent overview of related work is given in [SM07]. The XCRF model [JGTT06] applies CRF on XML tree structures, with both ordered (elements) and unordered (attributes) nodes. The graphical structure defined by XCRF (recall that CRF use probability distributions over a graphical model) are the 3-cliques formed by a node and two adjacent children of it (for ordered portions of the trees), and parent-child edges (for unordered portions of the trees). This takes into account both parent/child and sibling relationships. Given a set of features (boolean functions defined by an XPath expression) that represent the *observables* in a tree, an XCRF wrapper is trained on labeled examples to select the features that best describe the annotation for each node of the tree, allowing interdependencies between the annotations inside each aforementioned clique. A Java implementation of XCRF is freely available at `http://treecrf.gforge.inria.fr/`.

4.2 Unsupervised Learning with the help of a Gazetteer

We present in this section our method for building a structural wrapper for a set of result pages. The main observation is that result pages for a given source of the hidden Web are built from the same template, and hence, share a common structure (in some occasions, there are several different

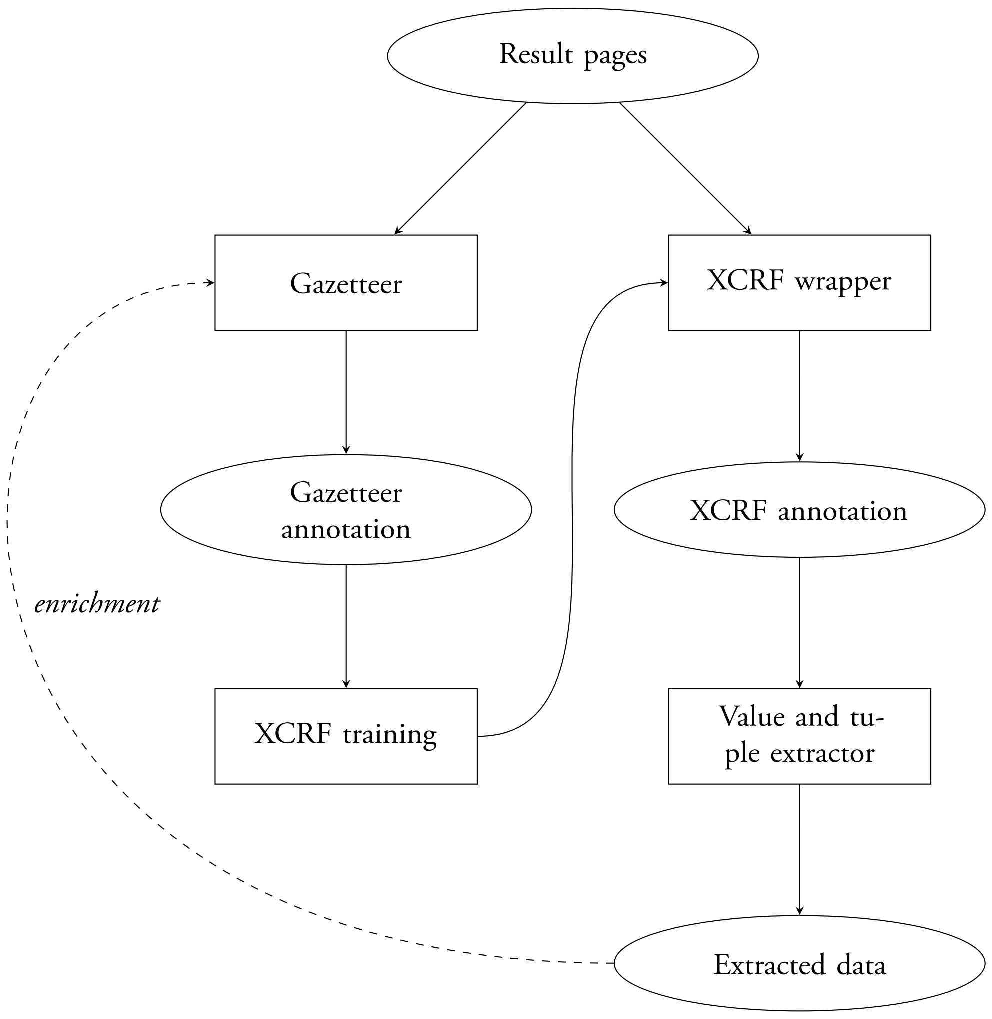


Figure 4.1: Simplified architecture of the information extraction module

Algorithm 4.1 Extraction of data from result pages

```
INPUT: Set of result pages with a common structure.
OUTPUT: Structural wrapper to extract data from this kind of result pages.
(a) Tokenize the result pages so that each word is in a separate token (see Section 4.2.1).
(b) Use the gazetteer to annotate the tokenized result pages (see Section 4.2.2).
(c) Remove outliers, and segment the result pages into records (see Section 4.2.3).
(d) Train an XCRF wrapper on this segmented annotation (see Section 4.2.4).
(e) Optionally, use this XCRF wrapper to extract data from the result pages, use this data to enrich
    the gazetteer, and go back to step (b) (see Section 4.2.5).
```

templates, but we saw in the previous chapter how to distinguish between them). We assume that we are given a set of (HTML) pages with the same structure, relevant to the domain of interest, and we aim at building a wrapper that extracts relevant data from pages with this structure. We present in Figure 4.1 a simplified architecture of our method. In this figure, rectangles represent components, while ellipses represent data. An alternative high-level view of the process is presented in Algorithm 4.1. We detail in the following the different steps of this algorithm.

4.2.1 Tokenization

The information that we want to extract is in substrings of textual nodes in the DOM tree of a page. As the XCRF model is used to label nodes in the tree structure of a document, it is thus necessary to tokenize textual nodes, typically at the word level. This is done as a preprocessing step, and the same tokenization is used by the gazetteer as described next.

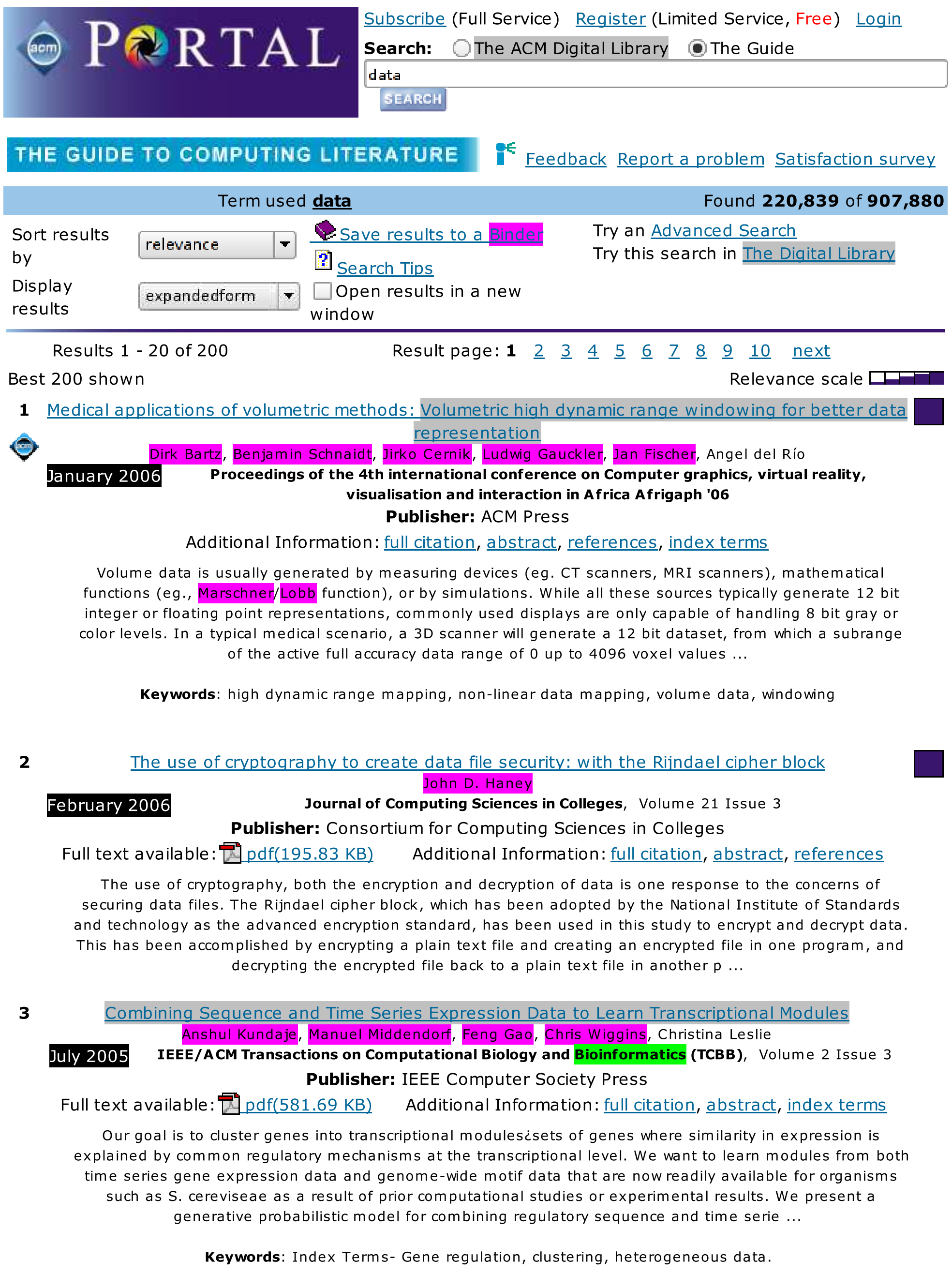
4.2.2 Gazetteer

Following [SO06], we use the term *gazetteer* for a dictionary of named entities that is used to annotate documents in information extraction tasks. Recall from Chapter 1 that one of the components of our domain knowledge is a probabilistic model for string representations of instances of concrete concepts. This probabilistic model can be seen as a gazetteer and used to annotate pages in the following way. We browse each textual node of the DOM representation of the HTML document, and whenever we recognize a substring as an instance of some concept with probability p, we add the corresponding annotation. This is done in a *greedy* way: leftmost longest string matches are preferred. In case multiple concepts match a string, all annotations are added with corresponding probability values.

Figure 4.2 is an example of the result of this annotation on a result page from the ACM Digital Library (`http://portal.acm.org/dl.cfm`). Recognized instances are highlighted with various colors (for instance, in white over black for dates). As can be seen, the annotation is neither perfect (e.g., *`Marschner`* and *`Lobb`*, that occur in an abstract, are recognized as author names), nor complete (*`Angel del Rio`* is not recognized as an author name). We assume that the annotation, however, is good enough so that its generalization will in general correct such errors. Note that this is highly dependent on the quality of the domain knowledge, and of its adequacy with respect to the considered source.

4.2.3 Segmentation

There is a number of issues with the annotation of the gazetteer. First, there may be some *outliers*, labels at unlikely positions (*`Binder`*, identified as an author name in the header of Figure 4.2, is such an outlier). Second, some records may not have any annotation at all, and may hinder the training of the wrapper. A solution to these problems is to perform a *segmentation* of the pages, to extract each different record (in Figure 4.2, there are thus three different segments). Then, only segments that have enough annotations are kept and used in the training phase, and annotations outside these segments are considered as outliers. A simple approach is used to obtain this segmentation: just find the most common nodes that are *least common ancestors* of annotated nodes. Details are given in [Mus07].



Subscribe (Full Service) Register (Limited Service, Free) Login

Search: The ACM Digital Library [✓ The Guide]

data

SEARCH

THE GUIDE TO COMPUTING LITERATURE

Feedback Report a problem Satisfaction survey

Term used **data** Found **220,839** of **907,880**

Sort results by: relevance

Display results: expandedform

Save results to a Binder

Search Tips

Open results in a new window

Try an Advanced Search

Try this search in The Digital Library

Results 1 - 20 of 200 Result page: **1** 2 3 4 5 6 7 8 9 10 next

Best 200 shown Relevance scale

1 Medical applications of volumetric methods: Volumetric high dynamic range windowing for better data representation

Dirk Bartz, Benjamin Schnaidt, Jirko Cernik, Ludwig Gauckler, Jan Fischer, Angel del Río

January 2006 **Proceedings of the 4th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa Afrigaph '06**

Publisher: ACM Press

Additional Information: full citation, abstract, references, index terms

Volume data is usually generated by measuring devices (eg. CT scanners, MRI scanners), mathematical functions (eg., Marschner/Lobb function), or by simulations. While all these sources typically generate 12 bit integer or floating point representations, commonly used displays are only capable of handling 8 bit gray or color levels. In a typical medical scenario, a 3D scanner will generate a 12 bit dataset, from which a subrange of the active full accuracy data range of 0 up to 4096 voxel values ...

Keywords: high dynamic range mapping, non-linear data mapping, volume data, windowing

2 The use of cryptography to create data file security: with the Rijndael cipher block

John D. Haney

February 2006 **Journal of Computing Sciences in Colleges**, Volume 21 Issue 3

Publisher: Consortium for Computing Sciences in Colleges

Full text available: pdf(195.83 KB) Additional Information: full citation, abstract, references

The use of cryptography, both the encryption and decryption of data is one response to the concerns of securing data files. The Rijndael cipher block, which has been adopted by the National Institute of Standards and technology as the advanced encryption standard, has been used in this study to encrypt and decrypt data. This has been accomplished by encrypting a plain text file and creating an encrypted file in one program, and decrypting the encrypted file back to a plain text file in another p ...

3 Combining Sequence and Time Series Expression Data to Learn Transcriptional Modules

Anshul Kundaje, Manuel Middendorf, Feng Gao, Chris Wiggins, Christina Leslie

July 2005 **IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)**, Volume 2 Issue 3

Publisher: IEEE Computer Society Press

Full text available: pdf(581.69 KB) Additional Information: full citation, abstract, index terms

Our goal is to cluster genes into transcriptional modules¿sets of genes where similarity in expression is explained by common regulatory mechanisms at the transcriptional level. We want to learn modules from both time series gene expression data and genome-wide motif data that are now readily available for organisms such as S. cereviseae as a result of prior computational studies or experimental results. We present a generative probabilistic model for combining regulatory sequence and time serie ...

Keywords: Index Terms- Gene regulation, clustering, heterogeneous data.

Figure 4.2: Example result page annotated by the gazetteer

4.2.4 XCRF Training

An XCRF wrapper is then trained from these segments. We shall not describe the algorithm used for inducing the wrapper (see [JGTT06]), but only discuss the features that are used for the training. They are automatically generated from the training data and express notions such as:

- element name of (the ancestors of) the current node;
- attribute values of (the ancestors of) the current node;
- Unicode [Uni06] category of the textual content of each token;
- column index of a `<td>` table cell.

In order to limit the number of such notions that are considered, an additional feature selection step is carried out, that prunes out features with little support.

4.2.5 Enrichment and Bootstrapping Loop

Once an XCRF wrapper has been learned, it can be used to annotate pages with the same structure. Data can be extracted from this annotated page in the following way: consecutive tokens with the same annotation form an extracted value, while values are regrouped into tuples with the same segmentation that was used in Section 4.2.3.

An optional step is to use these extracted, annotated, values to *enrich* the gazetteer (we can just add the corresponding strings to the domain instances). Then, the whole process can be repeated (gazetteer annotation, XCRF training). This can lead to better results depending on the quality of the extracted values. This *bootstrapping* technique can also be used across sources: train an XCRF wrapper on a first source, gather data from this source into the domain knowledge, and use it to improve the performance of the wrapper induction on an another source, and so on.

4.3 Experimental Results

Table 4.1: F-measure (%) of the annotation by the gazetteer and by the XCRF

	ACM		Citeseer		DBLP	
	Gazetteer	XCRF	Gazetteer	XCRF	Gazetteer	XCRF
`Author`	86	94	59	68	92	95
`Conference`	19	58			87	88
`Date`	97	100	78	68	97	95
`Title`	77	100	54	79	94	96

We show in Table 4.1 experimental results on three different sources, to illustrate different behaviors depending on the source: the ACM digital library, Citeseer (`http://citeseer.ist.psu.edu/cs`), and DBLP [Ley]. We did not consider any bootstrapping in this experiment, and we use the F-measure that is defined as $F = 2 \cdot p \cdot r/(p + r)$, to summarize precision p and recall r, with respect to a human annotation. For each source, the first column gives the F-measure of an annotation by the gazetteer, while the second column gives the F-measure of an annotation by an XCRF wrapper that has been trained on 8 to 12 gazetteer-annotated pages with similar structure.

Note first that, with the exception of the concept `Date`, there is no performance degradation in the XCRF annotation. This means that the XCRF wrapper is at least capable of reproducing the gazetteer annotation (this was not guaranteed, since the wrapper does not have access to the domain knowledge). The case of the concept `Date` is special, since the gazetteer already performs very well due to the relative non-ambiguity of dates. The second observation is that, in most cases, the XCRF wrapper performs better than the gazetteer, in some cases very significantly so (this is especially noteworthy for the ACM source, where it sometimes reach a perfect precision and recall). The DBLP source is special in this respect, since our domain knowledge (and hence, the gazetteer) comes from DBLP records. This explains that the gazetteer is already very good at annotating result pages from DBLP, and that the XCRF wrapper does not significantly improve this annotation.

The performance of the XCRF wrapper heavily depends on the structure that is present in the result pages. In the case of the ACM Digital Library, result pages are well-structured, with a number of tags used to separate the different constituents of each record. In Citeseer result pages, however, bibliographic records are essentially text-based. Still, the XCRF wrapper is able to increase both precision and recall of the gazetteer annotation in this source.

We ran various other experiments, especially on the use of bootstrapping to improve the performance of the wrapper. Preliminary results suggest that it is indeed possible to do so, but that special care has to be taken, in order to avoid the process to degrade after a few iterations. Bootstrapping across multiple sources seems a particularly promising direction.

4.4 Wrapping a Form as a Web Service

We discussed in Section 3.6 how to use our probing module to wrap a form as a Web service, whose inputs are domain concepts and whose output is a set of Web pages. We can then use the method for extracting data from result pages that we presented in this chapter to extend this, and wrap a form as a Web service, with both inputs and outputs described as domain concepts, and then have a full abstraction of the original hidden-Web source.

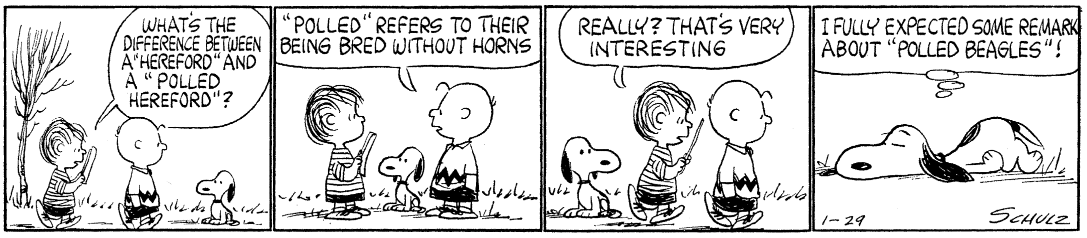
We have the following general process for wrapping a form: given its URL, we first use our probing module to wrap it as a Web service with abstracted inputs. We then use this service and our domain instances to generate a number of result pages. The same techniques as in Chapter 3 can be used to check that they have the same structure. We finally apply the method presented in this chapter to build a wrapper for the result pages, and thereby wrap the whole source as a Web service.

Conclusion

We described an original, unsupervised, approach to information extraction on Web pages resulting from the submission of a form, that uses a supervised structural learning method trained on automatic annotations by domain knowledge. This method improves the quality of the annotation, and can be used in conjunction to the work presented in Chapter 3 to abstract fully away the interface of a source of the hidden Web. We mention a possibility for further improvement by using some single-source or multi-source bootstrapping of domain knowledge.

Chapter 5

Deriving Schema Mappings from Database Instances


WHAT'S THE DIFFERENCE BETWEEN A "HEREFORD" AND A "POLLED HEREFORD"?
"POLLED" REFERS TO THEIR BEING BRED WITHOUT HORNS
REALLY? THAT'S VERY INTERESTING
I FULLY EXPECTED SOME REMARK ABOUT "POLLED BEAGLES"!
1-29
SCHULZ

This work has been carried out while visiting Georg Gottlob at University of Oxford. It is also presented in [10].

Arguably one of the most complex problems when dealing with services of the hidden Web, once they have been analyzed using the techniques of the two previous chapters, is to understand the relations that exist between input and output data. Thus, a service taking as input a person and providing as output another person may be a genealogical service outputting parents of a given individual, a corporation service giving the supervisor of an employee, or even a matchmaking service. This is actually a very general situation: Web services provide relations between objects of the world, and it is the role of the system to figure out which relations these can be. To solve this problem, one should once again use domain knowledge. The problem can then be articulated as deriving semantical relations between database instances, one being our knowledge domain, the other one derived from an unknown source. This particular problem can also be stated in the context of *data exchange*.

Data exchange [FKMP03] is a research area that is closely related to the topic of *data integration* [Len02]: We are given two schemata of relational databases **S** and **T**, along with dependencies Σ (that is, a finite set of formulas in some logical language, expressed in function of the relation symbols of **S** and **T**), and we are interested, for instance, in the set of instances J of **T** which satisfy the dependencies Σ with respect to a given instance I. The main difference with data integration is that we actually want to materialize J, and not only to compute the answers of a query on a virtual J.

Schemata and dependencies, in a data exchange context, form *metadata* that need to be managed in a systematic and formal way. Bernstein argues in [Ber03] for the definition, in such a setting, of *operators* on this metadata. Thus, [FKTP04] and [Fag06, FKPT07] respectively propose ways to define the composition and inverse operators on schema mappings. Another operator of importance is the *match* operator: given two schemata and instances of these schemata, how to derive an appropriate set of dependencies between these schemata. More precisely, given two relational databases schemata **S** and **T** and instances I and J of these schemata, the problem is to find a *schema mapping*, that is, a finite set Σ of formulas in a given language $\mathscr{L}$, such that $(I,J) \models \Sigma$ (or such that (I,J) is *nearly* a model of Σ, in case we assume that I is not perfectly mapped to J). Obviously, $\Sigma = \varnothing$ is always a possible answer; therefore, we want additional constraints on Σ: it must *explains* as many facts of J as possible, and it must be *concise*. We then have a problem of finding the optimal schema mapping with respect to three different parameters: validity, explanation of the target instance, conciseness.

This problem is related to the techniques used for *automatic schema matching* [RB01], but these typically use some kind of meta-information about the schemata (names of concepts and relations, concrete data types, etc.). We consider, in this chapter, an approach that is solely based on the presence and position of constants that appear in source and target instances, without any additional meta-information. Obviously, both kinds of approaches could be combined, in the presence of such meta-information. An example of practical application would be to interact with different sources of information, with different schemata (tabular data, HTML forms with result pages, Web services) on the (hidden) Web. All these sources might contain more or less the same information, and the problem is, by just looking at the data, to find the relations between the different ways of representing the same information; this lies at the semantic analysis step of our process for understanding the hidden Web (see Chapter 1).

We present here a theoretical framework that formalizes the notion of optimality of a schema mapping with respect to a pair of database instances. We present formal definitions and computability and complexity results, both for the general case (say, with $\mathscr{L}$ the language of first-order logic) and for more specific cases (namely, tuple-generating dependencies, full tgds, acyclic tgds, and acyclic full tgds). We are not aware of any work with the same focus on a theoretical and systematic analysis, although, in spirit, the problem that we deal with here is similar to the one that inductive logic programming [LD94] aims to solve.

We first introduce some preliminaries on schema mappings and tuple-generating dependencies in Section 5.1 before introducing in Section 5.2 a notion of *cost* and *optimality* of a schema mapping of tuple-generating dependencies, that we justify in Section 5.3 by noting its behavior on instances that are simply derived from each other. We carry out in Section 5.4 a complexity study of the different decision problems leading to the optimality of a schema mapping. In Section 5.5, we discuss an extension of the definitions to full relational calculus, and alternative cost functions.

5.1 Preliminaries

We assume some countably infinite sets $\mathscr{C}$ of constants (denoted a, b, 0, 1, etc.) and $\mathscr{V}$ of variables (denoted x, y, z, etc.). We use the notation $\mathbf{x}$ to represent a vector of variables $x_1 \dots x_n$. Constants appearing in formulas are here identified, as usual, with the domain elements they are interpreted by.

A (relational) schema is a finite set of pairs (R,n) where R is a relation name and $n \geqslant 1$ the arity of the relation. An instance I of a relational schema $\mathbf{S}$ consists, for every $(R,n) \in \mathbf{S}$, of a *finite* relation over $\mathscr{C}^n$. We occasionally denote R^I the interpretation of the relation name R in the instance I (if $|\mathbf{S}| = 1$, we shall make the confusion R^I=I). In the following, we assume that the schemata are implicitly given whenever we are given an instance.

A *language* $\mathscr{L}$ is a subset of the set of formulas of first-order logic with equality and constants, and without function symbols (with its usual semantics). Given a language $\mathscr{L}$, a *schema mapping* in $\mathscr{L}$ is a finite set of formulas in $\mathscr{L}$. We are particularly interested in the following languages, given instances I, J with schemata $\mathbf{S}$, $\mathbf{T}$:

Relational calculus. $\mathscr{L}_{\mathrm{rc}}$ is the set of first-order formulas without constants, with relations symbols in $\mathbf{S} \cup \mathbf{T}$.

Source-to-target tuple-generating dependencies (tgds). $\mathscr{L}_{\text{tgd}} \subset \mathscr{L}_{\text{rc}}$ is the set of formulas of the form

$$\forall \mathbf{x}\ \varphi(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi(\mathbf{x},\mathbf{y})$$

where:

(i) $\varphi(\mathbf{x})$ is a (possibly empty) conjunction of positive relation atoms, with relation symbols in $\mathbf{S}$;
(ii) $\psi(\mathbf{x},\mathbf{y})$ is a conjunction of positive relation atoms, with relation symbols in $\mathbf{T}$;
(iii) all variables of $\mathbf{x}$ appear in $\varphi(\mathbf{x})$.

Acyclic tgds. $\mathscr{L}_{\text{acyc}} \subset \mathscr{L}_{\text{tgd}}$ is the set of tgds such that the hypergraph of the relations on the left hand-side is acyclic [AHV95, BFM+81], as well as the hypergraph of the relations on the right hand-side, considering only existentially quantified variables. More precisely, let $\forall \mathbf{x}\ \varphi(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi(\mathbf{x},\mathbf{y})$ be a tgd, and let (N,E) (respectively, (N',E')) be the hypergraph whose vertices are the variables of $\mathbf{x}$ (respectively, $\mathbf{y}$) and whose edges are the relation atoms of $\varphi(\mathbf{x})$ (respectively, the relation atoms of $\psi(\mathbf{x},\mathbf{y})$ where at least one variable of $\mathbf{y}$ appears). The tgd is said to be acyclic if there are two forests F and F' (called the *join forests*) with each hyperedge of E (respectively, of E') a node of F (respectively, of F'), such that for all nodes $n \in N$ (respectively, $n \in N'$), the subgraph of F (respectively, F') induced by the edges of E (respectively, E') that contain n is connected. Other equivalent definitions of acyclic hypergraphs are given in [BFM+81].

Full tgds. $\mathscr{L}_{\text{full}} \subset \mathscr{L}_{\text{tgd}}$ is the set of tgds without an existential qualifier on the right-hand side, that is, of the form

$$\forall \mathbf{x}\ \varphi(\mathbf{x}) \rightarrow \psi(\mathbf{x}).$$

Acyclic full tgds. $\mathscr{L}_{\text{facyc}} = \mathscr{L}_{\text{acyc}} \cap \mathscr{L}_{\text{full}}$ is the set of full tgds such that the hypergraph of the relations on the left hand-side is acyclic.

We have the following inclusions between these languages:

$$\begin{tikzcd}
 & \mathscr{L}_{\text{acyc}} \arrow[dr, phantom, sloped, "\subset"] & & \\
\mathscr{L}_{\text{facyc}} \arrow[ur, phantom, sloped, "\subset"] \arrow[dr, phantom, sloped, "\subset"] & & \mathscr{L}_{\text{tgd}} \arrow[r, phantom, "\subset"] & \mathscr{L}_{\text{rc}}. \\
 & \mathscr{L}_{\text{full}} \arrow[ur, phantom, sloped, "\subset"] & &
\end{tikzcd}$$

We focus here on source-to-target tuple-generating dependencies (either arbitrary or with one of the restrictions mentioned above). Arbitrary tgds (and, in a lesser way, full tgds) have been at the basis of most works* in the data exchange setting [FKMP03, Kol05]. As we shall see in Section 5.4, acyclic full tgds have nice complexity results. We show in Section 5.5.1 how this work can be extended to arbitrary formulas of the relational calculus.

*The other important class of dependencies, namely equality generating dependencies, is less appropriate to this context.

5.2 Cost and Optimality of a TGD

We first introduce the two basic notions of *validity* and *explanation* that are at the basis of our framework.

Definition 5.1. A schema mapping Σ is *valid* with respect to a pair of instances (I,J) if $(I,J) \models \Sigma$.

Definition 5.2. A (ground) fact in a schema S is a tuple $R(c_1 \dots c_n)$ where $c_1 \dots c_n \in \mathscr{C}$ and R is a relation of S with arity n.

A schema mapping Σ *explains* a ground fact f in the target schema with respect to a source instance I if, for all instances K of the target schema such that $(I,K) \models \Sigma, f \in K$.

A schema mapping *fully explains* a target instance J with respect to a source instance I if it explains all facts of J with respect to I.

Note that we have quite an asymmetric point of view about the pair of instances here; we do not require a full explanation of I by the facts of J, for instance. This asymmetry is quite common in the context of data exchange.

Example 5.3. Let us consider the following database instances I and J, on schemata $\{(R,1)\}$ and $\{(R',2)\}$.

R
a
b
c
d

R'	
a	a
b	b
c	a
d	d
g	h

We can imagine a number of schema mappings that more or less express the relation between I and J:

$$\Sigma_0 = \varnothing$$
$$\Sigma_1 = \{\forall x\, R(x) \to R'(x,x)\}$$
$$\Sigma_2 = \{\forall x\, R(x) \to \exists y\, R'(x,y)\}$$
$$\Sigma_3 = \{\forall x \forall y\, R(x) \land R(y) \to R'(x,y)\}$$
$$\Sigma_4 = \{\exists x \exists y\, R'(x,y)\}$$

Actually, any combination of these schema mappings may also be of interest.

Σ_0 and Σ_4 seem pretty poor, here, as they fail to explain any facts of J, while there seems to be a definite relation (albeit with some noise) between I and J. Σ_3 explains most of the facts of J, but is far from being valid, since it also explains a large number of incorrect facts such as $R'(a,b)$ or $R'(b,d)$.

Σ_1 and Σ_2 are more interesting. Σ_1 explains 3 facts of J, but also incorrectly predicts $R'(c,c)$. Σ_2 fails to explain any facts of J, but explain most of them at least partially, in the sense that they are explained by a fact with an existentially quantified variable (a *skolem*); in addition, it is valid with respect to (I,J). Neither Σ_1 or Σ_2 explains the last fact of J.

As there seems to be some noise in the operation that produced J from I, it is hard to say with certainty which schema mapping is optimal here, in the sense that it reflects most closely the relation between I and J. At any rate, however, Σ_1 and Σ_2 seem far better candidates than the other ones.

To define in a formal way our notion of optimality of a schema mapping, the basic idea is to get the simultaneous optimal for all three factors of interest (validity, explanation of the target instance, conciseness) by minimizing the size of: the original formula, plus all the local corrections that have to be done for the formula to be valid and to fully explain the target instance. This is close in spirit to the notion of Kolmogorov complexity and Kolmogorov optimal [LV97] (though we do not consider a Turing-complete language for expressing either the formula or the corrections, but much more limited languages).

Definition 5.4. Given a schema mapping of tgds $\Sigma \subset \mathscr{L}_{\text{tgd}}$ and a pair of instances (I,J), we define the set of *repairs* of Σ with respect to (I,J), denoted $\mathsf{repairs}_{(I,J)}(\Sigma)$, as a set of finite sets of formulas, such that $\Sigma' \in \mathsf{repairs}_{(I,J)}(\Sigma)$ if it can be obtained from Σ by a finite sequence of the following operations:

- Adding to the left-hand side of a tgd θ of Σ, with θ of the form

  $$\forall \mathbf{x}\, \varphi(\mathbf{x}) \rightarrow \exists \mathbf{y}\, \psi(\mathbf{x},\mathbf{y}),$$

  a conjunction $\tau(\mathbf{x})$ of the form: $\bigwedge_i x_i \alpha_i c_i$ where α_i are either $=$ or $\neq$, x_i are variables from $\mathbf{x}$ and c_i are constants.

- Adding to the right-hand side of a tgd θ of Σ, with θ as above, a formula $\tau'(\mathbf{x},\mathbf{y})$ of the form:

  $$\bigwedge_i \left(\left(\bigwedge_j x_{ij} = c'_{ij} \right) \rightarrow y_i = c_i \right)$$

  where x_{ij} are variables from $\mathbf{x}$, y_i variables from $\mathbf{y}$, and c'_{ij} and c_i constants.

- Adding to Σ a ground fact $R(c_1 \dots c_n)$ where R is a relation of the target schema of arity n, and $c_1 \dots c_n$ are constants.

The language of repairs $\mathscr{L}^*$ of a language $\mathscr{L}$ is the language consisting of all formulas which can be obtained from formulas of $\mathscr{L}$ with these operations (along with all ground facts over the target schema).

In a repair of a tgd

$$\forall \mathbf{x}\, \varphi(\mathbf{x}) \land \tau(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x},\mathbf{y}) \land \tau'(\mathbf{x},\mathbf{y}),$$

the term $\tau(\mathbf{x})$ is responsible for correcting cases when the tgd is not valid, by adding additional constraints on the universal quantifier, whereas $\tau'(\mathbf{x},\mathbf{y})$ precises the right-hand side of J, by giving the explicit value of each existentially quantified variable, in terms of the universally quantified variables.

An interesting property of repairs is that they are reversible: Because all operations add constants to a language where constants do not exist, it is possible to compute (in linear time) the original schema mapping from a repair. Indeed, constants are only used for repairing formulas; in other words, we consider that the relations that we need to find between the source and target instances are to be expressed with constant-free formulas, in order to abstract them as much as possible. Clearly, this is a simplifying assumption that could be lifted in future works. Note that this extension is not straightforward, however: It is not clear how to distinguish between constants which are rightfully part of the optimal schema mapping description and constants which are just used to correct some noise or missing data.

The notion of size of a schema mapping is easily defined as follows; we could also use a more classical definition in function of the number of symbols of a formula, without much difference in the theory.

Definition 5.5. The *size* of a first-order formula $\varphi \in \mathscr{L}^*$, denoted $\mathsf{size}(\varphi)$ is computed as follows:

- The size of φ is the number of occurrences of variables and constants in φ (we stress that each variable and constant is counted as many times as it occurs in φ); occurrences of variables as arguments of quantifiers do not count.
- If φ is a ground fact $R(c_1 \dots c_n)$, then the size of φ is computed as if φ were the formula
$$\exists x_1 \dots \exists x_n\, R(x_1 \dots x_n) \wedge x_1 = c_1 \wedge \dots \wedge x_n = c_n.$$
Therefore, $\mathsf{size}(\varphi) = 3n$. This refinement is performed so that ground facts are not "too cheap": they are exactly as expensive as the corresponding repair of $\exists x_1 \dots \exists x_n\, R(x_1 \dots x_n)$.

The size of a schema mapping is the sum of the size of its elements.

We are now ready to define the cost of a schema mapping, in terms of the size of its repairs:

Definition 5.6. The *cost* of a schema mapping Σ, with respect to a pair of instances (I,J), is defined by:

$$\mathsf{cost}_{(I,J)}(\Sigma) = \min_{\substack{\Sigma' \in \mathsf{repairs}_{(I,J)}(\Sigma) \\ \Sigma' \text{ valid and fully explains } J}} \mathsf{size}(\Sigma').$$

If the minimizing set is empty, we denote $\mathsf{cost}_{(I,J)}(\Sigma) = \infty$.

A schema mapping $\Sigma \subset \mathscr{L}$ is *optimal* in the language $\mathscr{L}$, with respect to a pair of instances (I,J), if:

$$\mathsf{cost}_{(I,J)}(\Sigma) = \min_{\substack{\Sigma' \subset \mathscr{L} \\ \Sigma' \text{ finite}}} \mathsf{cost}_{(I,J)}(\Sigma').$$

It is indeed possible that $\mathsf{cost}_{(I,J)}(\Sigma) = \infty$. This is for instance the case for $\mathbf{T} = \{(R',1)\}$, $\Sigma = \{\exists x\, R'(x)\}$ and $J = \varnothing$. However, this case is easily recognizable; in other cases we have a linear bound on the cost of a schema mapping:

Proposition 5.7. *There is a linear-time algorithm to check whether a schema mapping in $\mathscr{L}_{\text{tgd}}$ has an infinite cost. If it has a finite cost, the cost is bounded by a linear function of the size of the data and the schema mapping itself.*

Proof. Let Σ be a schema mapping, I and J instances of the source and target schema. Then, Σ has an infinite cost if and only if it contains a tgd θ without a left-hand side (that is, of the form $\exists \mathbf{y}\, \psi(\mathbf{y})$) and which is not valid in J, which can be tested in linear time in the size of J.

To see this, we shall prove that in all other cases, there is a linear bound on $\mathsf{cost}_{(I,J)}(\Sigma)$. Indeed, every tgd with a left-hand side can be canceled by adding an $x = c$ term on the left-hand side, where x is a universally quantified variable and c a constant which does not appear in I. Moreover, all facts of J can be added to the repair of the tgd as ground facts. We then have the following bound on the cost of Σ:

$$\mathsf{cost}_{(I,J)}(\Sigma) \leqslant \mathsf{size}(\Sigma) + 2|\Sigma| + 3r|J|$$

where r is the maximum arity of a target relation. □

This linear bound is interesting, since we can imagine to use local search algorithms to find the tgd with minimum cost, *as soon as we are able to compute in an efficient way the cost of a tgd.* We shall see in Section 5.4, unfortunately, that even for a very restricted language, computing the cost of a tgd is **NP**-complete.

Example 5.8. Let us go back to the instances and schema mapping of Example 5.3, compute their respective cost, and see which one is optimal.

$$\begin{aligned}
\mathsf{cost}_{(I,J)}(\Sigma_0) &= 3\cdot 2|J| = 30\\
\mathsf{cost}_{(I,J)}(\Sigma_1) &= 3+2+3\cdot 2\cdot 2 = 17\\
\mathsf{cost}_{(I,J)}(\Sigma_2) &= 3+4\cdot 4+3\cdot 2 = 25\\
\mathsf{cost}_{(I,J)}(\Sigma_3) &= 4+4+3\cdot 2\cdot 4 = 32\\
\mathsf{cost}_{(I,J)}(\Sigma_4) &= 2+4+3\cdot 2\cdot 4 = 30
\end{aligned}$$

It appears here that Σ_1 is the best of these schema mappings (and it can be shown that it is indeed optimal). As expected, Σ_2 is the second best.

This is no coincidence here if Σ_4 has the same cost as Σ_0, this is due to the choice we made for the cost of a ground fact.

At least on this simple example, our measure of cost seems reasonable. We will further justify it in Section 5.3.

The following decision problems arise naturally once given this notion of optimality. Each of them is defined for a given language $\mathscr{L}$, and we shall investigate their complexity in Section 5.4.

Validity. Given instances I, J, and a schema mapping $\Sigma \subset \mathscr{L}^*$, is Σ valid with respect to (I,J)?

Explanation. Given instances I, J, and a schema mapping $\Sigma \subset \mathscr{L}^*$, does Σ fully explain J with respect to I?

Zero-Repair. Given instances I, J, and a schema mapping $\Sigma \subset \mathscr{L}^*$, is $\mathsf{cost}_{(I,J)}(\Sigma)$ equal to $\mathsf{size}(\Sigma)$?

Cost. Given instances I, J, a schema mapping $\Sigma \subset \mathscr{L}$ and an integer $K \geqslant 0$, is $\mathsf{cost}_{(I,J)}(\Sigma)$ less than or equal to K?

Existence-Cost. Given instances I, J and an integer $K \geqslant 0$, does there exist a schema mapping $\Sigma \subset \mathscr{L}$ such that $\mathsf{cost}_{(I,J)}(\Sigma)$ is less than or equal to K?

Optimality. Given instances I, J, and a schema mapping $\Sigma \subset \mathscr{L}$, is it true that Σ is optimal with respect to (I,J)?

5.3 Justification of this Notion of Optimality

In this section, we justify the definitions of the previous section by observing that, when instances I and J are derived from each other by elementary operators of the relational algebra, the optimal schema mapping, in $\mathscr{L}_{\text{tgd}}$, is the one that "naturally" describes this operator.

Let r, r' be instances of relations. We consider the following elementary operators of the relational algebra:

Projection. $\pi_i(r)$ denotes the projection of r along its ith attribute.

Selection. $\sigma_\varphi(r)$, where φ is a *conjunction* of equalities and negated equalities between an attribute of r and a constant, denotes the selection of r according to φ. Note that we allow neither identities between attributes of r (this is the role of the *join* operation), nor disjunctions (they may be expressed using a combination of selection and union).

Union. $r \cup r'$ is the union of r and r'.

Intersection. $r \cap r'$ is the intersection of r and r'.

Product. $r \times r'$ is the cross product of r and r'.

Join. $r \bowtie_\varphi r'$ is the join of r and r' according to φ, where φ is an equality between an attribute of r and an attribute of r'; φ is omitted when the context makes it clear.

The relationship between a database instance I and the instance J obtained from I using one of these operators can often be (partially) expressed in a natural way by a tgd or a set of tgds, where I is the source instance and J the target instance (and similarly when the source instance is expressed as the result of applying some operator to the target instance). For instance $\forall x\, R_1(x) \wedge R_2(x) \rightarrow R'(x)$ is naturally associated with the intersection operator. The last two columns of Table 5.1 describe more precisely what schema mappings we associate to what operator of the relational algebra (in order not to clutter the table, universal quantifiers on the front of tgds are not shown). In some cases, as tgds are not powerful enough to express the relationship, or as some information is lost, the correspondence is only partial. The only missing case in Table 5.1 is for the reciprocal of the union operator: If $I = R_1^J \cup R_2^J$, the natural formula for describing this relation is $\forall x\, R(x) \rightarrow R_1(x) \vee R_2(x)$, which is not a tgd since we do not allow disjunction.

We now state that, using the notion of optimality of a schema mapping with respect to a pair of instances described in the previous section, and with some simple restrictions on the considered instances, the optimal schema mapping for a pair of instances obtained from each other with an operator of the relational algebra is precisely the schema mapping that is naturally associated with the operator. This justifies the choice of this notion of optimality, at least in these elementary contexts. We shall see in Section 5.5.2 other choices for the cost function, that might seem more natural at first, but that fail to satisfy the same property.

Theorem 5.9. *The tgds presented in the last column of Table 5.1 are optimal with respect to (I,J), when I and J are as described. In other words, for any elementary operator γ of the relational algebra, the tgd naturally associated with this operator (when it exists) is optimal with respect to $(I,\gamma(I))$ (or $(\gamma(J),J)$, depending on the considered case), if the basic assumptions shown on Table 5.1 (the instances are not of trivial size, there is no other relation between attributes than the one needed for the operator) are fulfilled.*

Proof. First observe that the size of a ground fact of arity n is the same as the size of any maximal repair of the tgd without a left-hand side $\exists x_1 \dots \exists x_n\, R(x_1' \dots x_n')$. This means that we do not need to consider such tgds.

Projection. Suppose $J = \pi_1(I)$ with $I \neq \varnothing$. Then:

$$\mathsf{cost}_{(I,J)}\left(\{\forall x \forall y\, R(x,y) \rightarrow R'(x)\}\right) = 3$$

Table 5.1: Optimal tgds for elementary operations of the relational algebra

	S	T	Condition on I	I and J	Optimal schema mapping
Projection	$(R,2)$	$(R',1)$	I non-empty	$J=\pi_1(I)$	$R(x,y)\rightarrow R'(x)$
	$(R,1)$	$(R',2)$	$\pi_1(J)\cap\pi_2(J)=\varnothing$, $\lvert\pi_1(J)\rvert\geqslant 2$	$I=\pi_1(J)$	$R(x)\rightarrow \exists y\, R'(x,y)$
Selection	$(R,1)$	$(R',1)$	$\left\lvert\sigma_\varphi(I)\right\rvert \geqslant \frac{\text{size}(\varphi)+2}{3}$	$J=\sigma_\varphi(I)$	$R(x)\rightarrow R'(x)$
	$(R,1)$	$(R',1)$	$\sigma_\varphi(J)$ non-empty	$I=\sigma_\varphi(J)$	$R(x)\rightarrow R'(x)$
Union	$(R_1,1),(R_2,2)$	$(R',1)$	$R_1{}^I \subsetneq R_1{}^I\cup R_2{}^I$, $R_2{}^I \subsetneq R_1{}^I\cup R_2{}^I$	$J=R_1^I\cup R_2^I$	$R_1(x)\rightarrow R'(x), R_2(x)\rightarrow R'(x)$
Intersection	$(R_1,1),(R_2,2)$	$(R',1)$	$R_1^I\not\subseteq R_2^I$, $R_2^I\not\subseteq R_1^I$, $R_1^I\cap R_2^I\neq\varnothing$	$J=R_1^I\cap R_2^I$	$R_1(x)\wedge R_2(x)\rightarrow R'(x)$
	$(R,1)$	$(R_1',1),(R_2',1)$	$R_1'^J\cap R_2'^J$ non-empty	$I=R_1'^J\cap R_2'^J$	$R(x)\rightarrow R_1'(x)\wedge R_2'(x)$
Product	$(R_1,1),(R_2,1)$	$(R',2)$	R_1^I and R_2^I non-empty	$J=R_1^I\times R_2^I$	$R_1(x)\wedge R_2(y)\rightarrow R'(x,y)$
	$(R,2)$	$(R_1',1),(R_2',1)$	$R_1'^J$ and $R_2'^J$ non-empty	$I=R_1'^J\times R_2'^J$	$R(x,y)\rightarrow R_1'(x)\wedge R_2'(y)$
Join	$(R_1,2),(R_2,2)$	$(R',3)$	$\pi_i(R_1^I\bowtie R_2^I)\neq\pi_j(R_1^I\bowtie R_2^I)$ $(i\neq j)$	$J=R_1^I\bowtie R_2^I$	$R_1(x,y)\wedge R_2(y,z)\rightarrow R'(x,y,z)$
	$(R,3)$	$(R_1',2),(R_2',2)$	$\pi_i(R_1'^J\bowtie R_2'^J)\cap\pi_j(R_1'^J\bowtie R_2'^J)=\varnothing$ $(i\neq j)$ $R_1'^J\bowtie R_2'^J$ non-empty	$I=R_1'^J\bowtie R_2'^J$	$R(x,y,z)\rightarrow R_1'(x,y)\wedge R_2'(y,z)$

since it is valid and fully explains all facts of J. We then only need to consider schema mappings of size strictly lesser than 3. The only relevant one is the empty schema mapping and $\mathsf{cost}_{(I,J)}(\varnothing) = 3|J| = 3|I|$ which is greater or equal to 3 as soon as $I \neq \varnothing$.

Consider now the case when $I = \pi_1(J)$ with $\pi_1(J) \cap \pi_2(J) = \varnothing$, $|\pi_1(J)| \geqslant 2$. We have:

$$\mathsf{cost}_{(I,J)}\left(\{\forall x\, R(x) \to \exists x\, R'(x,y)\}\right) = 3 + 4|I| + 3 \cdot 2(|J| - |I|) = 3 - 2|I| + 6|J|$$

(this is a valid schema mapping, but it fails to explain all facts of J; $|I|$ facts can be explained by repairing the existential quantifier, all others must be given as ground facts). As we assume that attributes of J have disjoint domains, all schema mappings with a $R'(w_1, w_2)$ where w_2 is not existentially quantified do not explain any facts of J. The only remaining schema mapping of interest is then $\varnothing$, whose cost is $6|J|$, which is greater than $3 - 2|I| + 6|J|$ as soon as $|I| \geqslant 2$.

Selection. If $J = \sigma_\varphi(I)$, with $\left|\sigma_\varphi(I)\right| \geqslant \frac{\mathsf{size}(\varphi)+2}{3}$ (in the common case where φ is a single equality or negated equality, $\mathsf{size}(\varphi) = 2$ and this condition amounts to $|J| \geqslant 2$), we have:

$$\mathsf{cost}_{(I,J)}\left(\{\forall x\, R(x) \to R'(x)\}\right) \leqslant \mathsf{size}(\forall x\, R(x) \wedge \varphi(x) \to R'(x)) = 2 + \mathsf{size}(\varphi).$$

The only other schema mapping which might have a lesser cost is $\varnothing$ and $\mathsf{cost}_{(I,J)}(\varnothing) = 3|J| = 3\left|\sigma_\varphi(I)\right|$.

Now, suppose $I = \sigma_\varphi(J)$ with $\sigma_\varphi(J) \neq \varnothing$. Observe that

$$\mathsf{cost}_{(I,J)}\left(\{\forall x\, R(x) \to R'(x)\}\right) = 2 + 3(|J| - |I|)$$

is lesser than $\mathsf{cost}_{(I,J)}(\varnothing) = 3|J|$ as soon as $|I| \neq \varnothing$.

Union. If $J = R_1^I \cup R_2^I$ with both relations strictly included in their union,

$$\mathsf{cost}_{(I,J)}\left(\{\forall x\, R_1(x) \to R'(x), \forall x\, R_2(x) \to R'(x)\}\right) = 4$$

while the cost of each of these tgds alone is greater than 5. We also have:

$$\mathsf{cost}_{(I,J)}\left(\{\forall x\, R_1(x) \wedge R_2(x) \to R'(x)\}\right) = 3 + 3\left(\left|R_1^I \cup R_2^I\right| - \left|R_1^I \cap R_2^I\right|\right) \geqslant 9.$$

Finally, the cost of the empty schema mapping is: $3\left|R_1^I \cup R_2^I\right| \geqslant 6$.

Intersection. Suppose $J = R_1^I \cap R_2^I$ with neither of these relations containing the other one;

$$\mathsf{cost}_{(I,J)}\left(\{\forall x\, R_1(x) \wedge R_2(x) \to R'(x)\}\right) = 3.$$

Neither $\{\forall x\, R_1(x) \to R'(x)\}$ nor $\{\forall x\, R_2(x) \to R'(x)\}$ nor the empty schema mapping have a lesser cost as soon as both R_1^I and R_2^I contain facts not in the other one, and the intersection is not empty.

Consider now the case where $I = R_1'^I \cap R_2'^I$. Then:

$$\mathsf{cost}_{(I,J)}\left(\{\forall x\, R(x) \to R_1'(x) \wedge R_2'(x)\}\right) = 3 + 3\left(\left|R_1'^J\right| + \left|R_2'^J\right| - 2|I|\right)$$

while

$$\begin{aligned}
\mathsf{cost}_{(I,J)}\left(\{\forall x\, R(x) \rightarrow R'_1(x)\}\right) &= 2 + 3\left(\left|R_1'^J\right| + \left|R_2'^J\right| - |I|\right)\\
\mathsf{cost}_{(I,J)}\left(\{\forall x\, R(x) \rightarrow R'_2(x)\}\right) &= 2 + 3\left(\left|R_1'^J\right| + \left|R_2'^J\right| - |I|\right)\\
\mathsf{cost}_{(I,J)}(\varnothing) &= 3\left(\left|R_1'^J\right| + \left|R_2'^J\right|\right).
\end{aligned}$$

The first schema mapping has a lower cost than the other ones as soon as $I \neq \varnothing$.

Product. In both cases, the cost of the schema mapping from Table 5.1 is 4 (it is valid and explains all facts of J) and, unless one of the instance is empty, no other schema mapping of lesser size is valid and explains all facts of J.

Join. Suppose $J = R_1^I \bowtie R_2^I$ with $\pi_1(J) \neq \pi_2(J)$, $\pi_1(J) \neq \pi_3(J)$, $\pi_2(J) \neq \pi_3(J)$. We have:

$$\mathsf{cost}_{(I,J)}\left(\{\forall x \forall y \forall z\, R_1(x,y) \wedge R_2(y,z) \rightarrow R'(x,y,z)\}\right) = 7$$

since this tgd is valid and explains all facts of J. The cost of the empty schema mapping, $9\,|J|$, is greater since J is not empty. The only remaining relevant schema mappings with lesser size (of 5) have a single relation symbol R_1 or R_2 on the left-hand-side. But this means that they either predict two identical columns in J (this is incorrect, and has to be fixed in a repair of the schema mapping, whose additional cost is at least 2), or use an existential quantifier on the right-hand size, which also has to be repaired.

Now consider the last case of the proof, where $I = R_1'^J \bowtie R_2'^J$, with all three attributes disjoint.

$$\begin{aligned}
&\mathsf{cost}_{(I,J)}\left(\{\forall x \forall y \forall z\, R(x,y,z) \rightarrow R'_1(x,y) \wedge R'_2(y,z)\}\right)\\
&= 7 + 6\left|\left\{(x,y) \in R_1'^J \mid \forall z\,(y,z) \notin R_2'^J\right\}\right| + 6\left|\left\{(y,z) \in R_2'^J \mid \forall x\,(x,y) \notin R_1'^J\right\}\right|.
\end{aligned}$$

$\mathsf{cost}_{(I,J)}(\varnothing) = 6\,|J|$ is greater than that as soon as I is not empty. As we assumed all three attributes of I disjoint, we can eliminate a number of schema mappings that do not produce any correct facts. The only remaining ones only have $R'_1(w_1,w_2)$ or $R'_2(w_2,w_3)$ terms on the right-hand size with those three variables either existentially quantified or appearing, respectively in the first, second or third position of a $R(w_1,w_2,w_3)$ atom on the left-hand side. None of these schema mappings can explain the facts that the schema mapping above does not explain, and existential quantifiers have to be accounted for in repairs. □

Note that, in all cases, the optimal tgd is acyclic, and in all but one it is full. These results could also be extended to the cases where we have relations of greater arity, but we would then require strong constraints, as the one we imposed for reciprocal projection and reciprocal join in Table 5.1 (that all attributes are disjoint), so as not to have any "hidden" relation between the different attributes. A weaker assumption that could be made is to use a notion of *Kolmogorov randomness* [LV97]: A database instance selected at random cannot have a description of length lower than its size, thanks to a simple counting argument. We can use such random instances to get a contradiction when we obtain a schema mapping that uses hidden relations between attributes of relations in the instance to have a lower cost than the schema mapping from Table 5.1.

5.4 Complexity Study

We now proceed to a study of the computational complexity of the different problems identified in Section 5.2, for the different subsets of $\mathscr{L}_{\text{tgd}}$ that we presented in Section 5.1. We focus here on *combined complexity* (when K and Σ are part of the input to the problem), since we are precisely reasoning about the schema mappings themselves. We first describe general relationships between the different problems, before giving complexity results for $\mathscr{L}_{\text{tgd}}$, $\mathscr{L}_{\text{full}}$, $\mathscr{L}_{\text{facyc}}$ and $\mathscr{L}_{\text{acyc}}$, in that order (it might seem natural to analyze $\mathscr{L}_{\text{acyc}}$ before $\mathscr{L}_{\text{facyc}}$ but the proofs for the latter are slightly simpler, and will help to understand the proofs for the former). Cost and Existence-Cost will be discussed separately. We present at the end of the section *data complexity* results.

5.4.1 General Complexity Results

As the complexity of the different decision problems depends on the particular language considered, we add to the problem name a subscript identifying the considered language (say, $\text{Optimality}_{\text{tgd}}$ for the Optimality problem in $\mathscr{L}_{\text{tgd}}$).

We have the following elementary relationships between these problems, that can be used to derive complexity results for one problem from complexity results for another one.

Proposition 5.10. *For any language $\mathscr{L}$:*

1. Zero-Repair = Validity $\cap$ Explanation.
2. *There is a polynomial-time reduction of* Validity *to* Zero-Repair.
3. *There is a polynomial-time reduction of* Zero-Repair *to* Cost.
4. *Given an algorithm $\mathscr{A}$ for* Zero-Repair*, and a polynomial-time algorithm for determining if a formula is in $\mathscr{L}$, there are non-deterministic algorithms for* Cost *and* Existence-Cost *that run by using once the algorithm $\mathscr{A}$, with an additional polynomial time cost.*
5. *Given an algorithm $\mathscr{A}$ for* Cost*, and a polynomial-time algorithm for determining if a formula is in $\mathscr{L}$, there is a non-deterministic algorithm for the complement of* Optimality *that runs by using a logarithmic number of times the algorithm $\mathscr{A}$, with an additional polynomial time cost.*
6. *If $\mathscr{L} \subseteq \mathscr{L}'$, for any problem among* Validity, Explanation, Zero-Repair *and* Cost*, there is a constant-time reduction from the problem in $\mathscr{L}$ to the problem in $\mathscr{L}'$.*

Proof.

1. By definition, $\mathsf{cost}_{(I,J)}(\Sigma) \geqslant \mathsf{size}(\Sigma)$. Because the size of a repaired formula is always greater than the original formula, the only case when the equality occurs is when the original formula is valid and fully explains J.
2. Let (I,J,Σ) be an instance of Validity. Let Σ' be the union of Σ and of all ground facts of J. Obviously, Σ' fully explains J with respect to I. That means that $\mathsf{cost}_{(I,J)}(\Sigma') = \mathsf{size}(\Sigma')$ if and only if Σ' is valid with respect to (I,J). As the ground facts of Σ' do not change its validity, $\mathsf{cost}_{(I,J)}(\Sigma') = \mathsf{size}(\Sigma')$ if and only if Σ is valid with respect to (I,J).
3. Just take $K = \mathsf{size}(\Sigma)$, which is computable in linear time.

Algorithm 5.1 Cost (non-deterministic)

```
INPUT: Instances $I, J$, schema mapping $\Sigma$, $K \geqslant 0$.
OUTPUT: Answer to Cost.
(a) Let $K'$ be the minimum between $K$ and the upper bound of Proposition 5.7.
(b) Guess a set of formulas $\Sigma'$ of total size less than or equal to $K'$.
(c) Check that $\Sigma'$ is a repair of $\Sigma$. Otherwise, give up this guess.
(d) Apply $\mathscr{A}$ on $\Sigma'$; if the result is true, return true.
```

4. Consider the non-deterministic algorithm for Cost shown as Algorithm 5.1.

   The algorithm for Existence-Cost is very similar, just replace the bound on K with the cost of the empty schema mapping, and step (c) by a check that Σ' is in $\mathscr{L}^*$ (this can be done in polynomial time by hypothesis). Note that the bound of $\mathsf{cost}_{(I,J)}(\varnothing)$ on the guess is critical, since otherwise the guess would be of size K, and thus exponential in the length of the representation of K.

5. Algorithm 5.2 is a non-deterministic algorithm for the complement of Optimality.

Algorithm 5.2 Optimality (non-deterministic)

```
INPUT: Instances $I, J$, schema mapping $\Sigma$.
OUTPUT: Answer to the complement of Optimality.
(a) Use $\mathscr{A}$ a logarithmic number of times to compute $K = \mathsf{cost}_{(I,J)}(\Sigma)$ (observe that we have a
    linear bound on this value, since we can cancel all formulas of $\Sigma$ and then add all ground facts
    of $J$ with a linear number of repairs).
(b) Guess a set of formulas $\Sigma'$ of total size less than $K$.
(c) Check that $\Sigma'$ is in $\mathscr{L}$. Otherwise, give up this guess.
(d) Apply $\mathscr{A}$ on $(\Sigma', K-1)$; if the result is true, return true.
```

6. Directly results from the fact that a formula of $\mathscr{L}$ is also a formula of $\mathscr{L}'$. Note that this property does not necessarily hold for Existence-Cost and Optimality, since both of them depend on the *existence* of a formula in the underlying language. □

Note that for all languages considered here, there is a linear-time algorithm for determining if a formula is in this language; this is obvious for all except for $\mathscr{L}_{\text{acyc}}$ and $\mathscr{L}_{\text{facyc}}$, and an algorithm from [TY84] gives a linear-time algorithm for the acyclicity of hypergraphs.

In the next sections, we shall investigate in detail the complexity of the different problems in each of the identified subsets of $\mathscr{L}_{\text{tgd}}$, starting from $\mathscr{L}_{\text{tgd}}$ itself. A summary of all combined complexity results proved in the following, along with their direct consequences, is shown in Table 5.2.

5.4.2 Combined Complexity for Tuple-Generating Dependencies

Let us first investigate the combined complexity of Validity and Explanation in $\mathscr{L}_{\text{tgd}}$.

Proposition 5.11.

1. Validity$_{\text{tgd}}$ *is* $\mathbf{\Pi}_2^P$*-complete.*
2. Explanation$_{\text{tgd}}$ *is in NP.*

Table 5.2: Combined complexity results

	$\mathscr{L}_{\text{tgd}}$	$\mathscr{L}_{\text{full}}$	$\mathscr{L}_{\text{acyc}}$	$\mathscr{L}_{\text{facyc}}$
Validity	$\mathbf{\Pi}_2^P$-complete	**coNP**-complete	**coNP**-complete	**PTIME**
Explanation	**NP**-complete	**NP**-complete	**NP**-complete	**PTIME**
Zero-Repair	$\mathbf{\Pi}_2^P$-complete	**DP**, (**co**)**NP**-hard	**DP**, (**co**)**NP**-hard	**PTIME**
Cost	$\mathbf{\Sigma}_3^P$, $\mathbf{\Pi}_2^P$-hard	$\mathbf{\Sigma}_2^P$, (**co**)**NP**-hard	$\mathbf{\Sigma}_2^P$, (**co**)**NP**-hard	**NP**-complete
Existence-Cost	$\mathbf{\Sigma}_3^P$, **NP**-hard	$\mathbf{\Sigma}_2^P$, **NP**-hard	$\mathbf{\Sigma}_2^P$, **NP**-hard	**NP**-complete
Optimality	$\mathbf{\Pi}_4^P$, (**co**)**NP**-hard	$\mathbf{\Pi}_3^P$, (**co**)**NP**-hard	$\mathbf{\Pi}_3^P$, (**co**)**NP**-hard	$\mathbf{\Pi}_2^P$, (**co**)**NP**-hard

Proof.

1. VALIDITY$_{\text{tgd}}$ is clearly in $\mathbf{\Pi}_2^P$. First check the validity of the ground facts. For the other formulas of Σ, guess a valuation of the variables of the left-hand side; if the left-hand side is false (can be decided in polynomial time), give up the guess. Otherwise, use the **NP** oracle to decide whether the right-hand side holds; if it does not, return `false`.

   For the hardness part, we use a reduction of $\forall\exists$3SAT: the satisfiability of the formula $\forall\mathbf{x}\exists\mathbf{y}\ \varphi(\mathbf{x},\mathbf{y})$, where φ is a propositional formula in 3-CNF over $\mathbf{x}\cup\mathbf{y}$. This problem is $\mathbf{\Pi}_2^P$-complete [Wra76, SU02]. Let $\forall\mathbf{x}\exists\mathbf{y}\ \bigwedge_{i=1}^{n} c_i(z_{i1},z_{i2},z_{i3})$ be an instance of $\forall\exists$3SAT, where each c_i is a 3-clause, and each z_{ij} is one of the variables of $\mathbf{x}\cup\mathbf{y}$. We consider then the following instance of VALIDITY$_{\text{tgd}}$:

   - $\mathrm{S}=\{(B,1)\}$ and $I=\{B(0),B(1)\}$;
   - $\mathrm{T}=\{(R_1,3)\dots(R_8,3)\}$ and J is such that the R_i^J's are the 8 distinct subsets of $\{0,1\}^3$ of cardinality 7 (this corresponds to the 8 possible truth tables of a 3-clause);
   - For each $1\leqslant i\leqslant n$, let $1\leqslant k_i\leqslant 8$ be the unique integer such that $c_i(z_{i1},z_{i2},z_{i3})$ is true if and only if $(z_{i1},z_{i2},z_{i3})\in R_{k_i}^J$ (with the usual abuse of notation of identifying values of boolean variables and values in $\{0,1\}$). We now define Σ as follows:

   $$\Sigma=\left\{\forall\mathbf{x}\bigwedge_{i=1}^{m}B(x_i)\rightarrow\exists\mathbf{y}\bigwedge_{i=1}^{n}R_{k_i}(z_{i1},z_{i2},z_{i3})\right\}.$$

   This reduction is clearly polynomial. Now we have:

   $$\begin{aligned}(I,J)\models\Sigma &\iff \left(\forall\mathbf{x}\bigwedge_{i=1}^{m}(x_i=0\vee x_i=1)\rightarrow\exists\mathbf{y}\bigwedge_{i=1}^{n}c_i(z_{i1},z_{i2},z_{i3})\right)\text{ is true}\\ &\iff \forall\mathbf{x}\exists\mathbf{y}\ \varphi(\mathbf{x},\mathbf{y})\text{ is true}\end{aligned}$$

2. Let F be the set of facts of J which are not directly in Σ as ground facts. Guess $|F|$ valuations of the variables on the left- and right-hand sides of the formulas of Σ which are not ground facts. If, for all $1\leqslant i\leqslant|F|$, there is some $\varphi\in\Sigma$ such that the ith valuation of the left-hand side of φ holds in I (which can be decided in polynomial time) and the ith fact of F appears in the ith valuation of the right-hand side of φ, then return `true`. □

5.4.3 Combined Complexity for Full Tuple-Generating Dependencies

We now consider the language of full tgds, $\mathscr{L}_{\text{full}}$.

Proposition 5.12.

1. VALIDITY$_{\text{full}}$ *is coNP-complete;*
2. EXPLANATION$_{\text{full}}$ *and* ZERO-REPAIR$_{\text{full}}$ *are NP-hard.*

Proof.

1. Here is a non-deterministic polynomial-time algorithm for this problem. First check the validity of the ground facts of Σ. For the other formulas of Σ, guess a valuation of the universally quantified variables. If the left-hand side is true in I and the right-hand side is false in J (both of which can be decided in polynomial time), return `false`.

   For the hardness part, we use a reduction from the problem of evaluating a boolean conjunctive query over a database, which is **NP**-complete [CM77]. Let D be a relational database of schema $\mathbf{U}$, and $Q = \exists \mathbf{x}\ \varphi(\mathbf{x})$ a boolean conjunctive query over $\mathbf{U}$. We build an instance (I,J,Σ) of $\textsc{Validity}_{\text{full}}$ in the following way: $\mathbf{S} = \mathbf{U} \cup \{(R,1)\}$, $I = D \cup \{R(1)\}$, $\mathbf{T} = \{(R',1)\}$, $J = \varnothing$ and $\Sigma = \{\forall \mathbf{x} \forall y\ \varphi(\mathbf{x}) \wedge R(y) \rightarrow R'(y)\}$. This reduction is polynomial, and Σ is valid with respect to (I,J) if and only if Q does not match D.

2. There is a straightforward reduction of the problem of deciding whether a tuple is in the result of a project-join expression in the relational algebra, which is a **NP**-complete problem [Yan81], to $\textsc{Explanation}_{\text{full}}$. However, we use another reduction, from 3SAT, which works both for $\textsc{Explanation}_{\text{full}}$ and $\textsc{Zero-Repair}_{\text{full}}$. It is similar to the reduction used in the proof of Proposition 5.11.

   Let $\varphi = \bigwedge_{i=1}^{n} c_i(z_{i1}, z_{i2}, z_{i3})$ be an instance of 3SAT, where the c_i are 3-clauses over some set $\mathbf{x}$ of variables (and $\bigcup_{i,j} z_{ij} = \mathbf{x}$). We consider the following instance of the problems $\textsc{Explanation}_{\text{full}}$ and $\textsc{Zero-Repair}_{\text{full}}$:

   - $\mathbf{S} = \{(R,1),(R_1,3)\dots(R_8,3)\}$ and I such that the R_i^I are the 8 distinct subsets of $\{0,1\}^3$ of cardinality 7, and $R^I = \{a\}$;
   - $\mathbf{T} = \{(R',1)\}$ and $J = \{R'(a)\}$;
   - For each $1 \leqslant i \leqslant n$, let $1 \leqslant k_i \leqslant 8$ be the unique integer such that $c_i(z_{i1}, z_{i2}, z_{i3})$ is true if and only if $(z_{i1}, z_{i2}, z_{i3}) \in R_{k_i}^I$. We then define Σ:

   $$\Sigma = \left\{ \forall \mathbf{x} \bigwedge_{i=1}^{n} R_{k_i}(z_{i1}, z_{i2}, z_{i3}) \wedge R(x) \rightarrow R'(x) \right\}.$$

   We have $(I,J) \models \Sigma$, which means that (I,J,Σ) is a solution of $\textsc{Explanation}_{\text{full}}$ if and only if it is a solution of $\textsc{Zero-Repair}_{\text{full}}$. Now, observe that φ is satisfiable if and only if $R'(a)$ is a necessary consequence of I and Σ or, in other words, if Σ fully explains J with respect to I. □

5.4.4 Combined Complexity for Full Acyclic Tuple-Generating Dependencies

We now look at the complexity of the same problems for $\mathscr{L}_{\text{facyc}}$. We shall need additional notions on *acyclic joins* from [BFM+81, Yan81]. Note first that an acyclic full tgd $\forall \mathbf{x}\ \varphi(\mathbf{x}) \rightarrow \psi(\mathbf{x})$ that describes the relation between a pair of instances (I,J) can be seen, in the relational algebra, as a project-join expression over the source instance, $\pi_\psi(\bowtie_\varphi(I))$, φ expressing the join (which is, by hypothesis, acyclic) and ψ expressing the projection. Adding repaired formulas, of the form $\forall \mathbf{x}\,(\varphi(\mathbf{x}) \wedge \tau(\mathbf{x})) \rightarrow \psi(\mathbf{x})$, means adding an additional selection: $\pi_\psi(\sigma_\tau(\bowtie_\varphi(I)))$.

A *full reducer* of a join expression is a program which removes some tuples to the relations to be joined (by performing semi-joins) so that each relation can then be retrieved as a projection of the full join. Such a full reducer always exists in acyclic databases and can be obtained in polynomial time [BC81]. The full reducer itself runs in polynomial time. Finally, note that a join tree of an acyclic join can be obtained in linear time [TY84].

[Yan81] proposes then Algorithm 5.3 for computing the result to a project-join expression on an acyclic database, that we reuse with slight modifications in our next proposition.

Algorithm 5.3 Result to a project-join expression on an acyclic database (after [Yan81])

```
INPUT: An acyclic join expression $\varphi$, a project expression $\psi$, an instance $I$.
OUTPUT: $\pi_{\psi}(\bowtie_{\varphi}(I))$.
(a) Compute a full reducer of the relation instances, and apply it.
(b) Compute a join tree $T$ of the acyclic expression. Each node of the tree initially contains the
    corresponding reduced relation instance.
(c) For each subtree of $T$ with root $r$, compute recursively for each child $r'$ of $r$ the join of $r$ with $r'$,
    and project to the union of the variables appearing in $\psi$ and the common variables of $r$ and $r'$.
    Remove $r'$ and replace node $r$ with this result.
```

An important property of this algorithm is that, at all time, the size of the relation stored in node r of T is bounded by the original (reduced) size of r times the size of the final output. This means in particular that this algorithm computes in polynomial time the result to the project-join expression. Actually, the same algorithm can be applied when repaired formulas are considered, since the only selection performed is a conjunction of constraints (equality and negated equality) on a given variable: These selections can be pushed inside the join.

Proposition 5.13. VALIDITY$_{\text{facyc}}$ *and* EXPLANATION$_{\text{facyc}}$ *are in PTIME.*

Proof.

1. First check that the ground facts of Σ are valid. Then, we have to apply Algorithm 5.3 on each φ of Σ which is not a ground fact to check whether its output is included in J. This, however, may not lead to a polynomial time algorithm, since Algorithm 5.3 is polynomial in the size of the join expression, the input, *and the output.* The solution is to take care, at each join step, that the output remains in the bound given above. If it does not, then we know that $(I,J) \not\models \varphi$. If it does, we can let the algorithm terminate and check then if the output is included in I.

2. For each fact f of J, proceed as follows. If f appears as a ground fact in Σ, it is fully explained in Σ. Otherwise, for each formula of Σ which is not a ground fact, we apply a variant of the algorithm presented above to decide whether f is in the output of the original algorithm (once again, by pushing selections inside joins), as described in Corollary 4.1 of [Yan81]. □

ZERO-REPAIR is then tractable in $\mathscr{L}_{\text{facyc}}$. One might hope that this tractability extends to COST. Unfortunately, we now show the **NP**-hardness of COST$_{\text{facyc}}$, even for a very simple schema mapping. For this purpose, we shall first need a quite general result on the minimal size of a vertex cover in a r-partite r-uniform hypergraph (for $r \geqslant 3$).

A hypergraph is r-partite if the set of vertices can be decomposed into an r-partition, such that no two vertices of the same partitioning subset are in a same hyperedge. It is r-uniform if all hyperedges have a cardinality of r. A vertex cover of a hypergraph is a subset X of the set of vertices, such that for every hyperedge e, at least one of the elements of e is in X. In regular graphs, VERTEX-COVER (determining whether there is a vertex cover of size $\leqslant K$) is one of the most known and useful **NP**-complete problems [GJ79]. This obviously implies that VERTEX-COVER is **NP**-hard in general hypergraphs. Note that a 2-partite 2-uniform hypergraph is just a bipartite graph, and Vertex-Cover in bipartite graphs is **PTIME**, thanks to Kőnig's theorem [Die05, Kőn36] which states that the maximal number of matchings in a bipartite graph is the mimimum size of a vertex cover.

Lemma 5.14. *The problem of, given an r-partite r-uniform hypergraph $\mathscr{H}$ and a constant K, determining whether there exists a vertex cover in $\mathscr{H}$ of size less than or equal to K is **NP**-complete for $r \geqslant 3$.*

Proof. This problem is clearly in **NP**: Just guess a set of vertices of size less than or equal to K and check in polynomial time whether it is a vertex cover. For the hardness part, we prove the case $r = 3$; there is an obvious reduction from this case to the same problem for other values of r. We use a reduction from 3SAT.

Note that this result appears in [ISOY02], but the reduction presented there is not exhaustive (in particular, nothing is said about interdependencies between clauses, or the fact that the hypergraph is tripartite) and it is not clear whether the proof was indeed led to completion. We use here a proof inspired by the proof that 3-Dimensional-Matching is **NP**-hard in [GJ79].

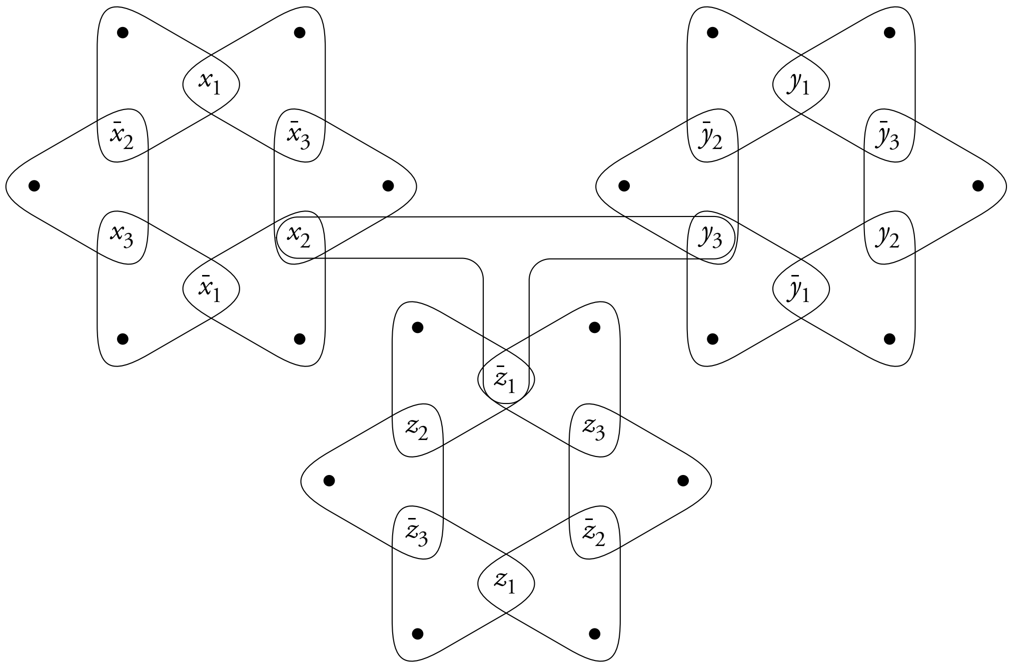


Figure 5.1: Example tripartite hypergraph corresponding to the 3SAT instance $\neg z \lor x \lor y$

Let $\varphi = \bigwedge_{i=1}^{n} c_i$ be an instance of 3SAT, where the c_i are 3-clauses over some set $\mathbf{x}$ of variables. We build a tripartite 3-uniform hypergraph $\mathscr{H} = (V, E)$ (with vertex partition $V = V_1 \cup V_2 \cup V_3$) in the following way (see Figure 5.1 for an illustration when $\varphi = \neg z \lor x \lor y$):

- For each variable $x \in \mathbf{x}$, we add 12 nodes and 6 hyperedges to $\mathscr{H}$. 6 out of the 12 nodes are anonymous nodes which only appear in one hyperedge; they are denoted by •. The other nodes are denoted x_1, x_2, x_3, $\bar{x}_1$, $\bar{x}_2$, $\bar{x}_3$. Intuitively, all x_i's are in a minimum covering if and only if a valuation satisfying φ maps x_i to true (similarly with the $\bar{x}_i$'s and false). For each i, x_i and $\bar{x}_i$ belong to V_i. The so-called *local* hyperedges are the following:

V_1	V_2	V_3
x_1	●	$\bar{x}_3$
●	x_2	$\bar{x}_3$
$\bar{x}_1$	x_2	●
$\bar{x}_1$	●	x_3
●	$\bar{x}_2$	x_3
x_1	$\bar{x}_2$	●

- For each clause c_i, we add a single *global* hyperedge which contains the vertices corresponding to the variables appearing in c_i, while taking into account their position in the clause and whether they are negated. For instance, if $c_i = \neg z \vee x \vee y$, we add a hyperedge $(\bar{z}_1, x_2, y_3)$. This ensures that the hypergraph remains tripartite.

This reduction is polynomial. Let m be the cardinality of $\mathbf{x}$. We now show that φ is satisfiable if and only if there is a vertex cover in $\mathscr{H}$ of size less than or equal to $3m$ (or, equivalently, if there is a minimum vertex cover of size less than or equal to $3m$).

Suppose first that φ is satisfiable, and let ν be a valuation of $\mathbf{x}$ which satisfies φ. Let us consider the following set S of vertices of $\mathscr{H}$: For each $x \in \mathbf{x}$, we add to S, x_1, x_2 and x_3 if $\nu(x)$ is true, $\bar{x}_1$, $\bar{x}_2$ and $\bar{x}_3$ otherwise. S is of cardinality $3m$. Observe that S covers all local hyperedges and, since ν satisfies φ, all global hyperedges.

Suppose now that there is a minimum vertex cover S of size less than or equal to $3m$. Since anonymous vertices only appear in a single hyperedge, we can always assume that S does not contain any anonymous vertex (they can always be replaced by another vertex of the hyperedge). Let S_i be, for each $1 \leqslant i \leqslant m$, the subset of S_i containing only the vertices corresponding to the ith variable of $\mathbf{x}$. It is easy to see that $|S_i| \geqslant 3$ for all i, for all local hyperedges to be covered, which means that $|S_i| = 3$ since $\left|\bigcup S_i\right| \leqslant 3m$. S_i forms a vertex cover of the local sub-hypergraph corresponding to the ith variable of $\mathbf{x}$ (let us call it x) and must cover the hyperedges of this sub-hypergraph. But there are only two vertex covers of this sub-hypergraph of cardinality 3: Either S_i contains all x_k's, or it contains all $\bar{x}_k$'s. We consider the valuation ν of the variables in $\mathbf{x}$ which maps x to true in the first case, to false in the second. Then, since S is a vertex cover of $\mathscr{H}$, ν satisfies all the clauses of φ. □

We now use this lemma to prove the **NP**-hardness of $\textsc{Cost}_{\text{facyc}}$.

Proposition 5.15. $\textsc{Cost}_{\text{facyc}}$ *is NP-hard.*

Proof. We first consider the case where we only allow negated equalities $x \neq c$, and no equalities $x = c$, on the left-hand side of repairs of tgds, with x a universally quantified variable, as the proof is clearer. We detail then the changes that have to be made in the general case.

We reduce the vertex cover problem in tripartite 3-uniform hypergraphs to $\textsc{Cost}_{\text{facyc}}$. Let $\mathscr{H}$ be a tripartite 3-uniform hypergraph. We consider the following instance of $\textsc{Cost}_{\text{facyc}}$:

- $\mathbf{S} = \{(R, 3)\}$ and R^I is the representation of $\mathscr{H}$ as a three-column table, where each row corresponds to an edge, and each column to one of the set of the tripartition of $\mathscr{H}$;
- $\mathbf{T} = \{(R', 1)\}$ and $J = \varnothing$;
- $\Sigma = \{\forall x_1 \forall x_2 \forall x_3\, R(x_1, x_2, x_3) \rightarrow R'(x_1)\}$ (this is obviously an acyclic tgd).

As $J=\varnothing$, any schema mapping fully explains J. This also means that the only repairs of Σ to be considered are the ones that add a "$x_i \neq c_i$" term to the left-hand side of the single element of Σ. A repair of Σ has to "cancel" somehow with these additions each tuple of R^I. In other words, the cost of Σ is $\mathsf{size}(\Sigma)+2r$, where r is the minimal number of conjuncts in a formula of the form $\bigwedge x_i \neq c_i$, such that this formula is false for all tuples of R^I. Such a formula expresses a vertex cover in $\mathscr{H}$, and $\mathscr{H}$ has a vertex cover of size less than or equal to K if and only if $\mathsf{cost}_{(I,J)}(\Sigma) \leqslant \mathsf{size}(\Sigma)+2K$, which concludes the proof in this case.

In the general case where we allow arbitrary repairs, including $x=c$ terms on the left-hand side of a tgd, the same proof does not work, since it suffices to choose an arbitrary constant c which does not appear in I for the term $x_1=c$ to cancel every tuple of R^I. To fix this, we need to make prohibitive the addition of such a term to Σ in the following way: Let c', and, for $1 \leqslant i \leqslant 3$, $1 \leqslant j \leqslant K$, $c_{i,j}$ be $3K+1$ fresh constants. We can always assume that K is linear in the size of the input of $\textsc{Cost}_{\text{facyc}}$ (otherwise, just replace K with the upper bound of Proposition 5.7). We consider the following slightly modified instance of $\textsc{Cost}_{\text{facyc}}$:

- $\mathrm{S}=\{(R,3)\}$ and $R^I=A\cup B$, where A is the representation of $\mathscr{H}$ as a three-column table, where each row corresponds to an edge, and each column to one of the set of the tripartition of $\mathscr{H}$, and B is the set:

$$\left\{R(c_{1j},c',c') \mid 1 \leqslant j \leqslant K\right\} \cup \left\{R(c',c_{2j},c') \mid 1 \leqslant j \leqslant K\right\} \cup \left\{R(c',c',c_{3j}) \mid 1 \leqslant j \leqslant K\right\};$$

- $\mathrm{T}=\left\{(R',3)\right\}$ and J is the same as B if R is replaced by R';
- $\Sigma=\left\{\forall x_1 \forall x_2 \forall x_3\ R(x_1,x_2,x_3) \rightarrow R'(x_1,x_2,x_3)\right\}$ (this is obviously an acyclic tgd).

This reduction is polynomial if K is linear in the size of the input, as assumed. Σ fully explains J and the repairs considered in the previous case do not change this.

Let now $x_i=c$ be a term with $1 \leqslant i \leqslant 3$ and c some constant. Whatever the choice of c, the addition of this term to the tgd of Σ cancels at least K tuples of B, and hence, fails to explain at least K tuples of J (the best case is when $c=c'$). Just observe that, as the cost of a ground fact, which is the only way to repair unexplained tuples, is 9, the size of any repair of Σ with such an $x_i=c$ term on the left-hand side is greater than or equal to $\mathsf{size}(\Sigma)+9K$. We keep the fact that $\mathsf{cost}_{(I,J)}(\Sigma) \leqslant \mathsf{size}(\Sigma)+2K$ if and only if $\mathscr{H}$ has a vertex cover of size less than or equal to K. □

It is an open issue whether $\textsc{Cost}$ is in **PTIME** for the very restricted case when the schema mapping consists of a single full tgd with a single binary relation symbol appearing once in the left-hand side.

5.4.5 Combined Complexity for Acyclic Tuple-Generating Dependencies

The last subset of $\mathscr{L}_{\text{tgd}}$ that we consider here is $\mathscr{L}_{\text{acyc}}$.

Proposition 5.16.

1. $\textsc{Validity}_{\text{acyc}}$ *is* ***coNP-complete.***
2. $\textsc{Explanation}_{\text{acyc}}$ *and* $\textsc{Zero-Repair}_{\text{acyc}}$ *are* ***NP-hard.***
3. $\textsc{Explanation}_{\text{acyc}}$ *is in* ***PTIME*** if, for all existentially quantified variables y and for all constants c, there is at most one term $y=c$ appearing in each formula of the schema mapping. *This is in particular the case if the schema mapping is a subset of $\mathscr{L}_{\text{acyc}}$ instead of $\mathscr{L}^*_{\text{acyc}}$.*

Proof.

1. • Let us first prove that $\textsc{Validity}_{\text{acyc}}$ is in **coNP**. The validity of ground facts of Σ is trivial to check. Let θ be a formula of Σ which is not a ground fact. Recall that θ is of the form
$$\forall \mathbf{x}\ \varphi(\mathbf{x}) \wedge \tau(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi(\mathbf{x},\mathbf{y}) \wedge \tau'(\mathbf{x},\mathbf{y})$$
with $\tau(\mathbf{x})$ a conjunction of terms expressing equality or negated equality between a variable of $\mathbf{x}$ and a constant, and $\tau'(\mathbf{x},\mathbf{y})$ of the form
$$\tau'(\mathbf{x},\mathbf{y}) = \bigwedge_{i=1}^{n} \left(\left(\bigwedge_{j=1}^{m_i} x_{ij} = c'_{ij} \right) \rightarrow y_i = c_i \right).$$
Guess a valuation ν of the variables of $\mathbf{x}$. If the left-hand side of θ is made false by this valuation, give up this guess. Otherwise, consider the formula
$$\xi = \exists \mathbf{y}\ \psi(\nu(\mathbf{x}),\mathbf{y}) \wedge \tau'(\nu(\mathbf{x}),\mathbf{y})$$
where $\tau'(\nu(\mathbf{x}),\mathbf{y})$ is equivalent to a conjunction of terms of the form $y = c$ with $y \in \mathbf{y}$ and c a constant. We can then check in polynomial time if J satisfies ξ using Algorithm 5.3.

   • To prove that $\textsc{Validity}_{\text{acyc}}$ is **coNP**-hard, we use a reduction of 3SAT. Let $\varphi = \bigwedge_{i=1}^{n} c_i(z_{i1}, z_{i2}, z_{i3})$ be an instance of 3SAT, where the c_i are 3-clauses over some set $\mathbf{x}$ of variables (and $\bigcup_{i,j} z_{ij} = \mathbf{x}$). We consider now the following instance of $\textsc{Validity}_{\text{acyc}}$:
     – $\mathbf{S} = \{(B,1)\}$ and $I = \{B(0), B(1)\}$;
     – $\mathbf{T} = \{(R,1)\}$ and $J = \{R(1)\}$;
     – For all $1 \leqslant i \leqslant n$, let τ_i be the unique conjunction of the form $z_{i1} = b_1 \wedge z_{i2} = b_2 \wedge z_{i3} = b_3$ such that each b_k is either the constant 0 or 1, and $c_i(z_{i1}, z_{i2}, z_{i3})$ is false if this conjunction holds. We then define:
$$\Sigma = \left\{ \forall x_1 \ldots \forall x_m\ B(x_1) \wedge \cdots \wedge B(x_m) \rightarrow \exists y_1 \ldots \exists y_n\ R(y_1) \wedge \cdots \wedge R(y_n) \wedge \bigwedge_{i=1}^{n} (\tau_i \rightarrow y_i = 0) \right\}.$$
This transformation is polynomial, $\Sigma \subset \mathscr{L}*_{\text{acyc}}$, and Σ is valid with respect to (I,J) if and only if the original 3SAT instance is not satisfiable.

2. We use a reduction from 3SAT very similar to that of item 1. This reduction works both for $\textsc{Explanation}_{\text{acyc}}$ and $\textsc{Zero-Repair}_{\text{acyc}}$.
   • $\mathbf{S} = \{(B,1)\}$ and $I = \{B(0), B(1)\}$;
   • $\mathbf{T} = \{(R,n)\}$ and $J = \{R(1,\ldots,1)\}$;
   • For all $1 \leqslant i \leqslant n$, let $\tau_{i1} \ldots \tau_{i8}$ be the eight conjunctions of the form $z_{i1} = b_1 \wedge z_{i2} = b_2 \wedge z_{i3} = b_3$ such that each b_k is either the constant 0 or 1, and $c_i(z_{i1}, z_{i2}, z_{i3})$ is true if this conjunction holds. We then define:

$$\Sigma = \left\{ \forall x_1 \dots \forall x_m \, B(x_1) \wedge \dots \wedge B(x_m) \rightarrow \exists y_1 \dots \exists y_n \, R(y_1 \dots y_n) \right.$$

$$\left. \wedge \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{8} \left(\tau_{ij} \rightarrow y_i = 1 \right) \right\}.$$

Observe that Σ is valid with respect to (I,J). This transformation is polynomial, and Σ fully explains J with respect to I if and only if the original 3SAT instance is satisfiable.

3. Let f be a fact of J, we describe a polynomial algorithm for deciding whether f is explained by Σ. First, check if f is in the ground facts of Σ. Otherwise, for each atom $R(z_1 \dots z_k)$ in the right-hand side of a formula $\theta \in \Sigma$ which is not a ground fact, such that R is the relation name appearing in f, do the following. We keep the same notations as above for the sub-formulas of θ.

   Observe that f is explained by the $R(z_1 \dots z_k)$ atom of θ if and only if there is a valuation ν of the variables of $\mathbf{x}$ such that $\nu(\varphi(\mathbf{x}) \wedge \tau(\mathbf{x}))$ is true in I and, for all extensions ν' of ν to $\mathbf{x} \cup \mathbf{y}$, $\nu'\left(\tau'(\mathbf{x},\mathbf{y})\right)$ is true and $\nu'(R(z_1 \dots z_k)) = f$.

   This means that all variables of $\mathbf{y}$ appearing in $R(z_1 \dots z_k)$ must also appear as the right-hand side of an implication of $\tau'(\mathbf{x},\mathbf{y})$ (otherwise, an extension ν' of ν such that $\nu'(R(z_1 \dots z_k))$ is not equal to f is possible). By the hypothesis we made, there is exactly one conjunct of $\tau'(\mathbf{x},\mathbf{y})$ where each variable of $\mathbf{y}$ appears. Let $\rho(\mathbf{x})$ be the conjunction of the left-hand sides of these implications for all variables of $\mathbf{y}$. Then, f is explained by the $R(z_1 \dots z_k)$ atom of θ if and only if the boolean query $\exists \mathbf{x} \, \varphi(\mathbf{x}) \wedge \tau(\mathbf{x}) \wedge \rho(\mathbf{x})$ matches the instance I. As $\tau(\mathbf{x}) \wedge \rho(\mathbf{x})$ is a simple conjunction, we can first perform the corresponding selection on I, and then use Algorithm 5.3 to decide if the boolean query matches. □

Note that we used for both hardness results the repairs themselves to encode the instance of a **NP**-hard problem: Although the tgd itself is acyclic, its repairs are not. We could probably get polynomial algorithms for the same problems if we impose some acyclicity condition to repairs of a formula; this, however, would weaken our notion of optimality.

5.4.6 Combined Complexity of Existence-Cost and Optimality

With the help of Lemma 5.14, we show the intractability of Existence-Cost and Optimality, in all considered languages:

Proposition 5.17. Existence-Cost *(respectively,* Optimality*) is* ***NP****-hard (respectively, both* ***NP****-hard and* ***coNP****-hard) in all the following languages:* $\mathscr{L}_{\text{tgd}}$, $\mathscr{L}_{\text{full}}$, $\mathscr{L}_{\text{acyc}}$, $\mathscr{L}_{\text{facyc}}$.

Proof. To prove this result, we use, as in the proof of Proposition 5.15, a reduction from the vertex cover problem in tripartite 3-uniform hypergraphs. The core of the reduction is the same for Existence-Cost and Optimality.

Let $\mathscr{H} = (V,E)$ be a tripartite 3-uniform hypergraph with N vertices and K an integer. We denote by $\tau(\mathscr{H})$ the minimum size of a vertex cover in $\mathscr{H}$. Let $\alpha \geqslant 1$ an integer, to be defined further. Each vertex of V is seen as a constant of $\mathscr{C}$. We also use $3(\alpha+N)$ additional constants c_{ik} and c'_{jk} with $1 \leqslant i \leqslant \alpha$, $1 \leqslant j \leqslant N$, $1 \leqslant k \leqslant 3$. We now consider the following schemata and instances:

- $\mathbf{S} = \{(R,3)\}$ and
$$\begin{aligned} I = &\left\{ R(c_{i1},c_{i2},c_{i3}) \mid 1 \leqslant i \leqslant \alpha \right\} \\ &\cup \left\{ R(c'_{i1},c'_{i2},c'_{i3}) \mid 1 \leqslant i \leqslant N \right\} \\ &\cup \left\{ R(v,v',v'') \mid e=(v,v',v'') \in E \right\}; \end{aligned}$$
- $\mathbf{T} = \{(R',1),(S',3)\}$ and
$$\begin{aligned} J = &\left\{ R'(c_{i1}) \mid 1 \leqslant i \leqslant \alpha \right\} \\ &\cup \left\{ S'(c_{i1},c_{i2},c_{i3}) \mid 1 \leqslant i \leqslant \alpha \right\} \\ &\cup \left\{ S'(c'_{i1},c'_{i3},c'_{i3}) \mid 1 \leqslant i \leqslant N \right\}. \end{aligned}$$

Let $\Sigma_0 = \{\forall \mathbf{x}\, R(\mathbf{x}) \to S'(\mathbf{x})\}$. Σ_0 is valid, but fails to explain α tuples of J. Thus, $\mathsf{cost}_{(I,J)}(\Sigma_0) = 6+3\alpha$ (since the cost of a ground fact of arity 1 is 3, see Definition 5.5).

Let $\Sigma = \{\forall \mathbf{x}\, R(\mathbf{x}) \to R'(x_1) \wedge S'(\mathbf{x})\}$ ($\Sigma \subset \mathscr{L}_{\text{facyc}}$). An argument very similar to the one of the proof of Proposition 5.15 shows that

$$\mathsf{cost}_{(I,J)}(\Sigma) = \mathsf{size}(\Sigma) + 2\tau(\mathscr{H}) + ZN = 7 + 2\tau(\mathscr{H}) + 2N.$$

since S'^J guarantees that the size of any repair of Σ with an $x=c$ term on the left-hand side is at least:

$$\mathsf{size}(\Sigma) + 2 + 9(\alpha+N-1) > \mathsf{size}(\Sigma) + 9N > \mathsf{cost}_{(I,J)}(\Sigma).$$

The idea now is to choose α such that $\tau(H) \leqslant K$ if and only if $\mathsf{cost}_{(I,J)}(\Sigma) \leqslant \mathsf{cost}_{(I,J)}(\Sigma_0)$. This is the case if:

$$\alpha = \frac{2(K+N)+1}{3}.$$

With this value of α, we have $\mathsf{cost}_{(I,J)}(\Sigma_0) = 2(K+N)+7$ and $\mathsf{cost}_{(I,J)}(\Sigma) = 2\tau(\mathscr{H}) + N + 7$, which yields the property we were looking for. However, $2(K+N)+1$ may not be divisible by 3; in this case, we just transform the initial problem by observing that $\tau(\mathscr{H}) = K$ if and only if $\tau(\mathscr{H}') = K$ where $\mathscr{H}'$ is the tripartite 3-uniform hypergraph obtained from $\mathscr{H}$ by adding n new edges, each of which span 3 new vertices (this does not change the value of N mod 3). Up to such a transformation, we may then assume than $2(K+N)+1$ is divisible by 3.

Let us now show that, for any schema mapping $\Sigma' \subset \mathscr{L}_{\text{tgd}}$,

$$\mathsf{cost}_{(I,J)}(\Sigma') \geqslant \min\left(\mathsf{cost}_{(I,J)}(\Sigma_0), \mathsf{cost}_{(I,J)}(\Sigma)\right). \tag{5.1}$$

This will conclude the proof, since we then have the following reductions, obviously polynomial and valid for any of the considered languages:

NP-hardness of Existence-Cost. $\tau(\mathscr{H}) \leqslant K$ if and only if there exists a schema mapping whose cost with respect to (I,J) is lesser than or equal to $\mathsf{cost}_{(I,J)}(\Sigma_0) - 1$.

NP-hardness of Optimality. $\tau(\mathcal{H}) \leqslant K$ if and only if Σ is optimal with respect to (I,J).

coNP-hardness of Optimality. $\tau(\mathcal{H}) \leqslant K-1$ if and only if Σ_0 is not optimal with respect to (I,J).

Let Σ' be a non-empty schema mapping of $\mathcal{L}_{\text{tgd}}$. In order to prove (5.1), first observe that, as the constants c_{i1} are completely indistinguishable from each other, Σ' must either explain all or none of the facts of R'^J. We shall consider each case in turn.

- If Σ' does not explain any of the facts of R'^J, each of these must be accounted for in the repairs, by one of the following methods:
  - adding ground facts (additional cost: 3 each);
  - adding an unconditioned "$x=c$" term to a $R'(x)$ atom where x is existentially quantified (additional cost: 2 each, but this can only be done once per $R'(x)$ atom, whose size is 1, or this yields an inconsistent formula);
  - adding a conditioned "$\tau \to x=c$" term to a $R'(x)$ atom where x is existentially quantified (minimum additional cost: 4 each, since the size of τ is at least 2).

  Moreover, a repair of Σ' should also account for the facts of S'^J, either as explanations of Σ' (this cannot be done in a formula with size lesser than 6) or by enumerating all ground facts of S' (with a cost of $9(\alpha+N)$, which is greater than 6). This means that $\mathsf{cost}_{(I,J)}(\Sigma') \geqslant 3\alpha+6 = \mathsf{cost}_{(I,J)}(\Sigma_0)$.

- In the case where Σ' explains all facts of R'^J, there is a tgd $\theta \in \Sigma'$ such that θ explains all facts of $R'J$, and θ is necessarily of the form:

  $$\forall x_1 \forall x_2 \forall x_3 \forall \mathbf{u}\, R(x_1,x_2,x_3) \wedge \varphi(x_1,x_2,x_3,\mathbf{u}) \to \exists \mathbf{v}\, R'(x_1) \wedge \psi(x_1,x_2,x_3,\mathbf{u},\mathbf{v})$$

  with $\exists \mathbf{u}\, \varphi(c_{11},c_{12},c_{13},\mathbf{u})$ valid in I. Then, for all $e=(v,v',v'') \in E$,

  $$\exists \mathbf{u}\, \varphi(v,v',v'',\mathbf{u})$$

  is valid in I since φ does not contain anything else than relation atoms $R(w_1,w_2,w_3)$ with all x_i's necessarily in the ith position, and other variables existentially quantified. That means that $R'(v)$ is an incorrect fact implied by the tgd. As we saw earlier, adding an $x=c$ term on the left-hand side of θ has prohibitive cost. The only way to cancel these facts is then as in the proof of Proposition 5.15. Finally, a repair of θ must also explain all facts of S'^J, either as facts explained by θ itself (then, $\mathsf{size}(\varphi) \geqslant 3$), or by enumerating ground facts of S'^J, with a cost of $9(\alpha+N) > 3$. We have therefore:

  $$\mathsf{cost}_{(I,J)}(\Sigma') \geqslant \mathsf{cost}_{(I,J)}(\theta) \geqslant 7 + 2\tau(\mathcal{H}) + 2N = \mathsf{cost}_{(I,J)}(\Sigma). \qquad \square$$

Note that the same proof does not work in the case of $\mathcal{L}_{\text{rc}}$:

$$\begin{aligned}&\{\forall x_1 \forall x_2 \forall x_3\, R(x_1,x_2,x_3) \wedge \\ &\quad \neg \left(\exists x_2' \exists x_3'\, R(x_1,x_2',x_3') \wedge (x_2 \neq x_2' \vee x_3 \neq x_3')\right) \to R'(x)\}\end{aligned}$$

may have a lower cost for some instances than Σ (for instance if $\mathcal{H}$ is a hypergraph where all nodes have a degree greater than 1).

The other results from Table 5.2 are direct consequences of the former propositions.

Table 5.3: Data complexity results

		$\mathscr{L}_{\text{tgd}}, \mathscr{L}_{\text{full}}, \mathscr{L}_{\text{acyc}}, \mathscr{L}_{\text{facyc}}$
Validity	(Σ fixed)	**PTIME**
Explanation	(Σ fixed)	**PTIME**
Zero-Repair	(Σ fixed)	**PTIME**
Cost	(K fixed)	**PTIME**
Cost	(Σ fixed)	**NP**, **NP**-hard for some Σ
Existence-Cost	(K fixed)	**PTIME**
Optimality	(Σ fixed)	$\mathbf{\Pi}_2^P$, (**co**)**NP**-hard for some Σ

5.4.7 Data Complexity

As far as data complexity is concerned, the situation is simpler, since we do not have any difference in complexity for all four subsets of $\mathscr{L}_{\text{tgd}}$. The results are presented in the next proposition, and they are summarized in Table 5.3.

Proposition 5.18.

1. *If Σ is fixed,* Validity$_{\text{rc}}$ *is in **PTIME**.*

2. *If Σ is fixed,* Explanation$_{\text{tgd}}$ *is in **PTIME**.*

3. *If K is fixed,* Cost$_{\text{tgd}}$ *and* Existence-Cost$_{\text{tgd}}$ *are in **PTIME**.*

4. *For some fixed value of Σ,* Cost$_{\text{facyc}}$ *is **NP**-hard.*

5. *For some fixed value of Σ,* Optimality$_{\text{facyc}}$ *and* Optimality$_{\text{tgd}}$ *are both **NP**-hard and **coNP**-hard.*

Proof.

1. If k is the number of quantified variables in a first-order formula φ in prenex normal form, it is easy to see that checking whether φ is valid in a database of size n is $O(n^k)$.

2. Each formula of Σ is either a ground fact, or of the form $\forall \mathbf{x}\,(\varphi(\mathbf{x}) \land \tau(\mathbf{x})) \rightarrow \exists \mathbf{y}\,(\psi(\mathbf{x},\mathbf{y}) \land \tau'(\mathbf{x},\mathbf{y}))$ with τ and τ' propositional combinations of terms expressing equalities between a variable and a constant. For each fact f of J, first check if it appears as a ground fact Σ; otherwise, for each valuation of $\mathbf{x} \cup \mathbf{y}$ (there is a constant number of such valuations, since Σ is fixed), check whether the left-hand side holds, and f is a consequence of the right-hand side.

3. We can just enumerate all schema mappings of $\mathscr{L}^*_{\text{tgd}}$ whose size is lower than K, and check in polynomial time if they are valid and fully explain the target instance.

4. This results from the proof of Proposition 5.15.

5. This results from the proof of Proposition 5.17. □

5.5 Extension and Variants

We study in this section some extensions of our optimality notion to (i) formulas of the full relational calculus; (ii) other apparently simpler functions of cost.

5.5.1 Extension to the Relational Calculus

We can extend the definitions of Section 5.2 to the language $\mathscr{L}_{\mathrm{rc}}$ of relational calculus, by the following adaptation of the notion of repair.

A repair of a schema mapping $\Sigma \subset \mathscr{L}_{\mathrm{rc}}$ is a set of formulas obtained from Σ by a finite sequence of the following operations:

- Replacing in a formula of Σ a sub-formula $\forall \mathbf{x}\ \varphi(\mathbf{x},\mathbf{y})$ (we also assume that φ does not start with a $\forall$ symbol, and that the sub-formula is not preceded by a $\forall$ symbol) by $\forall \mathbf{x}\ \tau(\mathbf{x},\mathbf{z}) \rightarrow \varphi(\mathbf{x},\mathbf{y})$ where $\mathbf{z}$ is the set of variables free in φ and τ is a boolean formula over terms $w = c$ of the following form:
$$\bigwedge_i \left(\left(\bigwedge_j z_{ij} = c'_{ij} \right) \rightarrow x_i \alpha_i c_i \right)$$
with z_{ij} variables from $\mathbf{z}$, x_i variables from $\mathbf{x}$, α_i either $=$ or $\neq$, and c'_{ij} and c_i constants.

- Replacing in a formula of Σ a sub-formula $\exists \mathbf{x}\ \psi(\mathbf{x},\mathbf{y})$ (we also assume that ψ does not start with a $\exists$ symbol, and that the sub-formula is not preceded by a $\exists$ symbol) by $\exists \mathbf{x}\ \psi(\mathbf{x},\mathbf{y}) \land \tau'(\mathbf{x},\mathbf{z})$ where $\mathbf{z}$ is the set of variables free in ψ and τ' is a boolean formula over terms $w = c$ of the following form:
$$\bigwedge_i \left(\left(\bigwedge_j z_{ij} = c'_{ij} \right) \rightarrow x_i = c_i \right)$$
with z_{ij} variables from $\mathbf{z}$, x_i variables from $\mathbf{x}$, and c'_{ij} and c_i constants.

- Adding to Σ a ground fact $R(c_1 \dots c_n)$ where R is a relation of the target schema of arity n, and $c_1 \dots c_n$ are constants.

We can check that this definition amounts to the same as Definition 5.4 if we restrict ourselves to $\mathscr{L}_{\mathrm{tgd}}$. We can then use the same definitions of the size and cost of a formula, and consider the same decision problems. We have the following complexity results:

Proposition 5.19.

1. Validity$_{\mathrm{rc}}$ *is PSPACE-complete;*

2. Explanation$_{\mathrm{rc}}$ *is co-recursively enumerable;*

3. Explanation$_{\mathrm{rc}}$ *and* Zero-Repair$_{\mathrm{rc}}$ *are not recursive.*

Proof.

1. • Let us first show that $\textsc{Validity}_{\mathrm{rc}}$ is in **PSPACE**. Let $\varphi \in \Sigma$. If φ is a ground fact, this is trivial. Otherwise, we first rewrite φ in prenex normal form

$$\alpha_1 x_1 \alpha_2 x_2 \ldots \alpha_n x_n \ \psi(x_1, \ldots, x_n)$$

where each α_i is either $\exists$ or $\forall$.

Let C be the set of constants appearing in I and J, along with n distinct constant $\bot_1 \ldots \bot_n$. It can be shown that we do not need to consider valuations of the $x_1 \ldots x_n$ into other constants. For each valuation ν of $x_1 \ldots x_n$ into C, it is decidable in polynomial time whether $(I,J) \models \psi(\nu(x_1),\ldots,\nu(n_n))$. We then enumerate all valuations, enumerating recursively all valuations of x_{i+1} while keeping x_i fixed, and remembering for each $1 \leqslant i \leqslant n+1$ a single value ok_i which is equal to:

   – If $i = n+1$: 1 if $(I,J) \models \psi(\nu(x_1),\ldots,\nu(n_n))$ in the current valuation ν, 0 otherwise;
   – If $\alpha_i = \exists$: the maximum of ok_{i+1} and the preceding value of ok_i (and the preceding value is reset to 0 whenever the valuation of x_{i-1} is changed).
   – If $\alpha_i = \forall$: the preceding value of ok_i, multiplied par the current value of ok_{i+1} (and the preceding value is reset to 1 whenever the valuation of x_{i-1} is changed).

The algorithm returns `true` if, after enumerating all valuations, all ok_i are equal to 1. Otherwise, the algorithm returns `false`. This is obviously a **PSPACE** algorithm, and it returns `true` if and only if $(I,J) \models \varphi$. We can run the same algorithm in sequence on all $\varphi \in \Sigma$.

   • The **PSPACE**-hardness of $\textsc{Validity}_{\mathrm{rc}}$ comes from a polynomial-time reduction of QSAT (also known as QBF), which is **PSPACE**-complete [Pap94, SM73]. Let $\varphi = \alpha_1 x_1 \ldots \alpha_n x_n \ \psi(x_1 \ldots x_n)$ be an instance of QSAT (the α_i are either $\exists$ or $\forall$ symbols, and $\psi(x_1 \ldots x_n)$ is a propositional formula with variables $x_1 \ldots x_n$).

Let $\mathbf{S} = \{(R,1)\}$,$\mathbf{T} = \{(R',1)\}$, $I = \{R(a)\}$ and $J = \{R'(a)\}$. Let Σ be the singleton:

$$\{\forall y \alpha_1 x_1 \ldots \alpha_n x_n \ \psi(R(x_1) \ldots R(x_n)) \wedge R(y) \rightarrow \neg R'(y)\}.$$

Then, $(I,J) \models \Sigma$ if and only if φ is true.

2. To see that $\textsc{Explanation}_{\mathrm{rc}}$ is co-recursively enumerable, just enumerate all instances K of the target schema, and whenever they are such that $(I,J) \models \Sigma$ (which is decidable, see just above), see if they contain K. If this is not the case, we can conclude that Σ does not fully explain J with respect to I.

3. The uncomputability comes from a reduction of the satisfiability of the relational calculus, which is not recursive [Tra63, DP69]. The reduction is the same for both problems $\textsc{Explanation}_{\mathrm{rc}}$ and $\textsc{Zero-Repair}_{\mathrm{rc}}$. Let φ be a formula of the relational calculus over a schema $\mathbf{U}$. Consider the following instance of $\textsc{Explanation}_{\mathrm{rc}}$ (this is also an instance of $\textsc{Zero-Repair}_{\mathrm{rc}}$):

   • $\mathbf{S} = \{(R,1)\}$ and $I = \{R(a)\}$;
   • $\mathbf{T} = \mathbf{U} \cup \{(R',1)\}$ and $J = \{R'(a)\}$;
   • $\Sigma = \{\forall x \, R(x) \rightarrow \varphi \vee R'(x)\}$

Observe that $(I,J) \models \Sigma$, therefore (I,J,Σ) is a solution of $\textsc{Explanation}_{\mathrm{rc}}$ if and only if it is a solution of $\textsc{Zero-Repair}_{\mathrm{rc}}$.

Now, (I,J,Σ) is a solution of $\textsc{Explanation}_{\mathrm{rc}}$ if and only if, for all K such that $(I,K) \models \Sigma$, $R'(a)$ is a fact of K. This is the case if and only if φ is not satisfiable. □

Interestingly, the computability of $\textsc{Optimality}_{\mathrm{rc}}$ remains open. It seems to be a "harder" problem than $\textsc{Zero-Repair}_{\mathrm{rc}}$, but as there is no simple reduction between them, we cannot even be sure that $\textsc{Optimality}_{\mathrm{rc}}$ is not recursive. We do not even know whether it is recursively enumerable or co-recursively enumerable (although $\textsc{Cost}_{\mathrm{rc}}$ and $\textsc{Existence-Cost}_{\mathrm{rc}}$ both are co-recursively enumerable because of the co-recursive enumerability of $\textsc{Zero-Repair}_{\mathrm{rc}}$).

5.5.2 Variants of the Cost Function

The definition of repairs and cost that we presented in Section 5.2 may appear, at first, unnecessarily complicated. We argued in Section 5.3 for a justification of this notion by showing that it has nice properties with respect to instances that are derived from each other with elementary operations of the relational algebra. We consider in this section two alternative definitions of cost and optimality of a schema mapping with respect to database instances, and show that neither, although simpler and perhaps more intuitive, present the same properties and are thus adapted to our context.

We keep our notions of validity of a schema mapping, of full explanation of a database instance by a schema mapping, and of size of a schema mapping, and we want to consider alternative ways to characterize the *cost* of a given schema mapping. The first idea is to assign as the cost of a schema mapping the minimal number of tuples that have to be added or removed to the target instance J for the schema mapping to become valid and to fully explain J. (Each tuple may also be weighted by its arity, to get something closer to our original cost definition.) Thus, the cost of the empty schema mapping corresponds to the size of the target instance, as before, while the cost of a schema mapping that fully explains the target instance but also incorrectly explains some tuples is the (possibly weighted) number of such tuples. This sounds like a reasonable definition, but it presents an important problem: We lose the linear bound on the cost of a schema mapping in the size of the data and the schema mapping itself. Indeed, consider the following schema mapping, for a given n, where R is a source relation of arity 1 and R' a target relation of arity n:

$$\left\{\forall x_1 \dots \forall x_n\, R(x_1) \wedge \dots \wedge R(x_n) \to R'(x_1,\dots,x_n)\right\}.$$

If J is empty, the cost of this schema mapping according to the definition given in this paragraph is $|I|^n$ (or $n|I|^n$ if we weight with the arity of the relations), which is exponential in the size of the schema mapping. This combinatorial explosion discourages all hopes of getting an optimal schema mapping by local search techniques. Besides, all the problems that we describe for the variant that we consider next also arise here.

An alternate definition of cost, close to the previous one but for which we still have a linear bound on the cost of a schema mapping is the following: The cost of a schema mapping Σ is the minimal number of tuples to add or remove from the source instance I and of tuples to add or remove to the target instance J so that Σ becomes valid and fully explains J. As before, we assume that we weight tuples by their arity; we could also choose to add an arbitrary constant weight to operations on J with respect to operations on I, or to deletion with respect to addition of tuples, without much difference in the model. The linear bound is clear, then, since we can just remove all tuples of I and of J for Σ to be valid and to fully explain J. However, there is still a fundamental problem with this definition, which can be seen by looking back at Section 5.3. We showed there that, for elementary operations of the relational algebra, the definition of optimality of Section 5.2 yielded the same as the intuitive tgds expressing these operations. This is not true any more here, however, in particular in the presence of selections and projections. For projections, this is due to the fact that a schema mapping that predicts existentially quantified tuples has a higher cost than the same schema mapping where these existentially quantified relation atoms are removed. We exhibit next a concrete example of database instances that illustrate the problem with selections.

Example 5.20. Let us consider instances I and J of the following schemata:

$$\mathbf{S} = \{(P,2)\} \qquad\qquad \mathbf{T} = \{(P',1)\}$$

where:

- I contains a list of titles of publications (as first attribute) along with their kind: `article`, `book`, `report`, etc.
- J contains a list of book titles.

Let us assume that J and I contain the same book titles. In other words, $J = \pi_1(\sigma_{2=\mathtt{book}}(I))$. It is quite natural to expect $\Sigma = \{\forall x \forall y\, P(x,y) \to P'(x)\}$ as the "optimal" schema mapping in the language of tgds for these database instances, and indeed,

$$\mathsf{cost}_{(I,J)}(\Sigma) = \mathsf{size}(\forall x \forall y\, P(x,y) \land y = \mathtt{book} \to P'(x)) = 5$$

is minimal as soon as J is large enough and there is no hidden relation between the second attribute of I and J.

Now, observe that with the variant proposed in the preceding paragraph, the cost will be:

$$3 + \min(2(|I| - |J|), |J|),$$

which is, in all cases when there are more publications of another kind than `book` (a common situation), greater than the cost of the empty schema mapping, which is then the optimal schema mapping for these instances.

Then, although our definition of optimality is a bit complex, it is much more adapted to what we consider here than these simpler definitions, since it can capture such things as the worth of an existentially quantified relation atom, or the possibility of limiting the scope of a tgd with the addition of a simple selection to which facts of the source instance it applies.

Conclusion

We discussed a theoretical framework that addresses the problem of finding a schema mapping *optimal* with respect to a pair of database instances, based solely on the structure and occurrences of constants in the instances. We showed that this problem is of a high complexity, being both **NP**-hard and **coNP**-hard even for a very restricted language, namely full acyclic tuple-generating dependencies. There are a number of open theoretical issues, especially on the computability and precise complexity of OPTIMALITY for various languages, but the most obvious direction for future work would be to connect such a theoretical framework with practical heuristics and approximation algorithm; in particular, the relation to inductive logic programming has to be explored. We believe that this is an especially important problem, and that discovering and understanding hidden relations in data is one of the most fundamental task of artificial intelligence. Other theoretical problems of interest would be to improve our complexity upper bounds by generalizing the notion of acyclicity to that of bounded hypertree width [GLS99], and to look at the same problems when some fixed set of preconditions or constraints on source and target instances is given.

Chapter 6

Semantic Model and Query Processing

FETCH, SNOOPY!
"FETCH?"
FETCH (fech) 1. TO GO AFTER AND COME BACK WITH; BRING; GET. 2. TO CAUSE TO COME; PRODUCE.
TO GO AFTER THINGS, AND BRING THEM BACK; HENCE...TO RETRIEVE GAME: SAID OF HOUNDS...
SCHULZ
8-6
OH...

In this chapter, we present the semantic model we use to represent data, queries and services. This model allows describing concepts such as `Person`, relationships between concepts such as `WrittenBy`, and services with input and output types, possibly including nesting as in XML. The schema of extensional documents can also be expressed. The model is complex enough to capture services of interest found on the Web and simple enough so that we are able to cast information found on the Web to this model. We also mention (without providing a complete solution) the problem of indexing services and answering queries with a set of semantically described services.

We first describe the basic components of the semantic model: types and instances in Section 6.1, atoms and nested terms in Section 6.2, and services, documents and queries in Section 6.3. We next define the semantics, of types and instances in Section 6.4, and of services in Section 6.5. We finally discuss indexing and query answering in Section 6.6. We explain why the problem is different from what has already been done in the literature, and describe directions that we are currently exploring.

6.1 Types and Instances

We first give a formal definition of the *domain ontology* as introduced in Chapter 1.

Definition 6.1. A *domain ontology* is a 4-uple $(\mathscr{C}, \sqsubseteq, \mathscr{R}, \tau)$ where:
(i) $\mathscr{C}$ is a finite set of *concept names* and $\mathscr{R}$ is a finite set of *relation names*;
(ii) $\sqsubseteq$ is a (strict) partial order over $\mathscr{C}$ (*IsA* relationship);
(iii) $\tau : \mathscr{R} \to \bigcup_{n \in \mathbb{N}} \mathscr{C}^n$ is a *typing function*.

In the following, we consider fixed a domain ontology $(\mathscr{C}, \sqsubseteq, \mathscr{R}, \tau)$. An example of a very simple ontology has been given in Figure 1.3, page 10.

For $R \in \mathscr{R}$, $\tau(R) = (c_1 \dots c_n)$ is called the *type* of relation R and $n = |\tau(R)|$ the arity of R. Such a relation is denoted $R(c_1 \dots c_n)$. By abuse of notation, we assume that $\mathscr{C} \subseteq \mathscr{R}$ with $\forall c \in \mathscr{C}, \tau(c) = c$.

We also assume that, for each concept c, there exist countable sets $\mathscr{V}_c$, $\mathscr{D}_c$ of variables and constants, respectively. We assume that these sets, as well as $\mathscr{R}$, are pairwise disjoint. Let $\mathscr{V} = \bigcup \mathscr{V}_c$ and $\mathscr{D} = \bigcup \mathscr{D}_c$. For each x in $\mathscr{V}_c \cup \mathscr{D}_c$, let $\tau(x)$ be c. Observe that the typing function is now defined over $\mathscr{D} \cup \mathscr{V} \cup \mathscr{R}$. Note also that all constants are typed. We may extract the same string, e.g., `jaguar` in different contexts. It will yield two distinct constants, e.g., $\texttt{jaguar}_a$ (the animal) and $\texttt{jaguar}_b$ (the car brand).

The semantics of concepts and relationships (this will be extended to services further on) is provided by an *instance* **I**, i.e., a *finite* function from $\mathscr{C} \cup \mathscr{R}$ to $2^{\mathscr{D}}$ verifying:

(i) for $c \in \mathscr{C}$, $\mathbf{I}(c) \subseteq \bigcup_{c' \sqsubseteq c} \mathscr{D}_{c'}$;
(ii) for $c, c' \in \mathscr{C}$ such that $c \sqsubseteq c'$, $\mathbf{I}(c) \subseteq \mathbf{I}(c')$;
(iii) if $\tau(R) = (c_1 \dots c_n)$, $\mathbf{I}(R) \subseteq \mathbf{I}(c_1) \times \dots \times \mathbf{I}(c_n)$.

6.2 Atoms and Nested Terms

To define services, we use the auxiliary concepts of atoms and nested terms.

An *atom* is an expression $R(\mathrm{X})$ where R is a relation name and X is a vector of variables and constants, e.g., $R(\texttt{Ullman}, A)$. An atom $R(\mathrm{X})$ is *well-typed* if X has the same arity n as R and, for each $i \leqslant n$, the type c_i of the i-th component of X is compatible with that of the i-th component c_i' of $\tau(R)$, i.e., $c_i \sqsubseteq c_i'$.

Some of the documents as well as the output of services present data with some form of nesting. To capture this, we use the notion of *nested terms* over some countable set S (in practice, S will either be a set of variables or a set of concepts), that are defined using the abstract syntax:

$$T \leftarrow \langle S, \dots, S \rangle \mid \langle S, \dots, S, T^*, \dots, T^* \rangle.$$

More formally, they are defined as follows:

Definition 6.2. The set of *nested terms* over some countable set S, denoted $\mathscr{T}(S)$, is the set of finite expressions over the alphabet $S \cup \{\text{‘}\langle\text{’ ‘}\rangle\text{’ ‘,’ ‘*’}\}$ recursively defined as follows:

- If $s_1 \dots s_n \in S$, $n \geqslant 1$, then $\langle s_1, \dots, s_n \rangle \in \mathscr{T}(S)$.
- If $s_1 \dots s_n \in S$, $n \geqslant 1$ and $t_1 \dots t_m \in \mathscr{T}(S)$, $m \geqslant 1$, then $\langle s_1, \dots, s_n, t_1{}^*, \dots, t_m{}^* \rangle \in \mathscr{T}(S)$.

The restrictions of the kind of nesting allowed, in the spirit of nested relations [AB86], are important to define the semantics of valuations (see Definition 6.8).

For nested terms over a set of variables, we can extend the definition of types:

Definition 6.3. Let $t \in \mathscr{T}(\mathscr{V})$. The *type* of t, denoted $\tau(t)$, is a nested term over $\mathscr{C}$ recursively defined as follows: if $t = \langle v_1, \dots, v_n, t_1{}^*, \dots, t_m{}^* \rangle$ (with $n \geqslant 1$, $m \geqslant 0$, $v_1 \dots v_n \in \mathscr{V}$ and $t_1 \dots t_m \in \mathscr{T}(\mathscr{V})$),

$$\tau(t) = \langle \tau(v_1), \dots, \tau(v_n), \tau(t_1)^*, \dots, \tau(t_n)^* \rangle.$$

6.3 Services, Documents, and Queries

In this section, we define services, as well as documents, as Datalog-like rules. We also present how a query can be seen as a service without any input.

Definition 6.4. A *service definition* is a rule

$$W : T \leftarrow R_1(\mathrm{X_1}) \dots R_n(\mathrm{X_n}), \mathrm{Input}(\mathrm{Y})$$

where W is a *service identifier*, each $R_i(\mathrm{X_i})$ is a well-typed atom, **Y** (called the input) is a tuple of variables with no repetition, and $T \in \mathscr{T}(\mathscr{V})$ is a nested term (called the output), verifying:

(i) No variable appears twice in T.
(ii) Each variable occurring in T also occurs in some $\mathbf{X}_i$ or in $\mathbf{Y}$.

Example 6.5. Consider the definition of a service W providing titles of publications from some date D (the input), grouped by their authors:

$$W : \langle A, T^* \rangle \leftarrow \texttt{WrittenBy}(P, A), \texttt{PublishedIn}(P, D), \texttt{HasTitle}(P, T), \text{Input}(D).$$

Strictly speaking, we should also precise the type of each variable A, D, P, T. For simplicity of presentation, we assume that each atom is well-typed, types are as general as possible, and that there is no ambiguity. Here, for instance, we have $\tau(A) = \texttt{Author}$ and $\tau(T) = \texttt{Title}$. The type of the result is then $\langle \texttt{Author}, \texttt{Title}^* \rangle$.

The call $W(\mathit{2003})$ may be described as the input-less service:

$$\langle A, T^* \rangle \leftarrow \texttt{WrittenBy}(P, A), \texttt{PublishedIn}(P, \mathit{2003}), \texttt{HasTitle}(P, T). \qquad (6.1)$$

Note that this may also be viewed as the definition of a query: "Give me the titles of publications from 2003, grouped by their authors." An input-less service call may also be viewed as a *document schema*: an instance document of (6.1) would be a document with authors and article titles, grouped by authors, such that the corresponding paper was published in *2003*, e.g.:

$$\{(\mathit{Ed}, \{a, b\}), (\mathit{Sue}, \{b\})\}$$

We define more formally the semantics of services (and documents) in Section 6.5.

6.4 Semantics of Types and Valuations

In order to define the semantics of services and queries, we need to associate semantics to types and introduce the standard notion of valuation.

Definition 6.6. The *semantics* of types is a function $[\![\cdot]\!]$ that maps every concept and nested term of concepts as follows:

- for each $c \in \mathcal{C}$, $[\![c]\!] = \mathcal{D}_c$;
- for $t = \langle c_1 \ldots c_n, t_1{}^* \ldots t_m{}^* \rangle \in \mathcal{T}(\mathcal{C})$, $[\![t]\!]$ is the maximal subset of $2^{[\![c_1]\!] \times \cdots \times [\![t_m]\!]}$ that verifies the *key constraint* $c_1 \ldots c_n$.

Example 6.7. Let the semantics of `Person`, `Title` be respectively $\{\mathit{Ed}, \mathit{Sue}\}$ and $\{a, b\}$. Then the semantics of $\langle \texttt{Person}, \texttt{Title}^* \rangle$ is:

$$\{\varnothing, \{(\mathit{Ed}, \varnothing)\}, \ldots, \{(\mathit{Ed}, \{a, b\}), (\mathit{Sue}, \{a, b\})\}\}.$$

We use the classic notion of *valuation* of variables: A *valuation* v of a set of variables $\mathcal{V}'$ is a function that maps each x in $\mathcal{V}' \cap \mathcal{V}_c$ to some constant in $\bigcup_{c' \sqsubseteq c} \mathcal{D}_{c'}$. The set of valuations of the variables in $\mathcal{V}'$ is denoted $\mathsf{val}(\mathcal{V}')$.

Rules are typically used to produce tuples, the tuples of successful valuations. Our rules work similarly except that the successful valuations are "nested" as prescribed by the head of the rule. This nesting is captured by the following definition.

Definition 6.8. For each $\mathcal{V}' \subseteq \mathcal{V}$, $t = \langle v_1, \dots, v_n, t_1{}^*, \dots, t_m{}^* \rangle$ a nested term over $\mathcal{V}'$, and K a finite subset of $\mathsf{val}(\mathcal{V}')$, the t-nesting of K, denoted $\mathsf{nest}_t(K)$, is a finite element of $[\![\tau(t)]\!]$ defined recursively as follows:

$$\mathsf{nest}_t(K) = \Big\{ \langle \nu(v_1), \dots, \nu(v_n), \mathsf{nest}_{t_1}(J), \dots, \mathsf{nest}_{t_m}(J) \rangle \;\Big|\; \nu \in K \wedge J = \big\{ \nu' \in K \mid \forall 1 \leqslant i \leqslant n, \nu'(v_i) = \nu(v_i) \big\} \Big\}$$

Example 6.9. If ν_1 and ν_2 are defined by:

$$\nu_1 : \begin{cases} P \mapsto \mathsf{Sue} \\ T \mapsto \mathsf{a} \end{cases} \qquad \nu_2 : \begin{cases} P \mapsto \mathsf{Sue} \\ T \mapsto \mathsf{b} \end{cases},$$

we have:

$$\mathsf{nest}_{\langle P, T^* \rangle}(\{\nu_1, \nu_2\}) = \big\{ (\mathsf{Sue}, \{\mathsf{a}, \mathsf{b}\}) \big\}.$$

Similarly, we can introduce the notion of *unnesting* of a nested value:

Definition 6.10. Let $t = \langle v_1, \dots, v_n, t_1{}^*, \dots, t_m{}^* \rangle$ be a nested term over $\mathcal{V}$ and $K' \in [\![\tau(t)]\!]$ finite. As $K' \in 2^{[\![\tau(v_1)]\!] \times \dots \times [\![\tau(t_m)]\!]}$, we can denote:

$$K' = \Big\{ (c_{1,1} \dots c_{1,n}, k'_{1,1} \dots k'_{1,m}), \dots, (c_{p,1} \dots c_{p,n}, k'_{p,1} \dots k'_{p,m}) \Big\}.$$

We also use the following notation: for $c \in \mathcal{C}$ and $v \in \mathcal{V}$, let $\nu_c^{(v)}$ be the valuation of $\{v\}$ mapping v to c.

The *unnesting* of K', denoted $\mathsf{unnest}(K')$, is the set of valuations of the variables occurring in t recursively defined by:

$$\bigcup_{1 \leqslant j \leqslant p} \;\; \bigcup_{\substack{\nu'_1 \in \mathsf{unnest}(k'_{j,1}) \\ \dots \\ \nu'_n \in \mathsf{unnest}(k'_{j,m})}} \left\{ \nu_{c_{j,1}}^{(v_1)} \sqcup \dots \sqcup \nu_{c_{j,n}}^{(v_n)} \sqcup \nu'_1 \sqcup \dots \sqcup \nu'_n \right\}$$

where $\nu \sqcup \nu'$ is the function defined over the union of the domains of ν and ν' in a natural way (there is no ambiguity since t does not contain the same variable twice).

Example 6.11. If $K = \{(\mathsf{Sue}, \{\mathsf{a}, \mathsf{b}\})\} \in [\![\langle \mathtt{Person}, \mathtt{Title}^* \rangle]\!]$, the unnesting of K is the set $K' = \{\nu_1, \nu_2\}$ with the definitions of Example 6.9. In other words, $\mathsf{unnest}(\mathsf{nest}_t(K')) = K'$. This is an example of a more general result, stated further.

The following proposition states that our definitions of nesting and unnesting are consistent with each other.

Proposition 6.12. *Let t be a nested term over $\mathcal{V}$, and $Z_1 \dots Z_n$ the variables appearing in t. Let K be a finite subset of* $\mathsf{val}(\{Z_1 \dots Z_n\})$. *Then* $\mathsf{unnest}(\mathsf{nest}_t(K)) = K$.

Reciprocally, if t is a nested term over $\mathcal{V}$ and $K' \in [\![\tau(t)]\!]$ finite such that K' contains no empty set (however deep), then $\mathsf{nest}_t(\mathsf{unnest}(K')) = K'$.

Proof. We use a structural induction on t. Suppose that $t = \langle v_1, \dots, v_n, t_1{}^*, \dots, t_m{}^* \rangle$ (with m possibly equal to 0 to include the base case of the induction).

- Let K be a finite set of valuations of the variables occurring in t. Suppose that, for all $1 \leqslant i \leqslant m$, for each set K_i of valuations of the variables occurring in t_i, $\mathsf{unnest}(\mathsf{nest}_{t_i})(K_i) = K_i$, and let us show that $\mathsf{unnest}(\mathsf{nest}_t(K)) = K$. Let $K_{|\{v_1,\dots,v_n\}}$ be the set of valuations of $\{v_1,\dots,v_n\}$ that are restrictions of elements of K. We have:

$$\begin{aligned}\mathsf{unnest}(\mathsf{nest}_t(K)) &= \bigcup_{\nu\in K_{|\{v_1,\dots,v_n\}}} \bigcup_{\substack{\nu'_1\in K, \forall 1\leqslant p\leqslant n, \nu'_1(v_p)=\nu(v_p)\\ \dots \\ \nu'_m\in K, \forall 1\leqslant p\leqslant n, \nu'_m(v_p)=\nu(v_p)}} \left\{\nu_{|\{v_1\}} \sqcup \dots \sqcup \nu_{|\{v_n\}} \sqcup \nu'_1 \sqcup \dots \sqcup \nu'_m\right\}\\ &= K\end{aligned}$$

  with some abuse of notation since each ν'_i should only be defined on variables from t_i.

- Let $K' \in [\![\tau(t)]\!]$ finite. Suppose that, for all $1 \leqslant i \leqslant m$, for each $K'_i \in [\![\tau(t_i)]\!]$ finite, $\mathsf{nest}_t(\mathsf{unnest}(K'_i)) = K'_i$, and let us show that $\mathsf{nest}_t(\mathsf{unnest}(K')) = K'$. We use the same notations for the elements of K' as in Definition 6.10. Then

$$\mathsf{unnest}(K') = \bigcup_{1\leqslant j\leqslant p} \bigcup_{\substack{\nu'_1\in\mathsf{unnest}(k'_{j,1})\\ \dots \\ \nu'_n\in\mathsf{unnest}(k'_{j,m})}} \left\{\nu^{(v_1)}_{c_{j,1}} \sqcup \dots \sqcup \nu^{(v_n)}_{c_{j,n}} \sqcup \nu'_1 \sqcup \dots \sqcup \nu'_n\right\}$$

  and $\mathsf{nest}_t(\mathsf{unnest}(K'))$ is equal to:

$$\bigcup_{1\leqslant j\leqslant p} \left\{\langle c_{j,1},\dots,c_{j,n}, \mathsf{nest}_{t_1}(\mathsf{unnest}_{t_1}(k'_{j,1})),\dots,\mathsf{nest}_{t_m}(\mathsf{unnest}_{t_m}(k'_{j,m}))\rangle\right\} = K'. \qquad \square$$

We are now ready to define the semantics of queries and services.

6.5 Semantics of Services

An instance $\mathbf{I}$ also provides the semantics of services. More precisely:

Definition 6.13. Given a service definition

$$W : T \leftarrow R_1(\mathbf{X_1})\dots R_n(\mathbf{X_n}), \mathrm{Input}(\mathbf{Y}),$$

and an instance $\mathbf{I}$, the semantics for W is a function w from $[\![\tau(\mathbf{Y})]\!]$ to $[\![\tau(T)]\!]$ such that, for all $\mathbf{y} \in [\![\tau(\mathbf{Y})]\!]$:

$$\mathsf{unnest}(w(\mathbf{y})) \subseteq \bigcup \left\{ \nu_{|\{Z_1\dots Z_m\}} \mid \nu \in \mathsf{val}(\mathcal{V}') \wedge \nu(\mathbf{Y}) = \mathbf{y} \wedge \forall 1 \leqslant i \leqslant n, \nu(\mathbf{X_i}) \in \mathbf{I}(R_i) \right\} \tag{6.2}$$

where $\mathcal{V}'$ is the set of variables occurring in the service definition and $Z_1 \dots Z_m$ the variables occurring in T. We often denote this function $\mathbf{I}(W)$, extending in this way instances to service definition.

If W has no input (i.e., if it is a document definition), its semantic is then defined as a constant element of $[\![\tau(T)]\!]$, rather than as a function.

It is essential to observe the inclusion in (6.2): the semantics of a service only implies its *soundness* relatively to the instance, not its *completeness*.

Let **I** be an instance and Q a query (a service with no input). Informally, the *answer* to Q is the nesting of some set of valuations for which the body holds. Observe that the services are not used to obtain that answer. They may be viewed as constraints we know on the concepts and the relationships between them.

Let us now consider the case of the hidden Web. We do not have access to the set of concepts and relationships, although this is what we want to query. We see the world only through views. Some views are parameter-less (the documents) and some have parameters (services with input relation). Now we can think of services as factories to produce more information as illustrated in the following example.

Example 6.14. Suppose that we have (1) a no-input service (a document) that returns a list of authors; (2) a service in that returns the title of publications from a certain author; and (3) a service that, given an article title, returns its authors. We can use (1) to obtain an author, say a_1. Then (2) to obtain a title t_1. Then (3) to obtain an author a_2. Then (2) to obtain t_2, and so on. We obtain a chain of unbounded length of service calls.

A difficulty in the inference mechanism is that the inputs of services typically speak of concrete concepts (`Author`, `Title`) and not of abstract concepts such as `Publication`. So, using the views, we derive information with "skolems," typically playing the role of the existential variables in rule bodies. To see an example, consider the service definition:

$$W : \langle X, Z \rangle \leftarrow H(X, Y, Z).$$

Suppose we call that service and obtain ($\mathtt{Ed}$,$\mathtt{5}$). Then we know of the relationship $H(\mathtt{Ed}, f(\mathtt{Ed},\mathtt{5}), \mathtt{5})$ where $f(\mathtt{Ed},\mathtt{5})$ is a Skolem variable. Of course, we cannot call services with such skolems. They can be of use, however, for answering some queries. Consider for instance the query

$$Z \leftarrow H(X, Y, Z).$$

We can conclude that `5` is an answer to this query.

For each concept c, let us distinguish an infinite subset $Skol_c$ of $\mathcal{D}_c$ of otherwise unused constant names; the elements of $Skol_c$ will serve as Skolem variables. Let $Skol = \bigcup_{c \in \mathcal{C}} Skol_c$.

Definition 6.15. Let **I** be an instance and

$$W : T \leftarrow R_1(\mathrm{X_1}) \ldots R_n(\mathrm{X_n}), \mathrm{Input}(\mathrm{Y})$$

a service definition with semantics $\mathbf{I}(W)$. Let $Z_1 \ldots Z_m$ be the variables appearing in T.

Let **J** be another (partial) instance. The *immediate consequence* of **J** by $\mathbf{I}(W)$, denoted here $\chi_{\mathbf{I}(W)}(\mathbf{J})$ is the instance $\mathbf{J}'$ defined by:

- $\forall c \in \mathcal{C}$,

$$\mathbf{J}'(c) = \mathbf{J}(c) \cup \bigcup_{\substack{1 \leqslant k \leqslant m \\ \tau(Z_k) \sqsubseteq c}} \; \bigcup_{\substack{\mathbf{y} \in \mathbf{J}(\tau(\mathbf{Y})) \\ \forall i, y_i \notin Skol}} \; \bigcup_{\nu \in \mathrm{unnest}(\mathbf{I}(W)(\mathbf{y}))} \{\nu(Z_k)\}$$

- $\forall R \in \mathcal{R}$,

$$\mathbf{J}'(R) = \mathbf{J}(R) \cup \bigcup_{\substack{1 \leqslant k \leqslant n \\ R_k = R}} \; \bigcup_{\substack{\mathbf{y} \in \mathbf{J}(\tau(\mathbf{Y})) \\ \forall i, y_i \notin Skol}} \; \bigcup_{\nu \in \mathsf{unnest}(\mathbf{I}(W)(\mathbf{y}))} \{\bar{\nu}(\mathbf{X_k})\}$$

  where $\bar{\nu}$ is the valuation defined over all variables appearing in W in the following way:

  - if X is one of the Z_i, $\bar{\nu}(X) = \nu(Z_i)$;
  - otherwise, $\bar{\nu}(X)$ is a *fresh* Skolem variable (i.e., a fresh element of $Skol_{\tau(X)}$).

As long as the original partial instance is sound, the immediate consequence operator produces sound results:

Lemma 6.16. *Let* $\mathbf{I}$ *be an instance,* W *a service definition with semantics* $\mathbf{I}(W)$, *and* $\mathbf{J}$ *a partial instance. We assume that there exists a mapping* φ *from* $\mathcal{D}$ *to itself that leaves fixed all elements of* $\mathcal{D}$ *which are not skolems, and such that* $\varphi(\mathcal{D}) \cap Skol = \varnothing$ *and* $\varphi(\mathbf{J}) \subseteq \mathbf{I}$ *(in other words,* $\mathbf{J}$ *is a* sound *instance). Then there exists a mapping* φ' *from* $\mathcal{D}$ *to itself that leaves fixed all elements of* $\mathcal{D}$ *which are not skolems, and such that* $\varphi'(\mathcal{D}) \cap Skol = \varnothing$ *and* $\varphi'(\chi_{\mathbf{I}(W)}(\mathbf{J})) \subseteq \mathbf{I}$.

Proof. Consider the set S of Skolem variables that appear in $\chi_{\mathbf{I}(W)}(\mathbf{J})$ in addition to these that appear in $\mathbf{J}$. By definition of the immediate consequence operator, each $s \in S$ appears in exactly one tuple t_s of $\chi_{\mathbf{I}(W)}(\mathbf{J})$, and there exist $1 \leqslant k_s \leqslant n$, a vector $\mathbf{y_s} \in \mathbf{J}(\tau(\mathbf{Y}))$ of constants that are not skolems, and $\nu_s \in \mathsf{unnest}(\mathbf{I}(W)(\mathbf{y}))$ such that $t_s = \bar{\nu}_s(\mathbf{X}_{k_s})$. Besides, s occurs multiple times in t_s if and only if the corresponding variable is repeated. Let X_s be this variable.

Since $\varphi(\mathbf{J}) \subseteq \mathbf{I}$, $\varphi(\mathbf{y_s}) = \mathbf{y_s} \in \mathbf{I}(\tau(\mathbf{Y}))$. And, because of (6.2), $\nu_s(\mathbf{Y}) = \mathbf{y_s}$ and ν_s can be extended into a valuation ν'_s of all variables occurring in the service definition, such that, for all i, $\nu'_s(\mathbf{X}_i) \in \mathbf{I}(R_i)$. We then fix the following mapping φ' from $\mathcal{D}$ to itself:

$$\begin{cases} \varphi'(s) := \nu'_s(X_s) & \text{if } s \in S; \\ \varphi'(s) := \varphi(s) & \text{otherwise.} \end{cases}$$

Then $\varphi'(\chi_{\mathbf{I}(W)}(\mathbf{J}) \subseteq \mathbf{I}$. □

Let $W_1 \dots W_r$ be service definitions with semantics $\mathbf{I}(W_1) \dots \mathbf{I}(W_r)$. Let $(\mathbf{J}_j)_{j \in \mathbb{N}}$ be the sequence of instances recursively defined by:

$$\begin{cases} \mathbf{J}_0 = \varnothing \\ \forall j \in \mathbb{N}, \mathbf{J}_{j+1} = \chi_{\mathbf{I}(W_1)} \circ \dots \circ \chi_{\mathbf{I}(W_r)}(\mathbf{J}_j) \end{cases} \tag{6.3}$$

The *consequence* of $\mathbf{I}(W_1) \dots \mathbf{I}(W_r)$, denoted $\mathsf{Csq}_{\mathbf{I}(W_1) \dots \mathbf{I}(W_r)}$, is an instance defined as:

$$\mathsf{Csq}_{\mathbf{I}(W_1) \dots \mathbf{I}(W_r)} = \bigcup_{j \in \mathbb{N}} \mathbf{J}_j.$$

Observe that, since $\mathbf{I}$ is finite, only a finite number of service calls can be performed; therefore $\mathsf{Csq}_{\mathbf{I}(W_1) \dots \mathbf{I}(W_r)}$ is finite and can be computed in a finite number of iterations. Still, this number is unbounded (consider, for instance, a genealogy service taking as input a person, and returning as output the parents of this person; using this service to find all the ancestors of a given person needs an unbounded number of service calls).

We can now define the semantics of the answer to a query.

Definition 6.17. Let $Q : T \leftarrow R_1(\mathrm{X_1}) \dots R_n(\mathrm{X_q})$ be a query, $\mathbf{I}$ an instance and $W_1 \dots W_r$ r service definitions, with semantics $\mathbf{I}(W_1) \dots \mathbf{I}(W_r)$. Let $Z_1 \dots Z_p$ be the variables appearing in T. Two different semantics for the answer to a query Q are defined:

- Given an instance $\mathbf{I}$, the *answer to Q*, denoted $\mathbf{I}(Q)$, is defined as:

$$\mathsf{nest}_T \left(\left\{ \nu_{|\{Z_1 \dots Z_p\}} \,\middle|\, \nu \in \mathsf{val}(\mathcal{V}') \wedge \forall 1 \leqslant i \leqslant q, \nu(\mathrm{X_i}) \in \mathbf{I}(R_i) \right. \right.$$
$$\left. \left. \wedge\ \forall 1 \leqslant j \leqslant p, \nu(Z_j) \notin \mathit{Skol} \right\} \right)$$

  where $\mathcal{V}'$ is the set of variables occurring in Q.

- The *consequence answer to Q*, is $\mathsf{Csq}_{\mathbf{I}(W_r)\dots\mathbf{I}(W_r)}(Q)$.

$\mathbf{I}(Q)$, $\mathsf{Csq}_{\mathbf{I}(W_r)\dots\mathbf{I}(W_r)}(Q)$ are subsets of $[\![\tau(T)]\!]$.

Theorem 6.18. *Let* $\mathbf{I}$ *be an instance, Q a query and $W_1 \dots W_r$ service definitions with semantics* $\mathbf{I}(W_1) \dots \mathbf{I}(W_r)$. *Then* $\mathsf{unnest}(\mathsf{Csq}_{\mathbf{I}(W_1)\dots\mathbf{I}(W_r)}(Q)) \subseteq \mathsf{unnest}(\mathbf{I}(Q))$.

Proof. We use the result from Proposition 6.12, which states that $\mathsf{unnest} \circ \mathsf{nest}_t$ is the identity function.

With the notations of Definition 6.17, $\mathsf{unnest}(\mathsf{Csq}_{\mathbf{I}(W_1)\dots\mathbf{I}(W_r)}(Q))$ is the following:

$$\left\{ \nu_{|\{Z_1 \dots Z_p\}} \,\middle|\, \nu \in \mathsf{val}(\mathcal{V}') \wedge \forall 1 \leqslant i \leqslant q, \nu(\mathrm{X_i}) \in \mathsf{Csq}_{\mathbf{I}(W_1)\dots\mathbf{I}(W_r)}(R_i) \right.$$
$$\left. \wedge\ \forall 1 \leqslant j \leqslant p, \nu(Z_j) \notin \mathit{Skol} \right\}.$$

Let ν be such a valuation. With the notations of (6.3),

$$\mathsf{Csq}_{\mathbf{I}(W_1)\dots\mathbf{I}(W_r)} = \bigcup_{j \in \mathbb{N}} \mathbf{J}_j.$$

So there necessarily exists a $k \in \mathbb{N}$ such that, for all $1 \leqslant i \leqslant q$, $\nu(\mathrm{X_i}) \in \left(\bigcup_{0 \leqslant j \leqslant k} \mathbf{J}_j \right)(R_i)$. But it directly follows from Definition 6.15 that for all j, $\mathbf{J}_j \subseteq \mathbf{J}_{j+1}$. We have thus:

$$\forall 1 \leqslant i \leqslant q, \nu(\mathrm{X_i}) \in \mathbf{J}_k(R_i).$$

Observe now that $\mathbf{J}_0 = \varnothing$ is a sound instance, and $\mathbf{J}_k$ is obtained from $\mathbf{J}_0$ by a finite sequence of applications of the immediate consequence operator. It then follows from Lemma 6.16 that there exists a φ from $\mathcal{D}$ to itself that leaves all skolems fixed and with range disjoint from the set of skolems, such that $\varphi(\mathbf{J}_k) \subseteq \mathbf{I}$. Since $\nu(\mathrm{X_i}) \in \mathbf{J}_k(R_i)$ for each i, $\varphi \circ \nu(\mathrm{X_i}) \in \mathbf{I}(R_i)$ for each i. Besides, $\varphi \circ \nu(Z_i) \notin \mathit{Skol}$ for each j. This means that $\nu \circ \varphi_{|\{Z_1 \dots Z_p\}} = \nu_{|\{Z_1 \dots Z_p\}} \in \mathsf{unnest}(\mathbf{I}(Q))$, which concludes the proof. □

Note that we do not have, however, $\mathsf{Csq}_{\mathbf{I}(W_1)\dots\mathbf{I}(W_r)} \subseteq \mathbf{I}$ since $\mathsf{Csq}_{\mathbf{I}(W_1)\dots\mathbf{I}(W_r)}$ also contains data about Skolem variables.

Example 6.19. Let Q be the query $A^* \leftarrow$, with $\tau(A) = \texttt{Author}$.
Let W_1 be the service definition:

$$A \leftarrow \texttt{WrittenBy}(P,A), \texttt{HasTitle}(P,T), \text{Input}(T).$$

Since an input must be provided to the service to learn new facts, $\texttt{Csq}_{\mathbf{I}(W_1)}$ is the function mapping each concept and relation to the empty set (and $\texttt{Csq}_{\mathbf{I}(W)}(Q) = \varnothing$). However, If $\mathbf{I}$ is such that $\mathbf{I}(\texttt{Person}) = \{\textit{Sue}, \textit{Paul}\}$, $\mathbf{I}(Q) = \{\textit{John}, \textit{Paul}\}$.
Suppose now that we have another service definition W_2 with no input: $T \leftarrow$, with $\tau(T) = \texttt{Title}$. If $\mathbf{I}(W_2) = \{a, b\}$ and $\mathbf{I}(W_1)$ is defined by $a \mapsto \textit{Sue}$, $\texttt{Csq}_{\mathbf{I}(W_1),\mathbf{I}(W_2)}(Q) = \{\textit{Sue}\}$.

As can be seen in the previous example, the inclusion of Theorem 6.18 is not an equality. The semantics of a query describe the "ideal" semantics which could be given to an answer by an observer having access to the actual database instance, whereas this instance is only seen through views on the data. An important remark is that documents are *essential* for having a non-empty consequence answer to a query: they provide what is needed to *bootstrap* the use of services.

6.6 Indexing and Query Processing

Given a set of service descriptions (that may result from the semantic analysis of services of the hidden Web), we want to transform a user query into a set of service calls to answer the query. It is unpractical to directly compute the set of all consequences of services, because this would require an unbounded number of calls to services. Moreover, this would provide an *extensional* knowledge about the semantics of the services, i.e., a materialized view. We are more interested in an *intensional* way to store and query knowledge about services, because it is flexible, and because it is much more adapted to the case of Web databases (in particular, for bandwidth and server load reasons).

The first issue is to have a way to index service definitions, document definitions, as well as instance documents, in order to be able to compute efficiently query execution plans (see further). A long-term plan is to integrate this effort with KadoP [AMP05], a system for indexing documents, documents incorporating service calls, and services themselves over a peer-to-peer network. This will require to index, for a given service, both information about its input and output parameters and their types, and information about the relations between these parameters.

The second issue is in the evaluation of queries itself. This can be decomposed into two parts: a static execution plan computation, deriving a sequence of service calls to be carried out from the query, and a possible dynamic choice of services to be called to extend the static execution plan in the view of partial results computed so far. The question of the execution plan is similar to the problem of "Answering Queries Using Views" which has been the topic of many works in the literature. The problem was introduced in [LMSS95], and a survey of the different approaches can be found in [Hal01]. Let us highlight the specificities of our context before discussing related work.

Access-Pattern Limitations. The main and essential feature of our model for describing services of the hidden Web is that we do not have direct access to the data of each source. There are limitations on the patterns of access to the source, which can be seen as a *binding patterns*. Some input must be provided to get an output. In other words, some variables have to be *bound* in the call.

Composition and Recursion. It may be necessary to call several services to obtain results. In some contexts, it may also be needed to consider *recursive* query evaluation plans, that make successive calls to a single service. An example of such a case has been given in Example 6.14.

Incompleteness. Each service of the hidden Web is likely to have only a partial view of the world, that is, is essentially *incomplete*. This means that we cannot rely on techniques that find a single equivalent rewriting of the query but have to consider *maximally contained* rewritings. In the terminology of [Hal01], we are considering the problem of answering queries using views for *data integration*.

Subsumption. We have an IsA ontology of concepts, and an instance of a specific context is also an instance of a more general concept. These subsumption relations have to be taken into account in the query evaluation. One way to do this is to model them by adding pseudo-services $A \leftarrow B$ for every concept $b \subseteq a$.

Nesting. We consider nested types for the output of services (and the types of documents). This, however, is unlikely to be a major issue since it is always possible to nest and unnest data when needed. It is necessary on the Web, where nesting of data is frequent.

Most works on answering queries using views with binding patterns (e.g., [RSU95]), assume that each source is *complete*, and provide a single equivalent rewriting of the query, which is not adapted to our context, as argued above.

Well-known algorithms for answering queries using views are Bucket [LRO96] and MiniCon [PL00, PH01]. Bucket has been developed in the context of the *Information Manifold* system [KLSS95], a pioneer system for data integration on the Web. Subsumption issues are considered in [LRO96]. MiniCon is an improvement over Bucket and is quite similar, while performing better in practice. The first step of both algorithms is to identify views that are relevant to the query. This is not adapted to our need of both managing access-pattern limitations and allowing sequential chaining of services. Indeed, this step does not consider sources whose only role is to provide another service values for one of its bound parameters, even though this might be the only way to use this second service. Furthermore, recursive query plans are not considered.

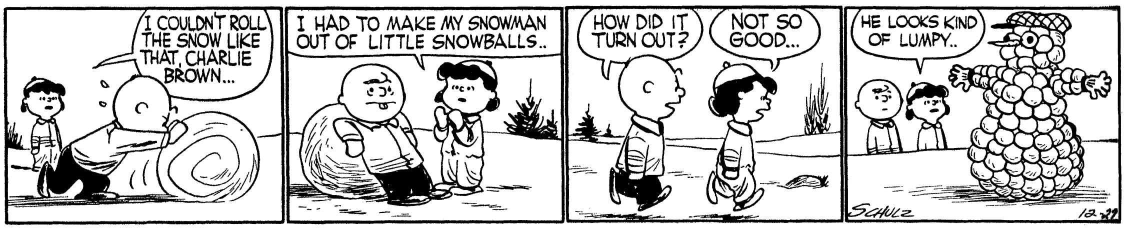
Another approach to answering queries using views for data integration purposes is the inverse-rules algorithm [DG97, DGL00]. It basically consists in rewriting a LAV (Local As View) system into a GAV (Global As View) system, by inverting each rule, introducing skolems in the process. Recursive query plans are considered. Besides, access-pattern limitations are discussed in [DGL00], but in a way that would be very inefficient in our context: it basically requires to compute the set of all consequences of a service.

Our current direction for solving this problem (work in progress) is to use techniques in the spirit Magic sets [AHV95, BR87, BR91]. Magic is a method for efficient evaluation of Datalog programs. Its name comes from the fact that, although it is a bottom-up approach, which propagates base facts with rules, it is “magically” as efficient as top-down approaches, that explore a derivation tree starting from a fact to prove. One of the main steps of Magic is to annotate each Datalog rule with all possible binding patterns, yielding thus several different rules. We are currently considering using Magic and just filtering out rewritten rules that present inappropriate binding patterns.

Conclusion

We presented a semantic model for representing services of the hidden Web, with typed inputs and outputs, semantic relations between them, and nesting. We also considered the semantics of queries over such a model. The actual indexing and query answering is still work in progress. Another issue that has not been discussed and remains for future work is the way to use information about the confidence in a service, and its relevance to a particular query, to order query evaluation plans, and, ultimately, to rank query results.

Conclusion


I COULDN'T ROLL THE SNOW LIKE THAT, CHARLIE BROWN...
I HAD TO MAKE MY SNOWMAN OUT OF LITTLE SNOWBALLS..
HOW DID IT TURN OUT?
NOT SO GOOD...
HE LOOKS KIND OF LUMPY..
SCHULZ

Putting it All Together

Let us first see how the various components discussed in this thesis are put together in practice, to obtain a complete system for understanding the Hidden Web. The general architecture has been discussed in Chapter 1 and, in particular, Figures 1.1 and 1.2 show how the components interact with one another. In practice, focusing on the parts of this architecture that we have actually developed, we have the following process:

1. Build the domain knowledge as described in Section 1.3.
2. Gather forms that are entry points of services of the hidden Web in some way (e.g., manually) and enter them into the probabilistic warehouse.
3. Wrap each of these forms as a Web service, as described in Sections 3.6 and 4.4. Insert the description of the resulting service, with adequate confidence values, into the probabilistic warehouse.
4. Run an *ad hoc* program that queries and updates the probabilistic warehouse in order to add semantic relationships between inputs and outputs (for instance, if the input of a service is a `Title` and the output a set of `Authors`, this program states that the latter are authors of a paper which has the former as a title, with high probability).
5. Present to the user a high-level query interface over the domain ontology.
6. Translate the user's query to a query over the set of known services, as addressed in Section 6.6, evaluate this query, and return the (probabilistic) results to the user.

We are currently working on implementing this entire process.

Perspectives

To conclude this thesis, we would like to highlight important problems in the context of understanding the hidden Web. We classify these problems into: (i) the ones that we are already addressing, (ii) those that, we believe, require effort but should be tractable without any intrinsic difficulties, and (iii) those that involve intense research.

Let us start with work in progress.

Practical application of Chapter 5. The theoretical framework of Chapter 5 does not have a practical application yet. We would like to explore two different ways to bridge this gap: (i) direct implementation of the framework, despite of the negative complexity results, with appropriate heuristics for efficiently approximating the cost of a schema mapping; (ii) relations with the field of inductive logic programming.

Answering queries using views. We are currently working on the problem of answering queries using views, views being here the semantically described services of the hidden Web. We mentioned in Section 6.6 a possible direction of research based on the Magic techniques for efficient evaluation of Datalog programs.

System integration and demo. We have discussed at the beginning of this conclusion how the different modules can be put together to form a complete system for the integration of services of the hidden Web. We are currently working on this integration, with the purpose of having a demo of the entire system. We also plan to apply the system to another domain (possibly weather services or person directories) in order to validate its generality.

The following problems seem reasonably solvable, possibly with some effort:

Service acquisition. This was discussed in Section 1.4.1. We believe that a combination of heuristics and focused crawling techniques discussed there may be enough to acquire a satisfactory number of relevant services. This requires implementing and combining all of this, which is not completely direct.

More general kinds of forms. We made a number of assumptions on the structure of the forms we dealt with in Chapter 3. Although dealing with an arbitrary form with no recognized structure is a very difficult task, one could remove a number of these assumptions to support, for instance, forms with some required inputs, or forms that use boolean operators. This would be useful because informal experiments show that there is not that much heterogeneity in the structure of forms on the Web. This would involve recognizing a general *template* a form follows and analyzing the different fields with respect to this template.

Better gazetteer. It is always possible to improve the annotation that is performed by the gazetteer that we described in Chapter 4. In particular, existing entity recognizers (cf. for instance [8], that could also be used for dealing with services in other languages than English), that are both general and quite effective, for dates, person names, addresses, and so on, may perform better than the ones we used. Besides, linguistic information could also be used, for instance, to annotate noun phrases of appropriate length as possible titles.

Finally, let us present some open issues that we believe to be important to the problem of understanding the hidden Web, and for which we do not know of existing solutions.

Updates of confidence values. Note that in the current prob-tree model described in Chapter 2, the probability that a given node is in the tree can decrease (if this node is conditionally deleted) but never increase (though a node with the same label and descendants can be inserted to simulate this). This reflects the fact that there is no much sense in general in arbitrary modification *a posteriori* of probabilities; the increase (or decrease) of the confidence in a fact is still something that is sometimes needed in an integration system. This operation should be formalized in a clean way and the issue of how performing it on a prob-tree should be considered. See also further (under Deduplication) for a related issue that might provide updated confidence values.

Top-*k* probabilistic results. One of the most obvious limitations of the prob-tree model as described in Chapter 2 is that the result to a query is given (and computed) in its entirety. When this result is large, or when we are just interested in the most probable results, it would be interesting to get top-*k* probabilistic results in an efficient way. We do not have a solution to this, since classical top-*k* techniques [FLN03] are not directly applicable.

A more adapted machine learning framework. Experimental results presented in Chapter 4 show that it is possible to use machine learning techniques to learn the structure underlying an imprecise and imperfect annotation. Some of the results are good, while some are somewhat disappointing. We believe that the main reason why this is so is the fact that conditional random fields (as, to our knowledge, all other supervised machine learning techniques) are designed to work in a context where the original annotation is supposed to be perfect. Therefore, there are risks of overfitting. A more adapted machine learning model would also try to minimize the description length of the obtained wrapper, in a manner similar to what we described in Chapter 5. How this should be done is far from obvious.

Tuple extraction. The method that we described in Chapter 4 for extracting tuples from individual annotated data values is quite *ad hoc* and not very robust. This cannot take into account pages where some of the components of a tuple are factored out. We have tried to use a machine learning technique for simultaneous extraction of tuples [GMTT06], but with less success than conditional random fields.

Deduplication. Different services may have slightly different information about the same entity (say, slightly different author or conference names for the same paper). In order not to present to a user a list of quasi-identical results, it is necessary to perform a *deduplication* step that identify and merge duplicates. This deduplication is also needed in other contexts, for instance when we identify constants as in 5. We do not have any solution for this. A related problem of interest is that of *data corroboration*: when multiple sources state different facts (say, when different databases have various information about the same entity), what is the global probability that the fact is true (and that the source is trustworthy)? A first approach to this problem is presented in [YHY07].

Semantic analysis. In order to identify the relations between inputs and outputs of a service, in addition to the methods that we started exploring in Chapter 5, that are based solely on the constants that appear, techniques that use the context of a service (especially its natural language context) could be used to derive such relations, for instance to understand that the output of a genealogical service is the father of the corresponding input. This is a hard problem, that should probably be attacked with a combination of natural language processing techniques and machine learning.

This list of open or partially solved problems has no pretention to be exhaustive.

Appendix A

Résumé en français

This appendix is a translation, in French, of the content of the introduction, Chapter 1 and conclusion; it does not contain any additional content, and may safely be skipped. An English-to-French lexicon of technical terms is also provided at the end of this chapter.

Introduction

Accéder à des informations du Web implique principalement, de nos jours, l'utilisation de moteurs de recherche par mots-clefs. Ces moteurs fournissent des résultats du *Web de surface*, l'ensemble des pages Web directement accessibles par des hyperliens, et ne considèrent quasiment pas les informations, en très grande quantité, qui se trouvent cachées derrière des formulaires et qui composent le *Web caché* (également appelé *Web profond* ou *Web invisible*). Le sujet de cette thèse est l'exploitation automatisée des ressources du Web caché et, plus précisément, la découverte, l'analyse, la compréhension et l'interrogation de telles ressources. Il est évident que ceci va bien au-delà de la portée d'une simple thèse de doctorat ; nous n'y apporterons pas de réponse complète. Nous présentons un cadre général et proposons des solutions à certains problèmes particuliers soulevés par l'exploitation du Web caché.

Dans le chapitre 1, nous introduisons un cadre général pour la compréhension du Web caché. L'accent est mis sur des systèmes *entièrement automatiques*, qui ne nécessitent pas d'intervention humaine. Étant donné que le problème de la compréhension des ressources du Web caché est indubitablement IA-complet, nous limitons notre intérêt à un *domaine d'application* spécifique, en nous basant sur une *base de connaissances du domaine*. Notre approche est *centrée sur le contenu*, c'est-à-dire que le cœur de notre système consiste en un entrepôt de contenu sur les services du Web caché, avec des modules indépendants qui enrichissent notre connaissance de ces services, afin de mieux les exploiter. Par exemple, un module peut être responsable de la découverte de nouveaux services pertinents (comme des URL de formulaires), un autre de l'analyse de la structure des formulaires, et ainsi de suite. Ces différents « agents » sont ensuite combinés[*] d'une manière qui est décrite dans le chapitre 1. Des aspects spécifiques de ces agents sont décrits dans les chapitres suivants. Une fonctionnalité importante, la découverte de services pertinents, est seulement brièvement évoquée en partie A.4.1. De tels services peuvent être obtenus, par exemple, en interrogeant des moteurs de recherche ou en utilisant une exploration guidée du Web.

La plupart des idées présentées dans cette thèse ont été implémentées dans des prototypes. Pour valider notre approche et tester ces systèmes, nous avons utilisé le domaine des publications scientifiques. Nous utilisons ici la même application pour illustrer notre travail. Nous insistons sur le fait que les idées et logiciels peuvent être utilisés dans n'importe quelle domaine d'application, à partir

[*] Bien que nous présentions ici un enchaînement séquentiel, la qualité de l'analyse des services pourra être améliorée en utilisant des combinaisons plus complexes. Cela ne sera pas considéré ici à fin de simplification.

du moment où une base de connaissances du domaine est disponible dans un format bien précis que nous présenterons.

Les données que nous traitons sont généralement assez irrégulières et souvent arborescentes. Il est ainsi naturel d'utiliser un modèle de données semi-structuré ; nous utilisons XML puisque c'est une norme du Web. Les différents agents qui coopèrent pour construire un entrepôt de contenu sont intrinsèquement imprécis (étant donné qu'ils utilisent en général des techniques d'apprentissage ou des heuristiques, qui sont sujets à imprécision). Nous utilisons ainsi un entrepôt de contenu XML *probabiliste*, qui est interrogé et mis à jour (les mises à jour incluant des probabilités) par les différents agents. Le modèle XML probabiliste est décrit dans le chapitre 2. Nous introduisons les arbres probabilistes comme des arbres classiques, annotés avec des conjonctions de variables aléatoires indépendantes (et leur négation). Nous montrons comment évaluer de manière efficace des requêtes et des mises à jour dans ce modèle, et en décrivons une implémentation. Il est à noter que, bien que le développement de ce modèle d'arbres probabilistes ait été motivé par notre contexte du Web caché, c'est un modèle très général qui peut être appliqué à des contextes différents.

Considérons un service du Web caché, par exemple un formulaire HTML, qui est pertinent pour le domaine d'application spécifié. Pour comprendre sa sémantique, une première étape est d'analyser ce formulaire, c'est-à-dire la structure des entrées du service. Ceci est traité dans le chapitre 3. Nous utilisons des heuristiques pour associer des concepts du domaine aux champs du formulaire, et *sondons* ensuite ces champs avec des instances du domaine pour confirmer ou infirmer ces hypothèses. Le cœur de la validation consiste en des techniques pour distinguer les *pages de résultats* des *pages d'erreur*. Nous présentons des résultats expérimentaux relativement satisfaisants, sur des formulaires de recherche dans des bases de données de publications scientifiques.

L'étape suivante est l'extraction d'information à partir des résultats d'un formulaire, qui sont des pages HTML. Nous décrivons dans le chapitre 4 comment des techniques d'extraction d'information supervisées, comme les champs aléatoires conditionnels, peuvent être utilisées de manière non supervisée, à l'aide des connaissances du domaine. Un programme d'annotation étiquette les pages de résultats avec des concepts du domaine (grâce aux instances du domaine qui sont reconnues). Cette annotation, à la fois imprécise et incomplète, est utilisée comme amorce du processus d'apprentissage.

Une compréhension de la structure des entrées et sorties d'un formulaire résulte de ces deux étapes (avec une certaine imprécision, bien sûr). Il est alors facile de transformer ce formulaire en un service Web standard décrit en WSDL.

Il est ensuite nécessaire de comprendre les relations sémantiques qui existent entre les entrées et sorties d'un service. Ce problème est abordé dans le chapitre 5. Pour simplifier, le cadre est relationnel. Nous introduisons un modèle théorique pour la découverte de relations entre deux instances de bases de données ayant des schémas distincts et inconnus. Ce modèle prend sa source dans le contexte de l'*échange de données*. Nous formalisons le problème de la compréhension de la relation entre deux instances comme celui de l'obtention d'une correspondance de schémas (un ensemble de formules dans un certain langage logique) dont la *réparation minimale* fournit une description parfaite de l'instance cible en fonction de l'instance source. Nous montrons que cette définition donne des résultats « intuitifs » quand on l'applique à des instances de bases de données qui sont dérivées l'une de l'autre par des opérations élémentaires (sélections, projections, jointures...). Nous étudions la complexité des problèmes de décision liés à cette notion d'optimalité dans le contexte de différents langages logiques (p. ex. dépendances acycliques génératrices de *n*-uplets). Pour ce problème particulier, notre contribution est strictement théorique ; des outils pratiques basés sur de telles idées sont laissés pour des travaux ultérieurs.

À la fin de la phase d'analyse, nous avons obtenu un certain nombre de services Web dont la sémantique est expliquée en termes d'un schéma global qui est spécifique à l'application. Dans le chapitre 6, nous abordons le modèle sémantique et la définition de services avec une notation logique à la Datalog qui prend en compte les types des entrées et sorties, les relations entre celles-ci, et l'imbrication des sorties des services. Nous montrons comment répondre à des requêtes en utilisant les services du Web caché. Cela nous conduit à étudier le problème de réponse à des requêtes en utilisant des vues, dans un contexte où les schémas locaux sont décrits comme des vues, avec des restrictions sur les motifs de liaison des accès à la source.

En résumé, soulignons les trois principales contributions de notre thèse :

1. Un cadre général pour la compréhension du Web caché de manière complètement automatique et non supervisée et, en particulier, des façons de découvrir la structure d'un formulaire et de pages de résultats et de représenter les services du Web caché, une fois leur analyse sémantique effectuée (voir les chapitres 1, 3, 4 et 6).
2. Un modèle arborescent probabiliste, avec capacités de requête et de mise à jour, avec une étude théorique et une implémentation (voir chapitre 2).
3. Un modèle théorique pour la découverte de relations entre schémas à partir d'instances de bases de données, ainsi qu'une analyse détaillée de sa complexité (voir chapitre 5).

L'annexe A est une traduction en français de l'introduction, du chapitre 1 et de la conclusion de cette thèse. Elle inclut un lexique anglais-français des termes et expressions techniques utilisés dans cette thèse. Nous mentionnons en appendice B d'autres travaux que nous avons effectués pendant notre thèse et qui n'ont pas de lien direct avec le Web caché.

Cadre général

Depuis sa création en 1991, le World Wide Web a considérablement crû, avec aujourd'hui une taille de plusieurs milliards de pages librement accessibles. Mais ce nombre ne couvre que le *Web de surface*, soit les pages Web accessibles en suivant des hyperliens. Une grande partie de l'information du Web se trouve dans le *Web caché*, aussi appelé *Web profond* ou *Web invisible*, qui fournit des points d'entrées à des bases de données accessibles grâce à des formulaires HTML ou des services Web. (Dans le reste de cette thèse, nous utiliserons le terme de *service (du Web caché)* de manière générique pour ces deux types de ressources.) Une étude [Bri00] de 2001 a estimé que les données du Web caché étaient approximativement 500 fois plus vastes que celles du Web de surface. Si de telles mesures sont sujettes à caution, le fait est qu'avec de l'information de première qualité (p. ex. les services de *Pages jaunes* ou le bureau du recensement des États-Unis), le Web caché est une source d'information inestimable.

Nous décrirons, dans ce chapitre, un cadre général, non supervisé et *entièrement automatique* pour la compréhension des services du Web caché. Cette approche est basée sur (i) des modules qui accomplissent des tâches variées, de la découverte de sources pertinentes à l'analyse de la sémantique de telles sources et leur indexation ; (ii) une architecture centrée sur le contenu, où des modules interagissent avec un entrepôt de contenu XML probabiliste. Nous considérons ici une approche *en compréhension*, plutôt qu'*en extension*, c'est-à-dire que nous n'avons pas pour but de récupérer et d'entreposer toute l'information des services du Web caché, mais d'indexer les services eux-mêmes afin de répondre à des requêtes de haut niveau d'un utilisateur en transmettant les requêtes aux services pertinents, en les traduisant en entrées attendues par chaque service, et en intégrant le résultat des services.

Pour illustrer, considérons un utilisateur du Web intéressé par les options disponibles pour une voiture donnée. Un moteur de recherche traditionnel retournerait un ensemble de pages HTML du Web en provenance, par exemple, du site Web du constructeur. Notre but est un système qui découvrirait (à l'avance) qu'un formulaire particulier fournit une telle information et qui, au moment de la requête, remplirait le formulaire et récupérerait la réponse précise. Une telle interprétation sémantique du Web caché requiert la découverte, l'analyse et l'indexation des ressources du Web caché. Elle demande de combiner des techniques de disciplines variées et, en particulier, d'apprentissage, de bases de données et de linguistique.

C'est évidemment un problème très large et difficile, d'autant plus que nous recherchons une approche non supervisée. Nous faisons donc l'hypothèse importante que nous sommes intéressé par des services d'un *domaine d'intérêt* donné, pour lequel nous avons une *base de connaissances du domaine*, sous une forme spécifique que nous décrivons dans la partie A.3. Il est certain qu'avec une interaction humaine, les techniques supervisées peuvent aller plus loin vers une meilleure compréhension du Web. Mais le type d'approche non supervisée que nous proposons (i) est utile au moins comme première étape, avant une intervention humaine ; (ii) est souvent la seule possible, dans des applications où des ressources humaines ne peuvent être utilisées ; (iii) est primordiale si nous voulons que le système puisse s'adapter à l'échelle et à la diversité du Web, ainsi qu'à son dynamisme.

Ce chapitre se concentre sur le cadre général, dont certains des composants sont décrits de manière plus détaillée dans les chapitres suivants. Nous nous intéressons d'abord aux travaux traitant du Web caché lui-même et de son analyse en partie A.1. Notre cadre général est ensuite présenté dans la partie A.2. Nous introduisons le type de connaissances du domaine considéré en partie A.3, avant de décrire les différents composants de notre architecture dans la partie A.4, en particulier ceux qui ne seront pas présentés dans un chapitre ultérieur.

A.1 Travaux similaires

A.1.1 Le Web caché

Les termes *Web caché* [RGM01], *Web profond* [Bri00] et *Web invisible* [RH05] ont été utilisés dans la littérature pour décrire plus ou moins la même chose : la partie du World Wide Web qui n'est pas accessible par des hyperliens, et qui est généralement construite à partir de bases de données. Les termes *caché* et *invisible* insistent sur le caractère inaccessible de cette information pour les moteurs de recherche, tandis que *profond* insiste sur le fait que cette information se trouve dans des bases de données derrière des formulaires, et requiert une exploration *plus profonde* que les explorations *de surface* habituellement effectuées par les moteurs de recherche. [Bri00] propose d'ailleurs l'analogie suivante entre information sur le Web et pêche en mer : bien que l'on puisse pêcher une certaine quantité de poisson en restant à la surface, une quantité bien plus importante est à pêcher à plus grande profondeur. À proprement parler, il serait peut-être plus approprié d'utiliser l'expression *Web profond* dans cette thèse, plutôt que *Web caché*, puisque nous mettons en général l'accent sur des bases de données accessibles à travers des formulaires, dont le contenu est parfois disponible depuis le Web de surface (le contenu de la plupart des bases de données commerciales de produits, par exemple, peut être parcouru dans son intégralité en suivant des liens, ceci pour attirer les utilisateurs de moteur de recherche sur ces sites Web). Nous choisissons cependant d'utiliser *Web caché* pour mettre en valeur le fait qu'une compréhension du Web caché est un moyen de dépasser les limitations des moteurs de recherche classiques. De plus, *Web caché* est une expression beaucoup plus mystérieuse qui suscite à coup sûr des interrogations de la part de néophytes, et, en Français, *Web profond* sonne beaucoup moins bien que *Web caché*.

Une étude de 2001 de la compagnie BrightPlanet [Bri00] a eu un large impact sur le développement de la recherche sur le Web caché. Cette étude utilise une analyse de corrélation des résultats de différents moteurs de recherche afin d'estimer la taille du Web caché ; ils découvrent ainsi qu'il contient entre 43 000 et 96 000 bases de données du Web, avec à peu près 500 fois plus de contenu que le Web de surface. Autrement dit, la plus grande partie (et de beaucoup) du contenu du Web n'est pas exploitable par les moteurs de recherche classiques ! Bien que ce type d'estimation soit intrinsèquement peu précis, d'autres travaux plus récents [CHL$^+$04, HPZC07] confirment cet ordre de grandeur avec une autre forme d'estimation (échantillonnage aléatoire d'adresses IP), et parviennent à un nombre d'approximativement 400 000 bases de données (ceci prend en compte la croissance du Web entre le moment des deux études). De plus, même les répertoires qui se spécialisent dans le recensement de bases de données du Web (un grand nombre d'entre eux sont cités dans [RH05][*]) ont une couverture assez faible des services du Web caché (15 % au mieux, cf. [HPZC07]). Ceci est une motivation claire pour des systèmes qui découvrent, comprennent et intègrent les services du Web caché.

A.1.2 Systèmes pour l'exploitation de l'information du Web caché

Une revue d'un certain nombre de systèmes traitant des services du Web caché est disponible dans [RH05]. Deux approches différentes existent : en extension (récupérer l'information du Web caché et l'entreposer en local pour l'utiliser) ou en compréhension (analyser les services pour en comprendre la structure, entreposer cette description, et l'utiliser pour transmettre les requêtes des utilisateurs à ces services). Le cadre que nous présentons dans cette thèse relève du deuxième type. Nous présentons plus bas les deux approches. Seuls les systèmes complets traitant du Web caché

[*] De tels répertoires ne font que donner une liste de ces bases de données, il ne fournissent pas de manière de les interroger par une interface commune ou d'intégrer leurs résultats.

sont présentés ; les travaux plus spécifiques seront abordés dans la suite de cette thèse, aux chapitres appropriés.

L'un des premiers travaux pratiques sur le Web caché [RGM01] suit la stratégie en extension : HiWE est un système général pour représenter des formulaires et mettre en correspondance les champs de formulaires avec des concepts (ceci utilise une annotation humaine de manière intensive), afin de pouvoir récupérer des pages de résultats qui sont ensuite stockées dans un index local. Il est intéressant de remarquer que cette approche en extension est préférée par des chercheurs du principal moteur de recherche Google [Goo] dans [MHC$^+$06] (les auteurs parlent d'approche *surfaçante*) pour des raisons d'indépendance du domaine, de résultats et d'efficacité, même si la plupart de la sémantique des services est perdue ce faisant, et qu'une lourde charge est mise sur la source dont le contenu est entièrement aspiré.

La plupart de la recherche active sur l'indexation en compréhension du Web caché [HC03, HMYW04, WYDM04, ZHC04, CHZ05, ZHC05, WDYM05, WDY06, CCZ07] vient d'un groupe de l'université d'Illinois à Urbana–Champaign, avec quelques collaborations externes, en particulier avec l'université de Binghamton. Deux systèmes qui ont été développés dans ce contexte sont MetaQuerier et WISE-Integrator. Le but est, comme dans notre cas, de découvrir et d'intégrer des sources du Web caché, afin de les interroger de manière uniforme. L'accent est fortement mis, dans ces travaux, sur l'appariement de schémas entre des sources différentes. MetaQuerier [HC03, CHZ05] utilise une approche *holistique* : apparier le schéma de plusieurs sources différentes simultanément, plutôt que de les apparier deux à deux. En particulier, le schéma global n'est en général pas prédéfini mais résulte d'une classification des champs de différentes sources. Ces travaux se concentrent essentiellement sur l'analyse de la syntaxe des formulaires et la découverte des concepts liés aux différents champs (des travaux récents [WDYM05, CCZ07] traitent également de l'analyse des pages de résultats). Il est en fait assez compliqué de comprendre les relations qui existent entre toutes les publications cités plus haut, mais [CHZ05] est ce qui s'approche le plus d'une vue de haut niveau sur l'ensemble du système. Nous examinerons certains de ces travaux plus avant dans les chapitres suivants.

A.2 Description du cadre

Décrivons notre cadre général. Une architecture fonctionnelle simplifiée est présentée en figure A.1. Les composants seront détaillés en partie A.4. Les différentes étapes sont les suivantes :

1. Les services pertinents sont d'abord acquis depuis le Web caché (voir partie A.4.1).
2. La syntaxe de ces services est analysée. Ceci implique également de mettre en correspondance les entrées et sorties des services d'une part, et les concepts du domaine d'intérêt d'autre part (voir partie A.4.2).
3. La sémantique des relations entre entrées et sorties des services du Web caché est ensuite recherchée (voir partie A.4.3).
4. Une fois la sémantique d'un service pleinement comprise, celui-ci est indexé. Cet index sert à répondre à des requêtes de haut niveau qui sont posées directement dans le langage de l'ontologie du domaine (voir partie A.4.4).

Nous avons élaboré un système d'après ces principes. Certains des composants sont encore en développement. Nous présenterons brièvement la manière dont ces composants s'intègrent les uns aux autres en conclusion de cette thèse.

Le processus général décrit en figure A.1 peut être vu comme la collaboration d'agents indépendants tels que des robots parcourant le Web ou des extracteurs d'information, qui manipulent (insèrent,

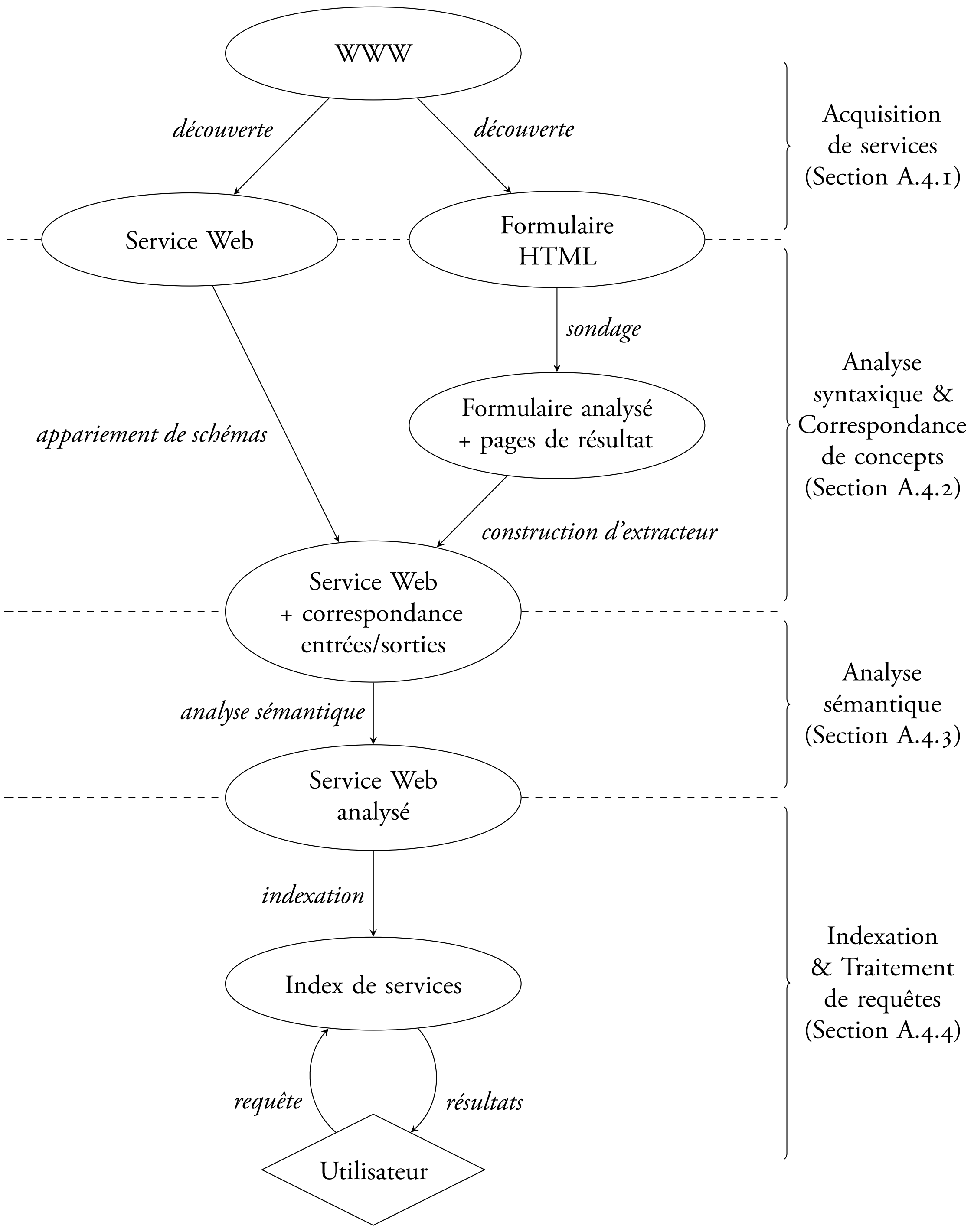

WWW
découverte
découverte
Service Web
Formulaire HTML
sondage
Formulaire analysé + pages de résultat
appariement de schémas
construction d'extracteur
Service Web + correspondance entrées/sorties
analyse sémantique
Service Web analysé
indexation
Index de services
requête
résultats
Utilisateur
Acquisition de services (Section A.4.1)
Analyse syntaxique & Correspondance de concepts (Section A.4.2)
Analyse sémantique (Section A.4.3)
Indexation & Traitement de requêtes (Section A.4.4)

Figure A.1 – Processus de compréhension du Web caché

suppriment) de l'information sur le Web caché contenue dans un entrepôt commun. L'information obtenue par la plupart de ces modules est imprécise. Par exemple, le module de sondage aura peut-être déterminé qu'un champ de formulaire donné représente le prénom d'une personne avec un certain niveau de confiance. De plus, la provenance de l'information est souvent importante. Enfin, remarquons que le processus de la figure A.1 apparaît plutôt séquentiel, alors qu'en réalité il ne l'est pas. Par exemple, l'information obtenue par des techniques de traitement du langage naturel pendant l'analyse sémantique pourrait être utile pour améliorer la qualité des extracteurs obtenus dans une étape antérieure.

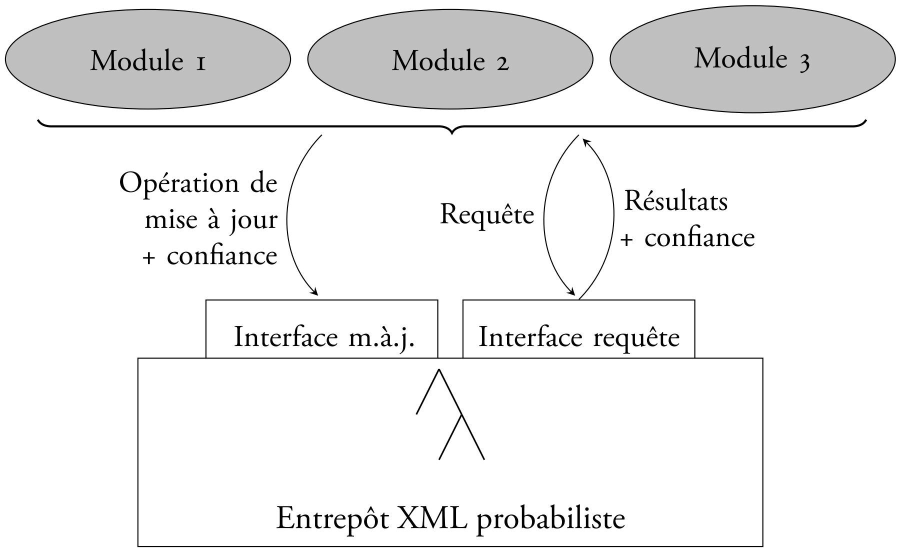


Figure A.2 – Entrepôt de données imprécises

On pourrait utiliser une approche purement séquentielle pour gérer la collaboration des différents agents. Mais comme leurs nombre et rôle ne sont pas fixés, et que leur ordonnancement peut être assez complexe, nous préférons utiliser un système *centré sur le contenu*, comme celui représenté en figure A.2. Nous suivons ainsi une approche dans le style de [ANR05]. Plus précisément, le système est construit sur un entrepôt de contenu semi-structuré, avec des possibilités de requête et de mise à jour qui gèrent information imprécise et provenance. L'information probabiliste est placée dans l'entrepôt et le suivi de la confiance est directement fourni par les interfaces de requête et de mise à jour. Chaque résultat de requête et opération de mise à jour vient avec une information de confiance.

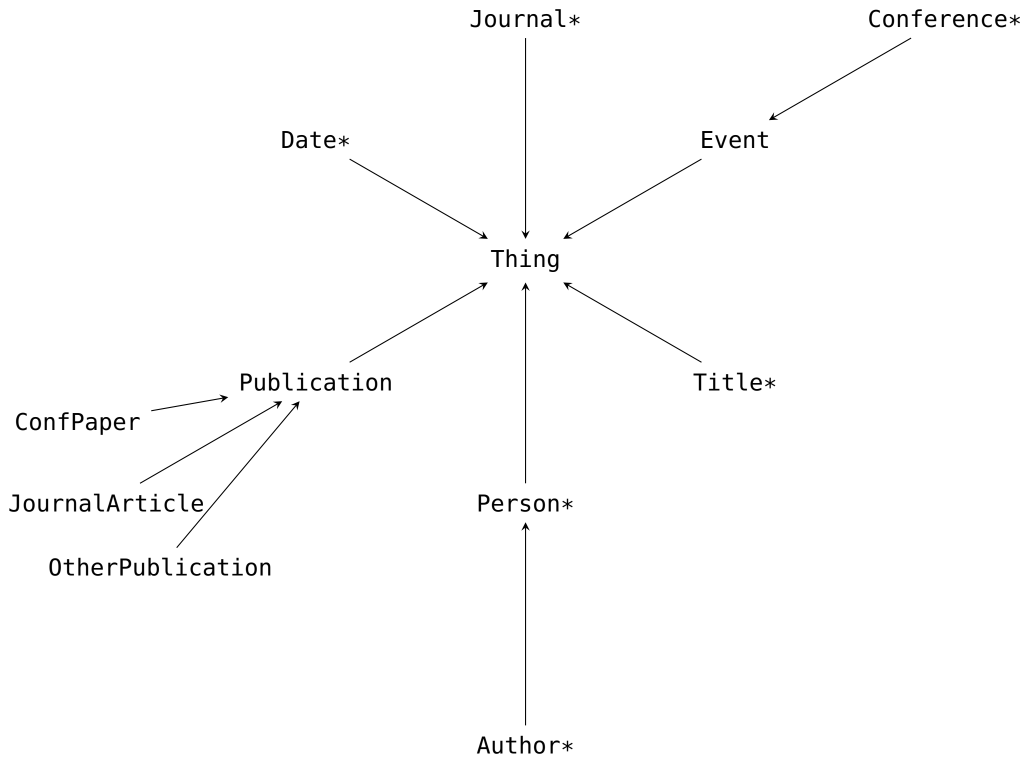
Le modèle sous-jacent à cet entrepôt semi-structuré imprécis, les *arbres probabilistes*, est décrit dans le chapitre 2. Il se base sur l'utilisation de variables d'événements probabilistes qui sont associées aux nœuds de l'arbre de données. Le modèle d'arbre probabiliste est *complet* (toutes les combinaisons de mondes possibles peuvent être représentées comme un arbre probabiliste), *concis* (par exemple, la taille de la représentation croît linéairement quand des nœuds imprécis sont insérés) et dispose d'un langage de requête puissant (les requêtes à motif d'arbre avec jointures) et permet des séquences arbitraires d'insertions et de suppressions.

A.3 Base de connaissances du domaine

Nous présentons dans cette partie la base de connaissances du domaine dont nous avons besoin pour décrire un domaine d'intérêt spécifique. Nous insistons sur le fait que toutes les techniques décrites dans cette thèse, même si elles sont illustrées avec l'exemple du domaine des publications scientifiques, peuvent être appliquées à n'importe quel domaine, si l'on dispose de la base de connaissances décrite ici.

La base de connaissances nécessaire peut être décomposée en deux parties différents : une *ontologie du domaine* et des *instances du domaine*.

Ontologie du domaine. L'ontologie du domaine est formée d'un ensemble de *noms de concepts*, organisés en un graphe orienté acyclique de relations *SorteDe*, et d'un ensemble de *noms de relations*, venant avec leur arité et leur type (le type d'une relation n-aire R est un n-uplet de noms de concepts). Un sous-ensemble de l'ensemble des noms de concepts est distingué comme ensemble des concepts *concrets* (les concepts concrets sont ceux pour lesquels des *instances* existent, voir ci-dessous).



```
HasTitle           (Paper        , Title     )
PublishedIn        (Paper        , Date      )
WrittenBy          (Paper        , Author    )
PublishedInJournal(JournalArticle, Journal   )
PublishedInConf    (ConfPaper    , Conference)
```

Figure A.3 – Ontologie élémentaire pour le domaine des publications de recherche

Donnons pour illustration l'ontologie du domaine des publications scientifiques représentée en figure A.3. Les relations SorteDe peuvent être interprétées comme suit : un `ConfPaper` est une sorte de `Publication`, qui est une sorte de `Thing`. Aucun de ceux-ci n'est un concept concret, contrairement à `Title` (les astérisques indiquent les concepts concrets). De plus, le nom de relation `HasTitle` définit une relation binaire entre une instance du concept `Paper` (ou de l'un de ses concepts subsumés) et une instance du concept `Title`.

De telles ontologies sont des exemples simples d'ontologies qui sont entre autres utilisées dans le domaine du Web sémantique. Elles sont en général exprimées dans des langages normalisés tels RDF Schema [W3C04a] ou OWL-Lite [W3C04b]*.

Instances du domaine. Nous traitons ici de représentations concrètes (c'est-à-dire comme chaînes de caractères) de concepts concrets. Observons par exemple que les chaînes *`June 2007`* et *`07/06`* peuvent toutes deux représenter la même instance du concept `Date`. Les instances du domaine sont des chaînes de caractères qui représentent des instances de concepts concrets du domaine. Plus précisément, pour chaque concept concret c, nous avons besoin des éléments suivants :

– les mots utilisés dans les instances du concept, avec une distribution de fréquence approximative ;
– un modèle probabiliste approximatif des instances du concept, c'est-à-dire une manière d'affecter à une chaîne de caractères donnée une probabilité que cette chaîne représente une instance de c.

Au minimum, nous avons besoin d'une liste de chaînes qui représentent des instances de concepts pour chaque concept. La manière la plus simple d'obtenir les deux éléments de notre base de connaissances est alors la suivante :

– prendre l'ensemble des mots apparaissant dans ces chaînes, avec leur fréquence respective, comme distribution de fréquence approximative ;
– pour chaque chaîne donnée s, si s peut représenter une instance des concepts $c_1 \dots c_n$, affecter à s la probabilité $1/n$ que s représente une instance de chaque concept c_i.

Insistons sur le fait que nous avons uniquement besoin d'approximations de cette distribution de fréquence et de ce modèle probabiliste. Pour certains concepts, nous pouvons fournir une description plus élaborée (voir ci-dessous pour les concepts de notre exemple d'ontologie), mais il n'est en aucun cas nécessaire de fournir une description parfaite des instances de concepts. Nous aborderons au chapitre 3 la manière dont les fréquences de mots peuvent être utilisées pour comprendre la structure d'un formulaire HTML, et au chapitre 4 la manière dont le modèle probabiliste nous sert à extraire de l'information des pages de résultats à un formulaire.

Décrivons les instances du domaine que nous utilisons dans le cas du domaine des publications scientifiques. Nous avons téléchargé le contenu de la bases de données DBLP [Ley], sous la forme d'un fichier XML, de `http://dblp.uni-trier.de/xml/` et l'avons utilisé pour engendrer nos instances du domaine :

– Pour les concepts `Title`, `Journal`, `Conference`, nous avons utilisé les techniques élémentaires décrites plus haut.
– Pour le concept `Date`, nous fournissons une fonction de reconnaissance d'entités appropriée (sous la forme d'un ensemble d'expressions rationnelles qui décrit les dates mensuelles et annuelles).

* RDF Schema et OWL ne considèrent tous deux que des relations binaires, alors que nous traitons de relations d'arité quelconque, même si les relations binaires sont les plus courantes. Remarquons qu'une relation n-aire peut être encodée par des relations binaires avec une *réification* si besoin est.

– Pour le concept `Author`, nous avons extrait les prénoms et noms de famille des noms de personnes de DBLP avec des heuristiques, et utilisons des expressions rationnelles pour décrire les façons de recombiner prénoms et noms de famille (ce qui donne, par exemple, les chaînes de caractères *`Abiteboul`*, *`Pierre Abiteboul`*, *`Abiteboul Pierre`*, *`Abiteboul Pierre Paul`* ou *`Abiteboul, Pierre`*, avec des probabilités variées, à partir du nom de famille *`Abiteboul`* et des prénoms *`Pierre`* et *`Paul`*).

Dans les deux derniers cas, les probabilités associées aux expressions rationnelles sont choisies de manière *ad hoc*. Idéalement, elles devraient provenir d'une évaluation statistique sur un grand corpus.

Les différents modules que nous décrivons dans cette thèse utilisent uniquement ces formes de connaissances (ontologie du domaine, avec un graphe acyclique de concepts et des relations typées, et instances du domaine, avec une distribution de fréquence des mots et un modèle probabiliste pour chaque concept concret). Il est possible, cependant, que certaines étapes du processus pour lesquelles nous ne fournissons pas de systèmes (en particulier, l'acquisition de services et l'analyse sémantique) requièrent des connaissances supplémentaires (par exemple, des mots-clefs liés au domaine d'intérêt, ou une description linguistique des noms de relation).

A.4 Modules

Nous décrivons ici les différents modules qui apparaissent en figure A.1. Certains d'entre eux sont le sujet de chapitres distincts.

A.4.1 Acquisition de services

La première phase du processus est d'acquérir de l'information. Cette acquisition d'information vient d'abord d'une publication par des utilisateurs. Le système acquiert également de l'information en utilisant des moteurs de recherche ou en explorant le Web. Le système est principalement intéressé par :

– des formulaires HTML [W3C99] ; ils sont extrêmement utilisés et des millions peuvent être trouvés sur le Web.
– des services Web [W3Ca] ; on les trouve en général dans des entrées d'annuaire UDDI. Leur description est habituellement donnée en WSDL [W3C01]. On en trouve de plus en plus sur le Web.

Le système est également intéressé par les ressources *en extension*, tels des documents XML et HTML contenant de l'information utile. Parce que l'accent est mis sur le Web caché, les informations en extension ne sont pas notre principal intérêt ici. Toutefois, il faut qu'elles puissent être importées dans le système. De plus, comme nous le verrons au chapitre 6, les documents sont utiles pour *amorcer* l'utilisation des services.

Il est important de noter que nous nous intéressons à des services qui fournissent de l'information, comme le site Web de DBLP, et non à des services avec effets de bord, comme des services de réservation ou des interfaces de gestion de listes de discussion. En particulier, la norme Internet sur HTTP 1.1 [IET99] spécifie que la méthode HTTP GET *devrait* être sans effet de bord, et cette règle est en général suivie. Ainsi, les formulaires utilisant la méthode GET sont en général acceptables. Par ailleurs, les formulaires HTML qui utilisent la méthode POST sont souvent utilisés avec effets de bord. Notons cependant que ce n'est pas toujours le cas. Par conséquent, quand nous explorons le Web à la recherche de services, nous devons détecter les services avec effets de bord afin de les exclure, et ce n'est pas une tâche facile. Nous devrons utiliser à cet effet des heuristiques, comme le fait d'éviter les services nécessitant l'entrée d'une adresse e-mail, d'un numéro de carte bleue ou d'un mot de passe (sauf si ce service a été explicitement déclaré par un utilisateur du système).

Comme mentionné précédemment, nous ne nous intéressons qu'aux services qui sont pertinents par rapport à un domaine d'intérêt donné. Si, par exemple, nous explorons le Web à la recherche de services, nous devons « guider » cette exploration vers le domaine d'intérêt ; ce type d'*exploration guidée* du World Wide Web a été étudié dans [CvdBD99, DCL$^+$00]. Une approche intéressante au problème spécifique de l'exploration guidée pour découvrir des formulaires est présentée dans [BF05]. Les auteurs combinent des techniques classiques d'exploration guidée, avec une catégorisation des pages en fonction du domaine d'intérêt, avec deux sources de retours qui servent à la contrôler : une catégorisation des formulaires qui vérifie si un formulaire donné est pertinent, et une catégorisation des liens qui considère des caractéristiques de l'historique des liens suivis pour arriver à la page courante (l'hypothèse est qu'il y a des caractéristiques distinguant les liens pertinents des non pertinents dans le chemin suivi à partir de la racine d'un site Web, par exemple, jusqu'à un formulaire de recherche dans une base de données). Les auteurs montrent que cela permet de trouver des bases de données du Web cachées avec des ressources d'exploration moindres que des explorations complètes ou des techniques génériques d'exploration guidée. Remarquons cependant que ces trois catégorisations utilisent des exemples annotés de pages, formulaires et chemins de liens.

Les services présentant un intérêt peuvent également être obtenus en interrogeant des moteurs de recherche généraux avec des mots-clefs du domaine. Par exemple, les requêtes « publication database » ou « publications advanced search » suffisent déjà à renvoyer des services hautement pertinents pour le domaine des publications scientifiques. Les annuaires de bases de données du Web peuvent également être utilisés, même si nous avons déjà constaté la faiblesse de leur couverture.

L'acquisition de sources d'information du Web caché pertinent pour un domaine d'intérêt n'est pas l'une des étapes du processus sur lesquelles nous avons spécifiquement travaillé. Nous ne détaillerons pas davantage ce sujet dans cette thèse.

A.4.2 Analyse syntaxique et correspondance de concepts

Une fois que des services sont identifiés, leur structure doit être comprise si l'on veut pouvoir réellement les utiliser. Par *structure*, nous voulons dire le type des entrées et sorties attendues et produites par chaque service, la manière de fournir ces entrées au service, ainsi que la façon de récupérer les sorties du service. Un problème voisin est celui de faire correspondre les entrées et sorties avec les concepts de notre ontologie. En fait, comme nous le verrons, nous avons besoin d'accomplir ces deux étapes en même temps dans le cas des formulaires HTML.

Considérons dans un premier temps les services Web, pour lesquels le problème est plus simple. Une description WSDL d'un service Web est une description formelle des types d'entrées qu'il attend et des sorties qu'il produit. Les noms des entrées et sorties, cependant, sont arbitraires, et peuvent ne pas avoir de lien direct avec les concepts du domaine. Mettre en correspondance les concepts et ces noms est un exemple de problème d'*appariement de schémas* ; nous n'apportons pas de contribution directe à ce sujet, et nous renvoyons le lecteur à [RB01] pour une revue des techniques d'appariement automatique de schémas. Les types concrets (en XML Schema) des entrées sont une autre source d'information : une expression rationnelle « `\(\d{3}\) \d{3}-\d{4}` » représente probablement un numéro de téléphone (des États-Unis), tandis qu'un type simple `xs:date` représente une date.

Considérons maintenant les formulaires HTML. La structure du formulaire lui-même mais aussi celle des pages de résultats (qui doivent être générées en *sondant* le formulaire, c'est-à-dire en le soumettant avec certaines valeurs entrées) doivent être comprises. Il est ainsi nécessaire, afin d'obtenir des pages de résultats, d'avoir une certaine compréhension des concepts avec lesquels les champs du formulaire sont en correspondance. En d'autres termes, l'analyse structurelle et la mise en correspondance de concepts doivent être accomplies simultanément. Nous présentons un processus en deux étapes pour résoudre ce problème :

1. Le chapitre 3 montre comment utiliser des heuristiques afin de faire correspondre les concepts de l'ontologie du domaine avec les champs d'un formulaire, et comment confirmer ces annotations en sondant ces champs.
2. Le chapitre 4 présente une approche pour extraire de l'information des pages de résultats ; un programme d'annotation étiquette la page avec des instances de concepts du domaine, et cette annotation imprécise et incomplète est généralisée comme entrée d'un système de construction d'extracteur utilisant de l'apprentissage.

A.4.3 Analyse sémantique

La sémantique d'un service dont est fournie la description syntaxique doit être analysée afin de comprendre les relations sémantiques entre ses entrées et sorties ; le but est ici d'exprimer la sémantique de ce service de manière formelle.

Nous présentons au chapitre 6 un modèle sémantique pour décrire des services. La manière d'obtenir cette description sémantique est toutefois assez complexe. Dans certains domaines, il y a peu d'ambiguïtés : par exemple, dans le domaine des publications scientifiques, si nous avons les concepts `Title`, `Person`, `Journal` et `Date`, il est très probable que le titre `Title` est celui d'un article écrit par la personne `Person` dans un journal `Journal` publié à la data `Date`. Ce n'est pas obligatoire cependant : la date pourrait être la date de dernière modification de la publication, et la personne pourrait être l'éditeur du journal. Dans d'autres domaines, des problèmes plus compliqués peuvent également survenir. Considérons par exemple une base de données de généalogie. Il est ici très important de savoir si la relation `FatherOf` est entre P et P' ou entre P' et P. Les outils qui sont à notre disposition pour résoudre ces problèmes sont des techniques élémentaires de traitement du langage naturel, ou de la sélection de mots-clefs dans la description du service, par exemple dans la page du formulaire ou dans les pages pointant vers cette page.

Notre contribution à la question de l'analyse sémantique réside dans le modèle théorique décrit dans le chapitre 5. Nous y présentons une manière de formaliser la notion de relation *optimale* entre deux instances de bases de données. Cette partie de notre travail n'a pas donné lieu à implémentation.

A.4.4 Indexation et traitement de requêtes

En supposant que nous avons un ensemble de services dont la sémantique est décrite, la dernière étape de notre processus est de fournir à l'utilisateur la possibilité d'utiliser ces services pour répondre à des requêtes de haut niveau, dans la langage de l'ontologie du domaine. Cela signifie indexer les services et traduire les requêtes sur l'ontologie du domaine en requêtes sur les services pertinents. Ce problème dépend fortement de la représentation de la sémantique d'un service, et la réflexion sera donc reportée au chapitre 6, où nous présentons le modèle sémantique.

Conclusion

Dans ce chapitre, nous avons présenté un processus général et entièrement automatique pour l'interprétation sémantique du Web caché. Ce processus est basé sur des agents indépendants qui accomplissent différentes tâches, telles l'acquisition d'information et son enrichissement, et participent à la construction d'un entrepôt probabiliste semi-structuré centré sur le contenu. Nous avons décrit brièvement les différentes étapes du processus. Pour certaines d'entre elles, seules des idées prospectives sont données, étant donné que nous ne prétendons pas arriver à une solution complète au problème de l'exploitation d'information sur le Web caché.

Les chapitres qui suivent vont détailler plus avant certains composants, à commencer par le modèle de données probabiliste qui sous-tend l'entrepôt de contenu.

Conclusion globale

Synthèse

Examinons dans un premier temps de quelle manière les divers composants dont il a été question dans cette thèse peuvent être articulés pour obtenir un système complet de compréhension du Web caché. L'architecture générale a été abordée dans le chapitre 1 et, en particulier, les figures A.1 et A.2 illustrent l'interaction des composants. En pratique, en nous concentrant sur les parties de cette architecture que nous avons effectivement développées, nous avons le processus suivant :

1. Construire la base de connaissances du domaine comme décrit en partie A.3.
2. Assembler des formulaires qui sont des points d'entrée de services du Web caché d'une certaine façon (p. ex. manuellement) et les entrer dans l'entrepôt probabiliste.
3. Transformer chacun de ces formulaires en un service Web, comme indiqué en parties 3.6 et 4.4. Insérer la description du service résultant, avec des valeurs de confiance appropriées, dans l'entrepôt probabiliste.
4. Lancer un programme *ad hoc* qui interroge et met à jour l'entrepôt probabiliste de façon à ajouter les relations sémantiques entre entrées et sorties (par exemple, si l'entrée d'un service est un `Title` et que la sortie est un ensemble d'`Author`, ce programme déclarera que ces derniers sont les auteurs d'une publication dont le titre est l'entrée du service, avec une forte probabilité).
5. Présenter à l'utilisateur une interface de requête de haut niveau sur l'ontologie du domaine.
6. Traduire la requête de l'utilisateur en une requête sur l'ensemble des services connus, comme traité en partie 6.6, évaluer cette requête, et retourner les résultats (probabilistes) à l'utilisateur.

Nous travaillons actuellement à l'implémentation de l'intégralité de ce processus.

Perspectives

En conclusion de cette thèse, nous aimerions relever certains problèmes importants dans le contexte de la compréhension du Web caché. Nous classifions ces problèmes en : (i) ceux que nous traitons déjà, (ii) ceux qui demandent, à notre avis, un certain effort mais devraient pouvoir être résolus sans difficulté intrinsèque, et (iii) ceux qui nécessitent des travaux de recherche conséquents.

Commençons par les travaux en cours :

Application pratique du chapitre 5. Le modèle théorique du chapitre 5 n'a pas encore d'application pratique. Nous aimerions explorer deux manières différentes de combler ce manque : (i) une implémentation directe du modèle, en dépit des résultats de complexité négatifs, avec des heuristiques appropriées pour approximer efficacement le coût d'une correspondance de schémas ; (ii) les relations avec le domaine de la programmation logique inductive.

Réponse à des requêtes à partir de vues. Nous sommes en train de travailler sur le problème de réponse à des requêtes à partir de vues, les vues étant ici les services du Web caché dont la sémantique est décrite. Nous mentionnons en partie 6.6 une direction possible de recherche basée sur les techniques dites Magic d'évaluation efficace de programmes Datalog.

Intégration système et démonstration. Nous avons évoqué au début de cette conclusion la manière dont les différents modules peuvent être articulés afin de former un système complet pour l'intégration des services du Web caché. Nous travaillons actuellement à cette intégration, avec pour but d'obtenir une démonstration du système entier. Nous prévoyons également d'appliquer le système à un autre domaine (peut-être celui des services météorologiques ou des annuaires) afin d'en valider la généralité.

Les problèmes suivants semblent être raisonnablement résolubles, avec éventuellement un certain effort :

Acquisition de services. Nous avons évoqué cette question en partie A.4.1. Nous pensons qu'une combinaison des heuristiques et des techniques d'exploration guidée du Web qui y ont été présentées peut être suffisante pour acquérir un nombre satisfaisant de services pertinents. Ceci requiert l'implémentation et la combinaison du tout, ce qui n'est pas complètement direct.

Des types de formulaires plus généraux. Nous avons effectué un certain nombre d'hypothèses sur la structure des formulaires que nous traitions au chapitre 3. Bien que l'analyse d'un formulaire arbitraire sans structure préétablie soit une tâche très difficile, un certain nombre de ces hypothèses pourraient être supprimées afin de gérer, par exemple, des formulaires avec des entrées obligatoires ou des formulaires qui utilisent des opérateurs booléens. Cela serait utile car des expériences informelles montrent qu'il n'y a pas tant d'hétérogénéité dans la structure des formulaires du Web. Cela impliquerait une reconnaissance d'un *modèle* général qu'un formulaire respecte et une analyse des différents champs en fonction de ce modèle.

Meilleur programme d'annotation. Il est toujours possible d'améliorer l'annotation que nous avons décrite au chapitre 4. En particulier, des systèmes de reconnaissance d'entités (voir par exemple [8] qui pourrait également être utilisé pour gérer d'autres langues que l'anglais), qui sont à la fois généralistes et efficaces, pour des dates, des noms de personnes, des adresses, et ainsi de suite, peuvent donner de meilleurs résultats que ceux que nous avons utilisés. De plus, de l'information linguistique pourrait également être utilisée pour, par exemple, annoter des groupes nominaux de longueur appropriée comme titres potentiels.

Enfin, présentons quelques problèmes ouverts dont nous pensons qu'ils sont importants pour la compréhension du Web caché, et pour lesquels nous ne connaissons pas de solution.

Mise à jour des valeurs de confiance. Remarquons que dans le modèle d'arbres probabilistes décrit au chapitre 2, la probabilité qu'un nœud donné est dans l'arbre peut décroître (si ce nœud est supprimé avec conditions) mais jamais croître (même si on peut le simuler en ajoutant un nouveau nœud avec la même étiquette et les même descendants). Cela reflète le fait que des modifications arbitraires et *a posteriori* des probabilités n'ont en général pas grand sens ; l'augmentation (ou la diminution) de la confiance dans un fait est pourtant quelque chose dont on a parfois besoin dans un système d'intégration. Il faudrait formaliser cette opération de manière propre, et considérer la manière de l'appliquer sur un arbre probabiliste. Nous verrons également plus loin (sous Déduplication) un problème lié, qui pourrait fournir des valeurs de confiance mises à jour.

***k* premiers résultats probabilistes.** L'une des limitations les plus apparentes du modèle probabiliste tel qu'il est décrit au chapitre 2 est le fait que le résultat d'une requête est donné (et calculé) dans son intégralité. Quand ce résultat est trop grand, ou quand on est juste intéressé par les résultats les plus probables, il serait intéressant d'obtenir les *k* premiers résultats probabilistes de manière efficace. Nous n'avons pas de solution à cela, car les méthodes classiques de calcul des *k* premiers résultats [FLN03] ne sont pas directement applicables.

Un système d'apprentissage plus adapté. Les résultats expérimentaux que nous avons présentés au chapitre 4 montrent qu'il est possible d'utiliser des techniques d'apprentissage pour apprendre la structure sous-jacente à une annotation imprécise et imparfaite. Certains des résultats sont bons, tandis que d'autres sont quelque peu décevants. Nous pensons que la raison principale en est le fait que les champs aléatoires conditionnels (de même que, à notre connaissance, toutes les autres techniques d'apprentissage) sont conçus pour fonctionner dans un contexte où l'annotation initiale est censée être parfaite. Par conséquent, il y a des risques de trop « coller » à celle-ci. Un modèle d'apprentissage plus adapté essaierait de minimiser la longueur de description de l'extracteur obtenu, de manière similaire à ce qui est décrit au chapitre 5. La manière d'accomplir ceci est loin d'être évidente.

Extraction de *n*-uplets. La méthode que nous avons décrite au chapitre 4 afin d'extraire des *n*-uplets des données annotées individuelles est en quelque sorte *ad hoc* et peu robuste. Elle est incapable de tenir compte des pages où certains éléments d'un *n*-uplet ont été factorisés. Nous avons tenté d'utiliser une technique d'apprentissage qui extrait simultanément les *n*-uplets [GMTT06], mais avec moins de succès que les champs aléatoires conditionnels.

Déduplication. Des services différents peuvent contenir des informations légèrement différentes à propos de la même entité (par exemple des noms d'auteurs ou de conférences légèrement différents pour la même publication). Pour éviter de présenter à un utilisateur une liste de résultats quasi-identiques, il est nécessaire d'appliquer une étape de *déduplication* qui identifiera et fusionnera les doublons. Cette déduplication est également nécessaire dans d'autres contextes, par exemple quand nous identifions des constantes comme dans le chapitre 5. Nous n'avons pas de solution. Un problème intéressant qui est lié à cela est celui de la *corroboration de données* : quand des sources multiples déclarent des faits différents (par exemple, quand des bases de données différentes ont des informations diverses sur la même entité), quelle est la probabilité globale que le fait soit vrai (et que la source soit digne de confiance) ? Une première approche à ce problème est présentée dans [YHY07].

Analyse sémantique. Afin d'identifier les relations entre entrées et sorties d'un service, outre les méthodes que nous avons commencé à explorer au chapitre 5, qui sont uniquement basées sur les constantes qui apparaissent, des techniques utilisant le contexte d'un service (en particulier son contexte en langage naturel) pourraient être utilisées pour dériver de telles relations, par exemple pour comprendre que la sortie d'un service de généalogie est le père de l'entrée correspondante. C'est un problème difficile, qui devrait probablement être attaqué à l'aide d'une combinaison de techniques de traitement du langage naturel et d'apprentissage.

Cette liste de problèmes ouverts ou partiellement résolus n'est bien sûr pas exhaustive.

Lexique anglais-français

binding pattern	motif de liaison
(to) bootstrap	amorcer
classification	catégorisation
clustering	classification
(to) crawl the Web	explorer le Web
concept mapping	correspondance de concepts
conditional random field	champ aléatoire conditionnel
content-centric	centré sur le contenu
data exchange	échange de données
domain instances	instances du domaine
domain ontology	ontologie du domaine
deep Web	Web profond
domain knowledge	base de connaissances du domaine
extensional	en extension
focused crawling	exploration guidée
gazetteer	programme d'annotation
hidden Web	Web caché
HTML form	formulaire HTML
IsA	SorteDe
intensional	en compréhension
invisible Web	Web invisible
join	jointure
probing	sondage
prob-tree	arbre probabiliste
query	requête
(to) query	interroger
search engine	moteur de recherche
schema mapping	correspondance de schémas
schema matching	appariement de schémas
side effect	effet de bord
surface Web	Web de surface
tree-pattern query	requête à motif d'arbre
tuple-generating dependencies	dépendances génératrices de *n*-uplet
update	mise à jour
warehouse	entrepôt
Web service	service Web
wrapper	extracteur

Appendix B

Other Works

During the three years of my PhD thesis, I also had the chance to work on other research topics, for some started before those presented in this thesis. I shortly describe next these projects.

Data Warehousing

XML Warehousing and Data Integration for Social Scientists [7]

We describe a novel application of XML and Web based technologies: a sociological study of the W3C standardization process. We introduce a new methodology and tools, to be used by sociologists to study standardization processes. We illustrate them by considering the W3C XQuery Working Group. Information technology has received little attention from sociologists, yet standardization such as that of the Web is a crucial issue, both economical and political. Our approached is based on the use of a semi-structured content warehouse. We introduce a modeling and querying approach of an XML content warehouse, and show it produces high added-value information. This information is used to conduct a preliminary sociological analysis of the XQuery standardization process.

Graph and Web Mining

Similarity between Nodes in Graphs. Application to Synonym Extraction [2, 1]

This work is a follow-up of my master's internship at Université catholique de Louvain in 2001.

We introduce a concept of similarity between vertices of directed graphs. Let G_A and G_B be two directed graphs with, respectively, n_A and n_B vertices. We define an $n_B \times n_A$ similarity matrix S whose real entry s_{ij} expresses how similar vertex j (in G_A) is to vertex i (in G_B): we say that s_{ij} is their similarity score. The similarity matrix can be obtained as the limit of the normalized even iterates of $S_{k+1} = BS_kA^T + B^TS_kA$, where A and B are adjacency matrices of the graphs and S_0 is a matrix whose entries are all equal to 1. In the special case where $G_A = G_B = G$, the matrix S is square and the score s_{ij} is the similarity score between the vertices i and j of G. We point out that Kleinberg's "hub and authority" method to identify Web pages relevant to a given query can be viewed as a special case of our definition in the case where one of the graphs has two vertices and a unique directed edge between them. In analogy to Kleinberg, we show that our similarity scores are given by the components of a dominant eigenvector of a nonnegative matrix. There are many potential applications of our similarity concept. As an illustration, we consider the automatic extraction of synonyms in a monolingual dictionary.

Identifying Logical Web sites with Flow Simulation [9]

This work is a follow-up of my master's thesis work at INRIA Futurs in 2003.

We present a method to discover the set of Web pages contained in a logical Web site, based on the link structure of the Web graph. Such a method is useful in the context of Web archiving and Web site importance computation. To identify the boundaries of a Web site, we combine the use of an online version of the preflow-push algorithm, an algorithm for the maximum flow problem in traffic networks, and of the Markov CLuster (MCL) algorithm. The latter is used on a crawled portion of the Web graph in order to build a seed of initial Web pages, a seed that is then extended using the former. An experiment on a subsite of the INRIA Web site is described.

Discovering Similar Nodes in a Graph. Application to Wikipedia [3]

We introduce a new method for finding nodes semantically related to a given node in a hyperlinked graph, namely the Green method, based on classical Markov chains. It is generic, adjustment-free and easy to implement. We test it in the case of the hyperlink structure of the English version of an on-line encyclopedia, namely Wikipedia. We present an extensive comparative study of the performance of our method compared to several other classical methods. The Green method is found to have both the best average results and the best robustness.

Predicting the Rank of a Web Page [12]

Predicting the rank of a Web page is a challenging problem that is recently arising as a research issue both in academia and the industry. We propose a method $(PR)^2$ for predicting the ranking position of a Web page based on previous generic and query-based rankings. In summary, assuming a set of successive past top-k rankings, we study the evolution of Web pages in terms of ranking trend sequences used for Markov Models training, which are in turn used to predict future rankings. First, we quantify the ranking trends of Web pages through the rank change rate (*racer*) measure. The distinct *racer* values that appear in the trend sequences are reduced to a manageable size with regard to model extraction. The future rank of a Web page is predicted by matching its current *racer* sequences to the Markov Models paths. The prediction quality is quantified as the similarity between the predicted and the actual rankings and compared as well to alternative baseline predictors. We introduce a new similarity measure for comparing top-k ranked lists. In order to evaluate the effectiveness of $(PR)^2$ we performed extensive experiments on real world datasets both for global and query-based top-k rankings. The predictions are highly accurate for all experimental setups and similarity measures. This framework is thus a meaningful tool for Web page rank predictions.

Machine Translation

These works are follow-ups of works performed in a leading machine translation company, namely SYSTRAN, during a break year in 2003–2004.

Integration of a Legacy Machine Translation System into an Open Workflow [8]

A general, mature, rule-based MT system is bound to reach a saturation point because of the intrinsic complexity of the natural language description. For such systems, maintenance is a complex task and customization is expensive and time-consuming. Therefore, improving the system's interaction with the linguistic rules has proven to be more productive than endlessly adding new rules and exceptions to reach the theoretical accuracy level of 100%. We describe our strategy to "open up" such a system and provide practical details on how this was done on SYSTRAN's classical engines. This approach is the foundation of the SYSTRAN version 5 translation engines. We show the immediate benefits of the evolution of our architecture and the new extension potentiality.

Using XSLT to drive a Machine Translation Process [6]

XSL Transformation stylesheets are typically used to transform a document described in an XML formalism into another XML structure, to modify an XML document, or to publish content stored into an XML document to a publishing format (XSL-FO, (X)HTML...). SYSTRAN Translation Stylesheets (STSs) use XSLT to drive and control the machine translation of XML documents (native XML document formats or XML representations—such as XLIFF—of other kinds of document formats). STSs do not only provide a simple way to indicate which part of the document text is to be translated, but also enables the fine-tuning of translation, especially by using the structure of the document to help disambiguate natural language semantics and determine proper context. For instance, the phrase "Access from front door" is to be analyzed as "The access from front door" within a title, and as "Do access (something) from front door" in the text body. In that case, the STS would pass a title option to the translation engine. The stylesheet can activate specialized domain dictionaries for parts of the document and mark some expressions as not to be translated, in the same manner. Another key application of STS is to consider machine translation as part of the authoring and publishing process: source documents can be annotated with natural language markup produced by the author, markup which will be processed by STS to improve the quality of translation, the gateway to the automatic publishing of a multilingual Web site from a monolingual (annotated) source. This composition, inside the publishing process, is a real breakthrough for Web content translation.

Self References

Book Chapter

[1] Pierre Senellart and Vincent D. Blondel. Automatic discovery of similar words. In Michael W. Berry and Malu Castellanos, editors, *Survey of Text Mining: Clustering, Classification and Retrieval*. Springer-Verlag, January 2008. Second Edition.

Journal Article

[2] Vincent D. Blondel, Anahí Gajardo, Maureen Heymans, Pierre Senellart, and Paul Van Dooren. A measure of similarity between graph vertices: applications to synonym extraction and Web searching. *SIAM Review*, 46(4):647–666, 2004.

Conference Articles

[3] Yann Ollivier and Pierre Senellart. Finding related pages using Green measures: An illustration with Wikipedia. In *Proc. AAAI*, Vancouver, Canada, July 2007.

[4] Pierre Senellart and Serge Abiteboul. On the complexity of managing probabilistic XML data. In *Proc. PODS*, Beijing, China, June 2007.

[5] Serge Abiteboul and Pierre Senellart. Querying and updating probabilistic information in XML. In *Proc. EDBT*, Munich, Germany, March 2006.

[6] Pierre Senellart and Jean Senellart. SYSTRAN Translation Stylesheets: Machine translation driven by XSLT. In *Proc. XML Conference & Exposition*, Atlanta, USA, November 2005.

[7] François-Xavier Dudouet, Ioana Manolescu, Benjamin Nguyen, and Pierre Senellart. XML warehousing meets sociology. In *Proc. IADIS ICWI*, Lisbon, Portugal, October 2005.

[8] Mats Attnäs, Pierre Senellart, and Jean Senellart. Integration of SYSTRAN MT systems in an open workflow. In *Proc. MT Summit*, Phuket, Thailand, September 2005.

[9] Pierre Senellart. Identifying Websites with flow simulation. In *Proc. ICWE*, Sydney, Australia, July 2005.

Preprints

[10] Pierre Senellart and Georg Gottlob. On the complexity of deriving schema mappings from database instances. **Submitted for publication**.

[11] Pierre Senellart, Avin Mittal, Daniel Muschick, Rémi Gilleron and Marc Tommasi. Automatic wrapper induction from hidden-Web sources with domain knowledge. **Submitted for publication**.

[12] Michalis Vazirgiannis, Pierre Senellart, Dimitris Drosos, and Akrivi Vlachou. $(\text{PR})^2$: Training Markov models for Web page rank prediction. **Submitted for publication**.

External References

[AB86] Serge Abiteboul and Nicole Bidoit. Non first normal form relations: An algebra allowing data restructuring. *Journal of Computer and System Sciences*, 33(3):361–393, 1986.

[ACMM03] Luigi Arlotta, Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Automatic annotation of data extracted from large Web sites. In *Proc. WebDB*, San Diego, USA, June 2003.

[AGM03] Arvind Arasu and Hector Garcia-Molina. Extracting structured data from Web pages. In *Proc. SIGMOD*, San Diego, USA, June 2003.

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, USA, 1974.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, Reading, USA, 1995.

[AKG91] Serge Abiteboul, Paris Kanellakis, and Gösta Grahne. On the representation and querying of sets of possible worlds. *Theoretical Computer Science*, 78(1):158–187, 1991.

[AMP05] Serge Abiteboul, Ioana Manolescu, and Nicoleta Preda. Sharing content in structured P2P networks. In *Proc. BDA*, Saint-Malo, France, October 2005.

[ANR05] Serge Abiteboul, Benjamin Nguyen, and Gabriela Ruberg. Building an active content warehouse. In *Processing and Managing Complex Data for Decision Support*. Idea Group Publishing, 2005.

[ATT] ATT Research. Graphviz. `http://www.graphviz.org/`.

[AXY] AXYANA software. Qizx/open. `http://www.axyana.com/qizxopen/`.

[BC81] Philip A. Bernstein and Dah-Ming W. Chiu. Using semi-joins to solve relational queries. *Journal of the ACM*, 28(1):25–40, 1981.

[Ber03] Philip A. Bernstein. Applying model management to classical meta data. In *Proc. CIDR*, Asilomar, USA, January 2003.

[BF04] Luciano Barbosa and Juliana Freire. Siphoning hidden-Web data through keyword-based interfaces. In *Proc. Simpósio Brasileiro de Bancos de Dados*, Brasília, Brasil, October 2004.

[BF05] Luciano Barbosa and Juliana Freire. Searching for hidden-Web databases. In *Proc. WebDB*, Baltimore, USA, June 2005.

[BFM+81] Catriel Beeri, Ronald Fagin, David Maier, Alberto Mendelzon, Jeffrey Ullman, and Mihalis Yannakakis. Properties of acyclic database schemes. In *Proc. STOC*, Milwaukee, USA, May 1981.

[BGMP92] Daniel Barbará, Hector Garcia-Molina, and Daryl Porter. The management of probabilistic data. *IEEE Transactions on Knowledge and Data Engineering*, 4(5):487–502, 1992.

[BR87] Catriel Beeri and Raghu Ramakrishnan. On the power of magic. In *Proc. PODS*, San Diego, USA, March 1987.

[BR91] Catriel Beeri and Raghu Ramakrishnan. On the power of magic. *Journal of Logic Programming*, 10(3&4):255–300, January 1991.

[Bri00] BrightPlanet. The deep Web: Surfacing hidden value. White Paper, July 2000.

[CCZ07] Shui-Lung Chuang, Kevin Chen-Chuan Chang, and ChengXiang Zhai. Context-aware wrapping: Synchronized data extraction. In *Proc. VLDB*, Vienna, Austria, September 2007.

[CHL+04] Kevin Chen-Chuan Chang, Bin He, Chengkai Li, Mitesh Patel, and Zhen Zhang. Structured databases on the Web: Observations and implications. *SIGMOD Record*, 33(3):61–70, September 2004.

[CHZ05] Kevin Chen-Chuan Chang, Bin He, and Zhen Zhang. Toward large scale integration: Building a metaquerier over databases on the Web. In *Proc. CIDR*, Asilomar, USA, January 2005.

[CKGS06] Chia-Hui Chang, Mohammed Kayed, Mohem Ramzy Girgis, and Khaled F. Shaalan. A survey of Web information extraction systems. *IEEE Transactions on Knowledge and Data Engineering*, 18(10):1411–1428, October 2006.

[CLB04] James Caverlee, Ling Liu, and David Buttler. Probe, cluster, and discover: Focused extraction of qa-pagelets from the deep Web. In *Proc. ICDE*, Boston, USA, March 2004.

[CM77] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. STOC*, Boulder, USA, May 1977.

[CMM01] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large Web sites. In *Proc. VLDB*, Roma, Italy, September 2001.

[CP87] Roger Cavallo and Michael Pittarelli. The theory of probabilistic databases. In *Proc. VLDB*, Brighton, United Kingdom, September 1987.

[CvdBD99] Soumen Chakrabarti, Martin van den Berg, and Byron Dom. Focused crawling: A new approach to topic-specific Web resource discovery. *Computer Networks*, 31(11–16):1623–1640, 1999.

[DCL+00] Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C. Lee Giles, and Marco Gori. Focused crawling using context graphs. In *Proc. VLDB*, Cairo, Egypt, September 2000.

[DEW97] Robert B. Doorenbos, Oren Etzioni, and Daniel S. Weld. A scalable comparison-shopping agent for the World-Wide Web. In *Proc. Agents*, Marina del Ray, USA, February 1997.

[DG97] Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *Proc. PODS*, Tucson, USA, May 1997.

[DGH01] Alex Dekhtyar, Judy Goldsmith, and Sean R. Hawkes. Semistructured probabilistic databases. In *Proc. SSDBM*, Tokyo, Japan, July 2001.

[DGL00] Oliver M. Duschka, Michael R. Genesereth, and Alon Y. Levy. Recursive query plans for data integration. *Journal of Logic Programming*, 43(1):49–73, April 2000.

[Die05] Reinhard Diestel. *Graph Theory*. Springer, New York, USA, 2005.

[DP69] Robert A. Di Paola. The recursive unsolvability of the decision problem for the class of definite formulas. *Journal of the ACM*, 16(2):324–327, 1969.

[dR95] Michel de Rougemont. The reliability of queries. In *Proc. PODS*, San Jose, United States, 1995.

[DS04] Nilesh N. Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. In *Proc. VLDB*, Hong Kong, China, September 2004.

[DS07] Nilesh N. Dalvi and Dan Suciu. Management of probabilistic data: foundations and challenges. In *Proc. PODS*, Beijing, China, June 2007.

[eXi] eXist. `http://exist.sourceforge.net/`.

[Fag06] Ronald Fagin. Inverting schema mappings. In *Proc. PODS*, Chicago, USA, June 2006.

[FKMP03] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. In *Proc. ICDT*, Siena, Italy, January 2003.

[FKPT07] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Quasi-inverses of schema mapping. In *Proc. PODS*, Beijing, China, June 2007.

[FKTP04] Ronald Fagin, Phokion G. Kolaitis, Wang-Chiew Tan, and Lucian Popa. Composing schema mappings: Second-order dependencies to the rescue. In *Proc. PODS*, Paris, France, June 2004.

[FLN03] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66(4):614–656, June 2003.

[FR97] Norbert Fuhr and Thomas Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Transactions on Information Systems*, 15(1), 1997.

[GJ79] Michael R. Garey and David S. Johnson. *Computers And Intractability. A Guide to the Theory of NP-completeness*. W. H. Freeman, New York, USA, 1979.

[GKT07] Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *Proc. PODS*, Beijing, China, June 2007.

[GLS99] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. In *Proc. PODS*, Philadelphia, United States, May 1999.

[GMTT06] Rémi Gilleron, Patrick Marty, Marc Tommasi, and Fabien Torre. Interactive tuples extraction from semi-structured data. In *Proc. Web Intelligence*, Hong Kong, China, December 2006.

[Goo] Google search engine. `http://www.google.com/`.

[GT06] Todd J. Green and Val Tannen. Models for incomplete and probabilistic information. In *Proc. EDBT Workshops, IIDB*, Munich, Germany, March 2006.

[Hal01] Alon Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10:270–294, 2001.

[HC03] Bin He and Kevin Chen-Chuan Chang. Statistical schema matching across Web query interfaces. In *Proc. SIGMOD*, San Diego, USA, June 2003.

[HGS03] Edward Hung, Lise Getoor, and V. S. Subrahmanian. PXML: A probabilistic semistructured data model and algebra. In *Proc. ICDE*, Bangalore, India, March 2003.

[HMYW04] Hai He, Weiyi Meng, Clement Yu, and Zonghuan Wu. Automatic integration of Web search interfaces with WISE-integrator. *VLDB Journal*, 13(3):256–273, September 2004.

[HPZC07] Bin He, Mitesh Patel, Zhen Zhang, and Kevin Chen-Chuan Chang. Accessing the deep Web: A survey. *Communications of the ACM*, 50(2):94–101, May 2007.

[IET99] IETF. Request For Comments 2616, June 1999. `http://www.ietf.org/rfc/rfc2616.txt`.

[IG02] Panagiotis G. Ipeirotis and Luis Gravano. Distributed search over the hidden Web: Hierarchical database sampling and selection. In *Proc. VLDB*, Hong Kong, China, August 2002.

[IL84] Tomasz Imieliński and Witold Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, 1984.

[ISOY02] Lucian Ilie, Roberto Solis-Oba, and Sheng Yu. Reducing the size of NFAs by using equivalences and preorders. In *Proc. CPM*, Jeju Island, South Korea, June 2002.

[Jav] Java Compiler Compiler. `https://javacc.dev.java.net/`.

[JGTT06] Florent Jousse, Rémi Gilleron, Isabelle Tellier, and Marc Tommasi. Conditional Random Fields for XML trees. In *Proc. ECML Workshop on Mining and Learning in Graphs*, Berlin, Germany, September 2006.

[KLSS95] Thomas Kirk, Alon Y. Levy, Yehoshua Sagiv, and Divesh Srivastava. The information manifold. In *Proc. Information Gathering from Heterogeneous, Distributed Environments*, Stanford, USA, 1995.

[Knu97] Donald E. Knuth. *The Art of Computer Programming*, volume 1. Addison-Wesley, Boston, USA, third edition, 1997.

[Kol05] Phokion G. Kolaitis. Schema mappings, data exchange, and metadata management. In *Proc. PODS*, Baltimore, Maryland, June 2005.

[Kőn36] Dénes Kőnig. *Theorie der endlichen und unendlichen Graphen*. Akademische Verlagsgesellschaft, Leipzig, Germany, 1936.

[KS07] Benny Kimelfeld and Yehoshua Sagiv. Matching twigs in probabilistic XML. In *Proc. VLDB*, Vienna, Germany, September 2007.

[LD94] Nada Lavrač and Sašo Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, New York, USA, 1994.

[Len02] Maurizio Lenzerini. Data integration: a theoretical perspective. In *Proc. PODS*, Madison, USA, June 2002.

[Ley] Michael Ley. DBLP bibliography. `http://www.informatik.uni-trier.de/~ley/db/index.html`.

[LMP01] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional Random Fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. ICML*, Williamstown, USA, June 2001.

[LMSS95] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *Proc. PODS*, San Jose, USA, May 1995.

[LRO96] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. VLDB*, Bombay, India, September 1996.

[LSC06] Te Li, Qihong Shao, and Yi Chen. PEPX: a query-friendly probabilistic XML database. In *Proc. CIKM*, Arlington, USA, November 2006. Poster.

[LV97] Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, New York, USA, second edition, 1997.

[MHC+06] Jayant Madhavan, Alon Y. Halevy, Shirley Cohen, Xin Dong, Shawn R. Jeffery, David Ko, and Cong Yu. Structured data meets the Web: A few observations. *IEEE Data Engineering Bulletin*, 29(4):19–26, December 2006.

[Mit07] Avin Mittal. Probing the hidden Web. Research internship report. Technical Report 479, Gemo, INRIA Futurs, July 2007.

[Mus07] Daniel Muschick. Unsupervised learning of XML tree annotations. Master's thesis, Université de Technologie de Lille and Technische Universität Graz, June 2007.

[NJ02] Andrew Nierman and H. V. Jagadish. ProTDB: Probabilistic data in XML. In *Proc. VLDB*, Hong Kong, China, August 2002.

[Ott48] Richard Otter. The number of trees. *Annals of Mathematics*, 49(3):583–599, July 1948.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley Pub. Co., Reading, USA, 1994.

[PH01] Rachel Pottinger and Alon Y. Halevy. Minicon: a scalable algorithm for answering queries using views. *VLDB Journal*, 10(2):182–198, September 2001.

[PL00] Rachel Pottinger and Alon Y. Levy. A scalable algorithm for answering queries using views. In *Proc. VLDB*, Cairo, Egypt, September 2000.

[Por80] Martin F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, July 1980.

[Pri] Princeton University Cognitive Science Laboratory. WordNet. `http://wordnet.princeton.edu/`.

[RB01] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[RGM01] Sriram Raghavan and Hector Garcia-Molina. Crawling the hidden Web. In *Proc. VLDB*, Roma, Italy, September 2001.

[RH05] Yanbo Ru and Ellis Horowitz. Indexing the invisible web: a survey. *Online Information Review*, 29(3):249–265, 2005.

[RSU95] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering queries using templates with binding patterns. In *Proc. Principles of Database Systems*, San Jose, USA, May 1995.

[Sch80] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980.

[SM73] Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time. In *Proc. STOC*, Austin, USA, May 1973.

[SM07] Charles Sutton and Andrew McCallum. An introduction to conditional random fields for relational learning. In Lise Getoor and Ben Taskar, editors, *Introduction to Statistical Relational Learning*. MIT Press, Cambridge, USA, November 2007.

[SO06] Andrew Smith and Miles Osborne. Using gazetteers in discriminative information extraction. In *Proc. CoNLL*, New York, USA, June 2006.

[SU02] Marcus Schaefer and Christopher Umans. Completeness in the polynomial hierarchy, a compendium. *SIGACT News*, 33(3):32–49, September 2002.

[Tra63] Boris A. Trakhtenbrot. Impossibility of an algorithm for the decision problem in finite classes. *American Mathematical Society Translations Series 2*, 23:1–5, 1963.

[TY84] Robert E. Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13(3):566–579, 1984.

[Uni06] Unicode Consortium. *The Unicode Standard, Version 5.0*. Addison-Wesley, Reading, USA, November 2006.

[vKdKA05] Maurice van Keulen, Ander de Keijzer, and Wouter Alink. A probabilistic XML approach to data integration. In *Proc. ICDE*, April 2005.

[W3Ca] W3C. Web services activity. `http://www.w3.org/2002/ws/`.

[W3Cb] W3C. XQuery update facility. `http://www.w3.org/TR/xqupdate/`.

[W3C99] W3C. HTML 4.01 specification, September 1999. `http://www.w3.org/TR/REC-html40/`.

[W3C01] W3C. Web Services Description Language (WSDL) 1.1, March 2001. `http://www.w3.org/TR/wsdl`.

[W3C04a] W3C. RDF vocabulary description language 1.0: RDF schema. `http://www.w3.org/TR/rdf-schema/`, 2004.

[W3C04b] W3C. Web Ontology Language (OWL). `http://www.w3.org/2004/OWL/`, 2004.

[W3C07] W3C. XQuery 1.0: An XML query language. `http://www.w3.org/TR/xquery/`, January 2007.

[WDY06] Wensheng Wu, AnHai Doan, and Clement T. Yu. WebIQ: Learning from the Web to match deep-Web query interfaces. In *Proc. ICDE*, Atlanta, USA, April 2006.

[WDYM05] Wensheng Wu, AnHai Doan, Clement T. Yu, and Weiyi Meng. Bootstrapping domain ontology for semantic Web services from source Web sites. In *Proc. Technologies for E-Services*, Trondheim, Norway, September 2005.

[Wid05] Jennifer Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *Proc. CIDR*, Pacific Grove, USA, January 2005.

[Wra76] Celia Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):23–33, 1976.

[WYDM04] Wensheng Wu, Clement Yu, AnHai Doan, and Weiyi Meng. An interactive clustering-based approach to integrating source query interfaces on the deep Web. In *Proc. SIGMOD*, Paris, France, June 2004.

[XML00] XML::DB Initiative. Xupdate. `http://xmldb-org.sourceforge.net/xupdate/`, September 2000.

[Yan81] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proc. VLDB*, Cannes, France, September 1981.

[YHY07] Xiaoxin Yin, Jiawei Han, and Philip S. Yu. Truth discovery with multiple conflicting information providers on the web. In *Proc. KDD*, San Jose, USA, August 2007.

[ZHC04] Zhen Zhang, Bin He, and Kevin Chen-Chuan Chang. Understanding Web query interfaces: best-effort parsing with hidden syntax. In *Proc. SIGMOD*, Paris, France, June 2004.

[ZHC05] Zhen Zhang, Bin He, and Kevin Chen-Chuan Chang. Light-weight domain-based form assistant: Querying Web databases on the fly. In *Proc. VLDB*, Trondheim, Norway, September 2005.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Proc. ISSAC*, Marseille, France, 1979.