Effective Information Integration and Reutilization: Solutions to Technological Deficiency and Legal Uncertainty

by
Hongwei Zhu

B.S. Thermal Engineering, B.S. Environmental Engineering
Tsinghua University, Beijing, China, 1991

M.S. Environment Engineering
Tsinghua University, Beijing, China, 1994

M.S. Environment Engineering
University of Cincinnati, Cincinnati, OH, USA, 1998

S.M. Technology and Policy
Massachusetts Institute of Technology, Cambridge, MA, USA, 2002

Submitted to the Engineering Systems Division
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Technology, Management and Policy
at the
Massachusetts Institute of Technology
September 2005

Signature of Author ....
Engineering Systems Division
September 12, 2005

Certified by ....
Stuart E. Madnick, Thesis Supervisor
John Norris Maguire Professor of Information Technology and Professor of Engineering Systems

Certified by ....
Nazli Choucri
Professor of Political Science

Certified by ....
Michael D. Siegel
Principal Research Scientist

Certified by ....
Frank Manola
Senior Principal Software Systems Engineer

Certified by ....
Frank R. Field, III
Senior Research Engineer

Accepted by ....
Richard de Neufville
Professor of Engineering Systems
Chair, ESD Education Policy Committee

Effective Information Integration and Reutilization: Solutions to Technological Deficiency and Legal Uncertainty

by

Hongwei Zhu

Submitted to the Engineering Systems Division
on September 12, 2005 in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy in
Technology, Management and Policy

Abstract

The amount of electronically accessible information has been growing exponentially. How to effectively use this information has become a significant challenge. A post 9/11 study indicated that the *deficiency of semantic interoperability technology* hindered the ability to integrate information from disparate sources in a meaningful and timely fashion to allow for preventive precautions. Meanwhile, organizations that provided useful services by combining and reusing information from publicly accessible sources have been legally challenged. The Database Directive has been introduced in the European Union and six legislative proposals have been made in the U.S. to provide legal protection for non-copyrightable database contents, but the Directive and the proposals have differing and sometimes conflicting scope and strength, which creates *legal uncertainty for valued-added data reuse practices*. The need for clearer data reuse policy will become more acute as information integration technology improves to make integration much easier.

This Thesis takes an interdisciplinary approach to addressing both the technology and the policy challenges, identified above, in the effective use and reuse of information from disparate sources.

The technology component builds upon the existing Context Interchange (COIN) framework for large-scale semantic interoperability. We focus on the problem of temporal semantic heterogeneity where data sources and receivers make time-varying assumptions about data semantics. A collection of time-varying assumptions are called a *temporal context*. We extend the existing COIN representation formalism to explicitly represent temporal contexts, and the COIN reasoning mechanism to reconcile temporal semantic heterogeneity in the presence of semantic heterogeneity of time. We also perform a systematic and analytic evaluation of the flexibility and scalability of the COIN approach. Compared with several traditional approaches, the COIN approach has much greater flexibility and scalability.

For the policy component, we develop an economic model that formalizes the policy instruments in one of the latest legislative proposals in the U.S. The model allows us to identify the circumstances under which legal protection for non-copyrightable content is needed, the different conditions, and the corresponding policy choices. Our analysis indicates that depending on the cost level of database creation, the degree of differentiation of the reuser database, and the efficiency of policy administration, the optimal policy choice can be protecting a legal monopoly, encouraging competition via compulsory licensing, discouraging voluntary licensing, or even allowing free riding. The results provide useful insights for the formulation of a socially beneficial database protection policy.

Supervisor: Stuart E. Madnick

Title: John Norris Maguire Professor of Information Technology and
Professor of Engineering Systems

Acknowledgements

I am very grateful to Professor Stuart Madnick for his mentorship, encouragement, and guidance. Without his support and constant care, I would not have been able to complete this Thesis. He is so generous in making him available for discussions on research, and on career development in general. Working with Professor Madnick in the past five years has had profound impact on my life – I have learned the importance of doing good work, and at the same time, being a kind person.

I was fortunate to have the support of a relatively large committee. I am thankful to the help and advice of all committee members. Numerous discussions with Frank Manola have helped me to formulate the solutions and articulate them. Frank has also given me extensive comments on how to improve the Thesis, and on how to write technical papers in general. Professor Nazli Choucri has inspired me to look into the issues in geo-political domain and taught me how to approach these issues using appropriate tools. Over the course of my Thesis, I benefited from useful discussions with Dr. Michael Siegel on how to have both rigor and relevance in research. Comments from Dr. Frank Field and Professor Richard de Neufville have helped me define the scope and improve the formulation of the policy component of the Thesis.

Over the years, I also benefited from working with many people in the research group. I am thankful to Aykut Firat, who brought the legacy COIN prototype to life. His articulation on many aspects of the COIN framework helped me tremendously in formulating my research questions. The improvement to the prototype system by Philip Lee, Tarik Alatovic, and many others, has made it easy for me when I needed to put together demo applications. Numerous discussions with Allen Moulton have been helpful in narrowing down research questions. I also benefited from the discussions with Yu (Jeffrey) Hu, Xiaoquan (Michael) Zhang, Marshall van Alstyne, and Chander Velu on economics and with my office mate Sumit Bhansali on broad issues. Advice on career development from Tom Lee, Rich Wang, and Yang Lee is gratefully acknowledged.

I would like to thank Yubettys Baez and Eda Daniel, who make things happen and keep them in order. Thanks also go to numerous friends who have helped me in the past.

I am indebted to my family. To support my study, my wife changed her job during the early stage of her academic career. Her support and love have helped me to go through the toughest times of the journey. My daughter, now two years old, has given me new meaning of life. I am very thankful to my mother-in-law, who came here two years ago when everybody was afraid of traveling during the SARS outbreak, for her dedication in helping us raise our daughter. I am very grateful that my parents are in great health and that they finally obtained visas, after trying numerous times since 1997, to come to visit us this summer. I also thank my brother for his care and encouragement.

Table of Contents

Chapter 1

Introduction

This Thesis concerns the effective use and reuse of information in autonomous, heterogeneous, and evolving sources. We take an interdisciplinary approach to develop the semantic interoperability *technology* that enables information integration and analyze the public *policy* issues that arise when such technology is used for systematic reuse of information from sources created by different organizations. This chapter provides the motivation behind this study, highlights the major contributions, and outlines the structure of this Thesis.

1.1 Challenges and Objectives

This Thesis is motivated by the challenges depicted in Figure 1.1. We will explain these challenges in detail and present the specific objectives of this Thesis.

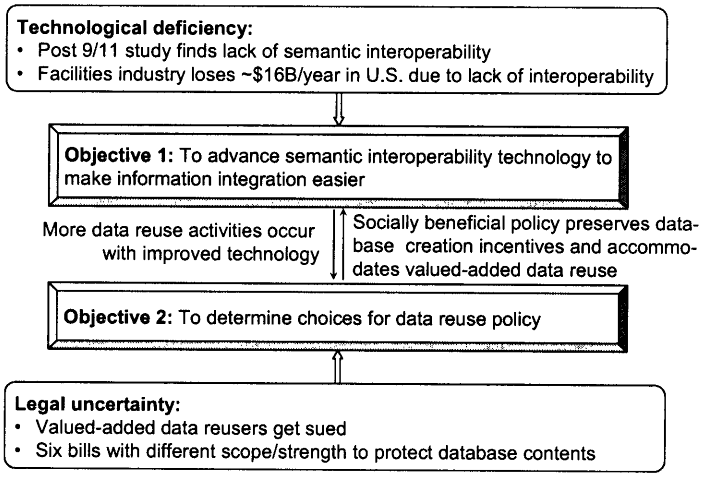

Technological deficiency:
• Post 9/11 study finds lack of semantic interoperability
• Facilities industry loses ~$16B/year in U.S. due to lack of interoperability
Objective 1: To advance semantic interoperability technology to make information integration easier
More data reuse activities occur with improved technology
Socially beneficial policy preserves database creation incentives and accommodates valued-added data reuse
Objective 2: To determine choices for data reuse policy
Legal uncertainty:
• Valued-added data reusers get sued
• Six bills with different scope/strength to protect database contents

Figure 1.1 Challenges and objectives

In the aftermath of the 9/11 tragedy, a study by the National Research Council (2002) found that "although there are many private and public databases that contain information potentially relevant to counter terrorism programs, they lack the necessary context definitions (i.e., metadata) and access tools to enable interoperation with other databases and the extraction of meaningful and timely information". A congressional investigation[1] on 9/11 indicated that had various businesses and government agencies been able to exchange and integrate information in a meaningful and timely fashion, the tragedy could perhaps have been prevented. This is a typical *semantic heterogeneity* problem that is often encountered in information integration, which we explain below.

When people design information systems and create databases, they typically make numerous assumptions about various aspects of the meaning of the data, such as units of measure (e.g., *profit* in Deutschmarks or Euros), scale factors (e.g., *profit* in thousands or millions), and definitions of data elements (e.g., whether the *profit* is gross profit including taxes or net profit excluding taxes). The *semantic heterogeneity* problem arises when different systems and the users of the systems have different assumptions about what appear to be similar data. Furthermore, the assumptions can change over times (e.g., currency can change from Deutschmarks to Euros), causing what appear to be similar data in different time periods to have different interpretations. We call this phenomenon *time-varying semantics* and the heterogeneity problem in the presence of time-varying semantics *temporal semantic heterogeneity*. The differences in those assumptions need to be reconciled, automatically or semi-automatically by integration software systems, when a user needs to integrate data from these systems; in other words, we need to achieve *semantic interoperability* amongst heterogeneous systems and system users.

Semantic interoperability is important for government agencies, businesses, and even individuals who want to effectively use the increasing amount of information available in an ever growing number of electronically accessible data sources. It is also a difficult challenge for the following reasons:

- The assumptions in various information systems and by diverse system users are often not explicitly recorded, making it impossible for a software system to account for the differences of the assumptions. We call a set of assumptions by a system or a user a *context*. Thus, different systems and system users are often in different contexts.
- Even if the assumptions are explicitly recorded, they may be in a format not accessible to the software that performs information integration. For example, the assumptions may be recorded in word processing files as system documentation that is meant for human consumption.
- Even if the assumptions are accessible to the software, e.g., if units of measure are made part of the data or are recorded as *metadata* (i.e., structured and systematic description of the data, often managed by metadata software), the integration software won't necessarily know what to do with mismatched metadata, e.g., units of measure. That is, metadata in different sources still need to be interpreted and instructions on how to deal with mismatches need to be provided.

To summarize, the assumptions (i.e., the contexts) of various systems and system users may be *unavailable*, *inaccessible*, or *unusable* for the integration software to reconcile any context

[1] "Joint Inquiry into Intelligence Community Activities before and after the Terrorist Attacks of September 11, 2001", by The House Permanent Select Committee On Intelligence And The Senate Select Committee on Intelligence, December 2002.

difference. These distinctions roughly correspond to the continuum of semantics in Uschold (2003)[2]. In the rest of the Thesis, we will not make these distinctions and simply call the assumptions *implicit assumptions*. These assumptions need to be captured explicitly and the differences should be reconciled systematically to allow for meaningful and timely integration for information from diverse sources. Technologies deployed today lack this capability.

A significant amount of effort has been put into addressing this challenge. Unfortunately, most existing approaches, which we will discuss in Chapter 5, are inefficient and very costly to implement and maintain. As a result, despite the fact that a typical large organization spends nearly 30% of its IT budget on integration and interoperation related efforts[3], many inter- and intra- organizational systems still have poor interoperability. We learned from the 9/11 tragedy that serious consequences can result from this *technological deficiency*. As another example, the National Institute of Standards and Technology (NIST) estimates that the lack of interoperability costs the U.S. capital facilities industry[4] alone $15.8 billion per year, representing 1-2% of industry revenue; this estimate is likely to be only a portion of the total cost of inadequate interoperability (Gallaher et al., 2004).

There have been various technical attempts to address different aspects of the challenge, and much work continues (Lenzerini, 2001; Abiteboul et al., 2005). In this Thesis, we aim to contribute to the technology for enabling semantic interoperability. The specific objective is:

> **Objective 1.** To develop a representation formalism and a reasoning mechanism for representing and reconciling temporal semantic heterogeneity.

The ease of accessing data provided by the Web has made possible certain new business practices that reuse and combine data from various sources. This often requires using existing semantic interoperability technology, even though it is in a primitive state. While most of the data reuse practices are value creating activities, they sometimes can run afoul of the interests of the initial database creators, who wish to have legal protection of database contents when the access to their databases cannot be restricted via other means (e.g., data in web pages that are meant to be publicly accessible). This has raised policy concerns about the rights a database creator (data source owner) should have over how that data is reused, and how any policy on this subject, if needed, should be formulated. The European Union first introduced the Database Directive[5] in 1996 to require its member states to provide legal protection to database contents beginning January 1, 1998. In the U.S., six legislative proposals, discussed in Chapter 6, have been introduced in Congress but none of them has been passed into law. These proposals offer protection with differing and sometimes conflicting strength and scope, which creates *legal uncertainty* for value-added data reuse practices. The need for clearer policy will become more acute as semantic interoperability

[2] Semantics can be implicit, explicit but informal, formal but for human processing, and formal for machine processing.

[3] See "Reducing Integration's Cost", Forrester Research, December 2001.

[4] The capital facilities industry, a component of the entire U.S. construction industry, encompasses the design, construction, and maintenance of large commercial, institutional, and industrial buildings, facilities, and plants. In 2002, the nation set in place $374 billion in new construction on capital facilities

[5] "Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases", a copy of the Directive can be found at http://europa.eu.int/ISPO/infosoc/legreg/docs/969ec.html

technology (the subject of Objective 1 above) improves to make data integration and reutilization much easier. Hence, this Thesis also investigates the policy issues of data reutilization, which leads to the following objective:

> **Objective 2.** To analyze the policy implications when semantic interoperability technology is applied for data reuse, identify the conditions under which legal protection for database contents may be needed, and determine socially beneficial policy choices that preserve database creation incentives and accommodate value-added data reuse.

1.2 Summary of Contributions

How to effectively use the increasing amount of electronically accessible information is one of the biggest challenges faced by many organizations and individuals. We make unique contributions to addressing this challenge by developing the enabling technology as well as considering the societal implications of the technology. The interdisciplinary approach allows the outcome of the Thesis to be of theoretical significance as well as practical impact.

The technology component of the Thesis is developed as extensions to the existing Context Interchange (COIN) framework (Goh, 1997; Goh et al., 1999; Firat, 2003), which offers a formalism for explicit representation of the different assumptions made by systems and system users and for declarative descriptions on how differences, if any, can be resolved. COIN also has a reasoning mechanism, implemented in a software component called the *mediator*, which automatically compares contexts and generates instructions in a mediated query to reconcile context differences.

Our extensions deal with the problem of temporal semantic heterogeneity caused by time-varying semantics. This is different from current schema versioning and schema evolution research, which focuses on the management of structural changes in data sources, i.e., it only concerns that data is not lost and old data remains accessible when the schema is changed (Roddick, 1995).

In describing the contributions of the present work, certain concepts, not yet explained but covered in the rest of the Thesis, will be used.

Formalism for Representing Time-Varying Semantics

We introduce the concept of *temporal context* and develop a formalism for representing time-varying semantics. The existing COIN framework uses a shared *ontology* and a set of special attributes called *modifiers* to encode semantic assumptions in data *sources* and *receivers*[6]. A (static) *context* is described using a single *rule* that assigns a value for each modifier in the shared ontology. In a temporal context, at least one modifier changes its value over time. That is, modifiers in a temporal context can be multi-valued. In this case, we use multiple rules for value assignment, with each rule applicable over a time period. Intuitively, this is equivalent to using time-stamped, multi-valued modifiers to describe temporal contexts.

[6] Any system that provides data upon request is a data source; any system (including a user or a user application) that makes such requests is a data receiver. These are the roles a system can play in information exchange.

Extensions to COIN Reasoning Mechanism

We extend the COIN reasoning mechanism to process temporal contexts and reconcile temporal semantic heterogeneity. With temporal context, the semantic differences between the sources and the receiver become a multi-set, i.e., there are different sets of semantic differences in different time periods. They are identified by comparing the modifier assignments in the source contexts and the receiver context. This involves temporal reasoning because the applicability of the modifier assignment rules is time dependent. We translate the temporal reasoning problem into a temporal constraint solving problem, which is solved by constraint solvers implemented using the declarative language Constraint Handling Rules (CHR) (Frühwirth, 1998). The effect of temporal consistency is to generate vertical partitions for the sources so that each partition corresponds to a static context. That is, this process reduces the temporal context problem into multiple problems with static contexts.

The temporal constraint solver in this Thesis is in fact a form of semantic query optimization; by removing unsatisfiable temporal constraints in the mediated query, we eliminate unnecessary data fetching in distributed sources. In addition, we also develop a method of treating certain source capability restrictions as constraints, which are solved using CHR so that certain kinds of capability heterogeneity can be reconciled during mediation.

Systematic Evaluation of the Flexibility and Scalability of the COIN Approach

We provide a systematic evaluation of the technology in terms of adaptability, extensibility, and scalability. Adaptability refers to the capability of adapting to changes in sources and receivers; extensibility is the capability of accommodating new sources and receivers with minimal effort. These two properties are collectively called flexibility. Scalability measures the amount of effort involved in developing data transformations between a given number of data sources and receivers. Prior to this work, the lack of such systematic analysis has led to certain misunderstandings of the COIN approach (Kashyap and Sheth, 2000). By comparing with commonly practiced traditional approaches, we show that COIN is flexible because of the declarative approach it takes. It is also scalable because it can compose all necessary conversions using a small set of component conversions.

Socially Beneficial Policy Choices for Database Protection

We develop an economic model to formally analyze public policy choices for the legal protection of database contents. This is still an unsettled and widely debated issue in the U.S. and worldwide. Similar to the existing Intellectual Property (IP) laws, a database protection law needs to balance between protecting the incentives of creating the databases and providing opportunities of value creation through reuse. This objective is very difficult to achieve because there is nothing like the *idea-expression* dichotomy in copyright law to guide the determination of what kinds of reuse should be allowed or disallowed. As discussed in Heald (2001), the *extraction-duplication* dichotomy does not exist in data reuse; instead, it is a continuum. The model developed in this Thesis allows us to identify social welfare enhancing policy choices under different conditions depending on the location on the continuum and other factors, e.g., cost of database creation and transaction cost of licenses. The results of this formal analysis allow us to derive a set of guidelines for formulating the database policy.

Successful Demonstration of the Value of Interdisciplinary Approach

Lastly, we argue that the interdisciplinary approach to semantic interoperability constitutes a unique contribution. A new technology often brings about new societal issues. For example, the earliest significant data reuse case was in 1918 between International News Service (INS) and Associated Press (AP), where INS lifted news stories from AP bulletins and wired them to its member newspapers on the west coast. This form of data reuse was enabled by telegraphic technology invented in the mid- to late-1800s. Many recent data reuse cases involve aggregators enabled by the Web and information integration technologies (Madnick and Siegel, 2002; Zhu et al., 2002a). Therefore, it is important that we understand the societal implications of the technologies we develop and advance. By studying the technology and policy aspects of information integration conjunctively, this Thesis generates interesting and timely results useful for improving the effective use of information in our society. In addition, the thesis also demonstrates the value of taking an interdisciplinary approach to technology research because technologists tend not to spend enough time to study the societal implications of technology[7], and the societal implications often have significant impact on the effectiveness of technology.

1.3 Thesis Outline

The Thesis is organized into seven chapters. There is no centralized literature review in the Thesis. Instead, we cite the literature that is most relevant to the particular issues addressed in each chapter.

Chapter 2 identifies and illustrates the kinds of temporal semantic heterogeneity. In addition to time-varying data semantics, the data models for time are tremendously heterogeneous across systems, e.g., different semantic assumptions, representations, and supported operations for temporal entities. We identify that little work has been done to represent and reason about time-varying semantics in the presence of heterogeneous time models. The technology component of this Thesis is to fill this gap.

Chapter 3 uses examples to illustrate how the COIN system with the extensions developed in this Thesis can be used to reconcile temporal semantic heterogeneities. We demonstrate the features from the perspectives of the users and system architecture. The COIN mediation service allows users to query data sources in diverse contexts as if they all were in the user's own context, therefore, users are alleviated from the burden of keeping track of contexts, the evolvement of contexts, and the data conversions from other contexts to the user context. This service reconciles semantic heterogeneity and at the same time supports heterogeneous user needs.

[7] For example, when discussing Project Oxygen, a massive effort on pervasive and human-centered computing at MIT's Computer Science and Artificial Intelligence Laboratory, co-director Professor Victor Zue comments "[W]e engineers think of wonderful ideas, but often they have unanticipated consequences" and points out the need for more study on the societal implications of the project, and pervasive computing in general. Details of his comments can be found at http://web.mit.edu/giving/spectrum/winter05/computer_talk.html.

Chapter 4 provides the technical details behind the features illustrated in Chapter 3. We start with a formal description of COIN, followed by the extensions for temporal semantic heterogeneity. Previously, data transformations are introduced declaratively during mediation and their execution is deferred to the query execution stage. When relations over temporal entities exist in the query, it becomes beneficial to execute the transformations for ground temporal values during mediation to perform semantic query optimization. We will discuss the tradeoffs between the two strategies. In addition, the roles of ontology and contexts in COIN and their relationship will be discussed.

Chapter 5 presents the metrics for evaluating information integration technologies. The metrics include adaptability, extensibility, and scalability. We then use the metrics to evaluate COIN against other commonly practiced approaches.

Chapter 6 focuses on the analysis of data reuse policy. After an overview of the history of database legislation, we present an extended spatial competition model to formalize the extraction-duplication continuum and the provisions proposed in a recently failed bill in the U.S. In addition, we also consider transaction cost and show how it affects the licensing provision. Based on the results of this formal analysis, we derive a set of guidelines for making policy choices that maximally allow value added data reuse without eliminating the incentives of creating databases.

Chapter 7 provides a brief summary and points out several areas for future investigation.

"A problem well-defined is half solved"
– John Dewey

Chapter 2

Temporal Semantic Heterogeneity

From Chapter 1 we know that data semantics is dependent on the assumptions made about the data. Semantic heterogeneity exists when different systems make different assumptions about similar data, and temporal semantic heterogeneity exists when the assumptions change over time. In this chapter, we develop a deeper understanding about the problem of temporal semantic heterogeneity by using examples and reviewing relevant literature. The issues identified in this chapter will be addressed by the technology component of this Thesis.

This chapter is organized as follows. In Section 2.1, we present a categorization of types of semantic heterogeneity that encompasses temporal semantic heterogeneity. In Section 2.2, we illustrate different aspects of temporal semantic heterogeneity using examples. In Section 2.3 we survey the existing approaches to temporal information management and identify the gap that this work is intended to fill. We summarize in Section 2.4.

Throughout the Thesis, we use *data* and *information* interchangeably. Any system that can provide data upon request is called a data *source*; any system (including a user or a user application) that makes such requests is a data *receiver*. Therefore, a system can be a source and a receiver, depending on whether it is requested to send data (source) or it requests to receive data (receiver). We also use *receiver* and *user* interchangeably.

2.1 Categorization of Semantic Heterogeneity

Various types of semantic heterogeneity have been documented in the literature (Goh, 1997; Firat, 2003; Kashyap and Sheth, 1996; Ram and Park, 2004; Naiman and Ouskel, 1995), though there is little consensus on how to categorize them. For example, Goh (1997) distinguishes schematic heterogeneity from semantic heterogeneity, and characterizes the distinction between them by the differences in the structure (i.e. the logical organization of data) versus the differences in the interpretation (i.e., the meaning of data); whereas in the classification of Ram and Park (2004), semantic heterogeneity exists in the data level as well in the schema level; while in Kashyap and

Sheth (1996), schematic heterogeneity includes the differences in data interpretation due to the differences in domain definition of a schema, e.g., different units of measure.

We observe that the logical organization of data (i.e., schema), and sometimes even the syntax used for representing the data, can convey commonly understood assumptions for interpreting the meaning of the data. So, it is difficult to have a crisp categorization that separate out semantics from everything else. For explication purposes, though, we still offer a categorization below.

Recall from Chapter 1 that semantic heterogeneity is usually caused by the different assumptions made by systems and system users. The assumptions (and hence the resulting heterogeneity) can be categorized into two types: *representational* and *ontological*; furthermore, depending on whether time is of concern, each type can be *atemporal*[8] or *temporal*. This categorization is illustrated in Figure 2.1 and explained below.

	Atemporal	**Temporal**
Representational	Profit is in DEM *v.* Profit is in USD	Profit is in DEM *until 1998* and in EUR *since 1999 v.* Profit is *always* in USD
Ontological	Profit is gross with taxes included *v.* Profit is net with taxes excluded	Profit is gross with taxes included *until 1998* and net with taxes excluded *since 1999 v.* Profit is *always* net with taxes excluded

Figure 2.1 Categorization of semantic heterogeneity.

Representational heterogeneity exists when different systems make different assumptions about the representation of the same thing, such as using different units of measure, scale factors (e.g., in thousands or millions), or syntax (e.g., the different orders of year, month, day appearing in a date). In the (atemporal) example in Figure 2.1, the *profit of a company* can be represented in DEM (i.e., Deutschmarks) in one system or in USD (i.e., U.S. dollars) in another, where the currency used is the assumption.

Ontology is an established branch in philosophy concerned with the nature and the relations of being. Bunge (1977) and Bunge (1979) provide a formal philosophical treatment on the topic. More recently, ontology has been widely studied in computer science as a formal and structured way of specifying a conceptualization of a set of related things. Intuitively, it offers a vocabulary for referring to things in the real world. An in-depth discussion of ontology is out of the scope of this Thesis; interested readers are referred to Wand and Webber (1988), Gruber (1993), Lee (1996), Guarino (1997), and Wand et al. (1999).[9] The intuitive understanding of ontology suffices for the purpose of discussing our categorization of semantic heterogeneity.

Ontological heterogeneity exists when different systems make different assumptions in terms of the definitions for what appears to be the same thing. In the (atemporal) example of Figure 2.1, the *profit* of a company can be *gross profit* including the taxes in one system and *net profit* excluding the

[8] Means "independent of time, timeless" (http://education.yahoo.com/reference/dictionary/entry/atemporal). It has been used in temporal database literature to refer to snap-shot data models.

[9] Wand and Webber (1988) draw the concepts from Bunge's works on ontology to propose a set of ontological principles useful in the design of information systems; Wand et al. (1999) illustrate the use the principles in entity-relation analysis. Lee (1996) seeks to establish a philosophical foundation for semantic interoperability by drawing Bunge's works on ontology as well as on semantics (Bunge, 1974a; Bunge, 1974b). Gruber (1993) and Guarino (1997) use ontology for purposes of knowledge representation and sharing.

taxes in another. In this case, the assumption is which definition of profit is meant by the term *profit* in each system.

Loosely speaking, representational heterogeneity concerns the different representations of the same thing, while ontological heterogeneity concerns the different meanings denoted by the same term.

Both the representational and the ontological assumptions can be static and do not change over time within an interested time period, in which case time is not of concern. The resulting heterogeneity is *atemporal*. Conversely, the assumptions can change over time, and the resulting heterogeneity is *temporal*. Thus, we can roughly distinguish four types of semantic heterogeneity:

- atemporal representational;
- atemporal ontological;
- temporal representational; and
- temporal ontological.

A data element can have a certain combination of these four basic types of semantic heterogeneity. For example, the profit data in a source can have time-varying definitions (temporal ontological), yet use the same currency through out (atemporal representational). We also give another example in Example 2.3 in Section 2.2.1.

The examples of temporal semantic heterogeneity in Figure 2.1 are extended from the atemporal examples by having the assumption in one system change over time. With temporal semantic heterogeneity, interpretations of data can differ not only between systems, but also between different time periods in a single system.

The following section focuses on temporal semantic heterogeneity. We will use examples to illustrate various aspects that need to be considered when reconciling temporal semantic heterogeneity.

2.2 Aspects of Temporal Semantic Heterogeneity

In addition to time-varying semantics, we identify two other aspects that are related to temporal semantic heterogeneity. These aspects, as summarized below, will be each discussed in the next three sub-sections:

time-varying semantics: we use several examples to illustrate temporal representational heterogeneity and temporal ontological heterogeneity

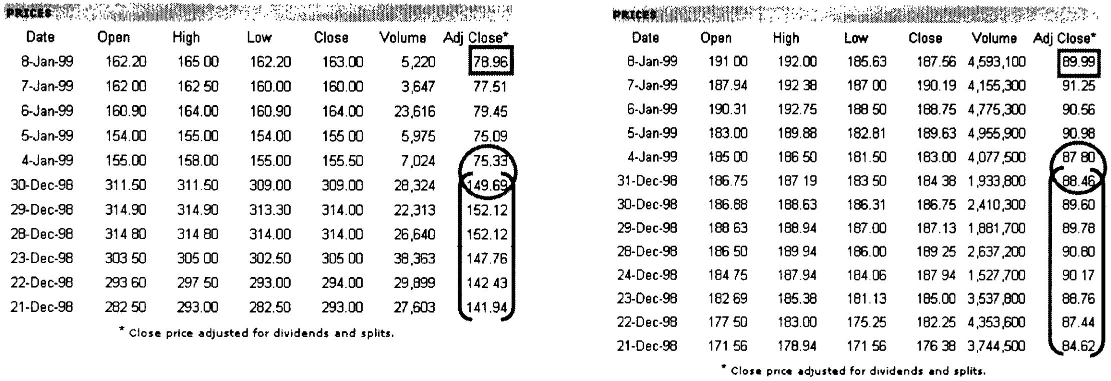
different time models: just as with any other type of data, time itself can be represented and interpreted differently in different systems. These differences correspond to representational and ontological heterogeneity of time. In addition, systems can differ in their capabilities of executing certain operations on time; we call these difference capability heterogeneity.

different associations between time and other types of data: we use examples from the literature to explain that there can be different interpretations of the association between time and other types of data.

2.2.1 Time-varying Semantics

When the implicit assumptions change over time, data corresponding to different time periods are subject to different interpretations. The following examples illustrate temporal semantic heterogeneity caused by time-varying implicit assumptions.

Example 2.1 (*Temporal representational heterogeneity*) Stocks of some companies are traded at multiple stock exchanges (i.e., markets) around the world. The prices of the same stock can be different in different markets, creating arbitraging opportunities (i.e., someone can profit from the price differences by buying shares in one market and selling them in another market at a higher price). There are usually some price differences, but not substantial differences. How big can price differences be in different markets? Are prices in multiple markets converging? These are the kinds of questions often asked by traders, financial analysts, hedge fund managers, investors, regulators, as well as financial economists. The availability of online financial databases provides the convenience for finding answers to such questions. In addition to subscription-based databases such as Datastream (from Thomson Financial at www.datastream.com), there are also sources such as Yahoo! Finance (at finance.yahoo.com) that provide free access to certain financial data. Figure 2.2 shows an excerpt of historical stock prices for IBM at Frankfurt (left) and New York (right) Stock Exchanges, reported by Yahoo! Finance. The adjusted close prices are listed in the last column (labeled Adj Close*) of each table.



Frankfurt, Germany

PRICES

Date	Open	High	Low	Close	Volume	Adj Close*
8-Jan-99	162.20	165 00	162.20	163.00	5,220	78.96
7-Jan-99	162 00	162 50	160.00	160.00	3,647	77.51
6-Jan-99	160.90	164.00	160.90	164.00	23,616	79.45
5-Jan-99	154.00	155.00	154.00	155 00	5,975	75.09
4-Jan-99	155.00	158.00	155.00	155.50	7,024	75.33
30-Dec-98	311.50	311.50	309.00	309.00	28,324	149.69
29-Dec-98	314.90	314.90	313.30	314.00	22,313	152.12
28-Dec-98	314 80	314 80	314.00	314.00	26,640	152.12
23-Dec-98	303 50	305 00	302.50	305 00	38,363	147.76
22-Dec-98	293 60	297 50	293.00	294.00	29,899	142 43
21-Dec-98	282 50	293.00	282.50	293.00	27,603	141.94

* Close price adjusted for dividends and splits.

New York, USA

PRICES

Date	Open	High	Low	Close	Volume	Adj Close*
8-Jan-99	191 00	192.00	185.63	187.56	4,593,100	89.99
7-Jan-99	187.94	192 38	187 00	190.19	4,155,300	91.25
6-Jan-99	190.31	192.75	188 50	188.75	4,775,300	90.56
5-Jan-99	183.00	189.88	182.81	189.63	4,955,900	90.98
4-Jan-99	185 00	186 50	181.50	183.00	4,077,500	87 80
31-Dec-98	186.75	187 19	183 50	184 38	1,933,800	88.46
30-Dec-98	186.88	188.63	186.31	186.75	2,410,300	89.60
29-Dec-98	188 63	188.94	187.00	187.13	1,881,700	89.78
28-Dec-98	186 50	189 94	186.00	189 25	2,637,200	90.80
24-Dec-98	184 75	187.94	184.06	187 94	1,527,700	90 17
23-Dec-98	182 69	185.38	181.13	185.00	3,537,800	88.76
22-Dec-98	177 50	183.00	175.25	182.25	4,353,600	87.44
21-Dec-98	171 56	178.94	171 56	176 38	3,744,500	84.62

* Close price adjusted for dividends and splits.

Figure 2.2 Historical stock prices for IBM (from Yahoo).

Comparing the values of adjusted close price in the Figure, we notice several surprising peculiarities. In the first row of data, the values enclosed in rectangles are significantly different: 78.96 in Frankfurt vs. 89.99 in New York. Even more dramatic differences are in the bracketed values in the lower half of the figure, which correspond to prices during the period near the end of 1998. In addition, noticing the circled values in the middle, we observe that the value in the Frankfurt table dropped by almost 50% on the opening day in 1999 while the change in New York was less than 1%.

How could this happen? It turns out that we need to know the currencies in which the prices are expressed in order to interpret the values. At the beginning of 1999, the Frankfurt Stock Exchange changed the currency from German Mark (DEM) to Euro (EUR) and the irrevocable exchange rate is

1 EUR = 1.95583 DEM[10], while the prices at New York Stock Exchange are always in US dollars (USD) and 1 USD = 1.67 DEM on December 30, 1998. Much of the difference between the two exchanges can be explained by the use of different currencies; similarly, the significant value drop on the opening day of 1999 in Frankfurt is due to currency change at the same exchange. In fact, there have been studies (e.g., Heimonen, 2002; Berbena and Jansenb, 2005) to show that worldwide financial markets have become more integrated and prices in different markets have been converging in the last two decades. That is, prices in different markets have been very close. ■

This example illustrates the challenges of lacking semantic interoperability among systems. Implicit assumptions made in each system need to be captured and used to reconcile their differences when data from these systems are combined. In the example, each stock exchange makes implicit assumption about the currency used for representing adjusted close price. The meaning of data is dependent on the assumption. For example, the number "149.69" circled in the Frankfurt table means (or denotes) something different depending on whether the currency is EUR or DEM. When the assumption changes, data semantics also changes. As a result, semantic heterogeneity exists not only between two stock exchanges but also within Frankfurt Exchange.

Example 2.2 (*Temporal ontological heterogeneity*) In everyday communications and in various information systems, it is very common that we refer to things using various codes, e.g., product codes of a company, subject numbers in a university subject catalog, and ticker symbols commonly used to refer to company stocks. Codes are sometimes reused in certain systems, thus the same code can denote different things at different times. For example, subject number "6.891" at MIT has been used to denote "Multiprocessor Synchronization", "Techniques in Artificial Intelligence", "Computational Evolutionary Biology", and many other subjects in the past decade. As an another example, ticker symbol "C" used to be the symbol for Chrysler; after Chrysler merged with Daimler-Benz in 1997, the merged company chose to use "DCX"; on December 4, 1998, the symbol "C" was assigned to Citigroup, which was listed as "CCI" before this change. ■

In Example 2.2., while a code may denote different things at different times, there is always a one-to-one correspondence between a code and what the code denotes in a given system. In Example 2.3 below, we illustrate that sometimes a one-to-one correspondence may become a one-to-many correspondence, or vice versa.

Example 2.3 (*Temporal ontological heterogeneity*) Data from multiple sources are often required to study economic and environmental development of the world. In the past 30 years, certain regions have gone through significant restructuring, e.g., one country breaking up into several countries. Such dramatic changes often make it difficult to use data from multiple sources or even from a single source. As an example, suppose a Balkans specialist is interested in studying the CO_2 emissions in the region of former *Yugoslavia* during 1980-2000 and prefers to refer to the region (i.e. the geographic area of the territory of former Yugoslavia) as Yugoslavia. Data sources like the Carbon Dioxide Information Analysis Center (CDIAC)[11] at Oak Ridge National Laboratory organize data by

[10] http://www.oanda.com/site/euro.shtml

[11] http://cdiac.esd.ornl.gov/home.html

country. Figure 2.3 lists some sample data from CDIAC. Yugoslavia as a country, whose official name is *Socialist Federal Republic of Yugoslavia* in 1963-1991, was broken into five independent countries in 1991: *Slovenia*, *Croatia*, *Macedonia*, *Bosnia and Herzegovina*, and *Federal Republic of Yugoslavia* (also called *Yugoslavia* for short in certain other sources). Suppose prior to the break-up the specialist had been using the following SQL query to obtain data from the CDIAC source:

```
Select CO2Emissions from CDIAC where Country = “Yugoslavia”;
```

Before the break-up, “Yugoslavia” in the receiver coincidentally referred to the same geographic area as to what “Yugoslavia” in the source referred, therefore, the query worked correctly for the receiver until 1991. After the break-up, the query stopped working because no country is named “Yugoslavia” (or had the source continued to use “Yugoslavia” for Federal Republic of Yugoslavia, the query would return wrong data because “Yugoslavia” in the source and the receiver refer to two different geographic areas). ■

Country	**Year**	**CO2Emissions**[12]
...	...	...
Yugoslavia	1990	35604
Yugoslavia	1991	24055
Slovenia	1992	3361
Croatia	1992	4587
Macedonia	1992	2902
Bosnia-Herzegovinia[13]	1992	1289
Federal Republic of Yugoslavia	1992	12202
...	...	...

Figure 2.3 Sample CO_2 emissions data from CDIAC.

In this example, the implicit assumption is what “Yugoslavia” denotes. For the receiver, it always denotes a particular geographic area; the area happens to be denoted by “Yugoslavia” in the source until 1991, and by five different names since 1992. According to our categorization, there is temporal ontological semantic heterogeneity between the source and the receiver.

Alternatively, we can view the implicit assumption in terms of the organization of data: until 1991, the source organizes emissions data the same way the receiver expects; since 1992, the sum of emissions of five particular records in the source corresponds to one record expected by the receiver.

We mentioned in Section 2.1 that there can be multiple kinds of semantic heterogeneity in one data element. In the example, the CO2Emissions data in the source also make implicit assumptions in three aspects: the emissions are in 1000s (*scale factor*) of metric tons (*unit of measure*) of carbon (not CO_2 gas, hence *ontology/definition*). Thus for this data element alone there could be both representational and ontological heterogeneity should other sources and receivers make different assumptions in these aspects.

A variety of implicit assumptions can change over time, e.g., units of measure, scale factors, naming conventions, accounting standards, concept definitions, and data gathering and processing methods. For example, while people in the 1980s believed that computers played a big role of

[12] The actual column is in the source is “Total CO2 emissions from fossil-fuels (thousand metric tons of C)”.

[13] Correct spelling is Herzegovina.

improving productivity in the U.S, the National Income and Product Accounts (NIPA)[14] data from the Bureau of Economic Analysis (BEA) did not show any evidence to support this belief. To a certain extent, this productivity paradox was a measurement problem, i.e. the quality improvement of computers was not measured in BEA's NIPA data (Jorgensen, 2001). To improve the productivity measurement, the BEA introduced a series of changes to it accounting methods. One of major changes is to use "constant quality price" (CQP) for various products of the information technology industry, e.g., computers (since 1985), digital telephone switching equipment and cell phones (since 1996), semiconductors (since 1997), and prepackaged software (since 1998). Another change is the reclassification of software expenditures as investment in 1999 (Jorgensen, 2001). Likewise, there have been certain changes to the definition of unemployment data provided by the Department of Labor[15]. A few other examples can be found in Ventrone and Heiler (1991). Although certain data sources retroactively update historical data to conform to current standards, for various reasons certain data in many data sources are not updated, as seen in the example of the Frankfurt stock price.

2.2.2 Different Time Models

Although the notion of *time* is often taken for granted in everyday use, it is actually a rather complex concept that has multiple meanings, e.g., there are as many as 14 entries for Time as a noun in Merriam-Webster Online Dictionary[16]. Hayes (1996) identifies six commonly used senses of time: (1) a physical dimension; (2) a plenum (or "container") within which all events are located; (3) a duration (e.g., 3 hours); (4) an interval (or period); (5) a point; and (6) a position in a temporal coordinate system. To appreciate the complexity of time, a printout of the documentation needed to work with dates and times in the Java programming language (the Java Calendar and GregorianCalendar classes) is typically over 40 pages long.

There are two major approaches to representing temporal phenomena: change-based and time-based (Shoham and Goyal, 1988). The intuition for the change-based approach is that time is important only when the world changes. In this approach, time is implicitly embedded in the changing states of the world or the relationships of events. The primitives are usually actions, as in situation calculus (McCarthy and Hayes, 1969), or events, as in event calculus (Kowalski and Sergot, 1986).

Time is explicitly introduced in the time-based approach, which is common in many information systems. It is desirable to have a general time model that is reusable in different systems. But different systems often have different needs for time representation, therefore not all senses of time are necessary in a particular system and the same sense may be formalized differently in different systems. As a result, there have been many time theories developed for different purposes. This is

[14] Gross Domestic Product (GDP) and Gross Domestic Income (GDI) are computed from the national income and product accounts prepared by the BEA.

[15] See the FAQ at http://www.bls.gov/cps/cps_faq.htm. The answer to question "Have there been any changes in the definition of unemployment" states: "The concepts and definitions underlying the labor force data have been modified, but not substantially altered".

[16] http://www.m-w.com/

also the case in a number of recent efforts of developing time ontologies (Zhou and Fikes, 2000; Hobbs and Pan, 2004), which define a set of concepts of time and their relationships.

Maiocchi and Pernici (1991) summarize five choices that most time theories need to make:

(1) primitive time entity: point or interval
(2) time ordering: linear, branching, or circular
(3) time structure: maps to the subset of Z (integer), Q (rational), or R (real)
(4) boundedness: if time can be infinite and if open time interval is allowed
(5) time metric: units for measuring duration

The choices are made depending on the application needs. For example, an interval-based theory is used in (Allen, 1983) to represent and reason about temporal relationships of events and states. The reasoning takes $O(n^3)$ time (n is the number of intervals in the constraint network). However, an interval-based theory cannot represent continuous changes (Galton, 1990). Point-based theories do not have this limitation (Koubarakis, 1994).

Most of the time models developed in Artificial Intelligence do not have a concrete representation of time because their focus is on the relationships of the times associated with states and events. For example, the interval logic of Allen (1983) is specifically devised for representing relative temporal information where the exact dates or even fuzzy dates are not available. The kinds of temporal information often have forms like "being in college" is *after* "being in high school" *and before* "having a full-time job". Here the primary interest is the temporal relations of propositions (e.g., "being in college") that represent states or events.

However, there are cases where the time position in a time coordinate system is known. For example, we know that Frankfurt Stock Exchange started to use Euros from January 1, 1999. Most temporal information recorded in databases has explicit time, sometimes called a timestamp. Time series data also contains explicit time at certain granularities, e.g., daily, weekly, monthly, etc. This Thesis focuses on heterogeneities in systems that contain explicit time. In these systems, time is usually identified as a date (possibly with clock time) in a calendar (usually the Gregorian calendar). As with other types of data, time data in different systems often have different semantic assumptions, representations, and supported operations.

Different ontological assumptions. Common implicit assumptions for date and time include calendar used, location or time zone, and standard time vs. Daylight Saving Time (DST). Under different assumptions, a date and time string may denote different time points. For example, which day "April 19, 2005" denotes depends on which calendar is used. If Julian calendar is used, it denotes the day of "May 2, 2005" in Gregorian calendar; if Chinese calendar is used, it denotes the day of "May 26, 2005" in Gregorian calendar.

Although most modern systems use the Gregorian calendar by default, other calendars can be used on certain occasions, e.g., for recording cultural events. Conversions between different calendar systems can be performed. There are also research efforts that aim to provide systematic multi-calendar support (Soo and Snodgrass, 1995; Bry and Spranger, 2004).

There are approximately 70 countries in the world that use DST, but DST starts and ends on different dates in different countries and regions[17]. The starting and ending dates in a country can change over time. For example, DST in the U.S. has changed several times, as summarized in the case study in Figure 2.4[18]. These aspects are often implicitly assumed in systems, e.g., local time vs. UTC (coordinated universal time); standard time vs. daylight saving time (or summer time).

1918-1919	"An act to preserve daylight and provide standard time for the United States" was enacted on March 19, 1918. Under the Act, DST was in use from last Sunday in March to last Sunday in October. At the end of World War I, the law was repealed and only a few states and cities continued to observe DST.
1920-1941	No federal law for DST
1942-1945	"An act to promote the national security and defense by establishing daylight saving time" was enacted on January 20, 1942. Under this law, DST was year-round from February 2, 1942 to September 30, 1945.
1946-1965	No federal law for DST.
1966-1971	"The Uniform Time Act of 1966" was enacted on April 12, 1966. DST was from last Sunday of April to last Sunday of October. States can pass law for exemption.
1972-1985	In 1972, the Act was amended to provide options for States residing in multiple time zones.
1986-present	In 1986, the Act was amended to set the beginning of DST to first Sunday of April.

Figure 2.4 Changes to Daylight Saving Time (DST) in the U.S.

Different representations. Even when a single calendar is used, there are various representations for date and time in terms of format and structure. These different representations continue to exist despite the international standard "ISO 8601" (ISO, 2000), which specifies standard representations of date and time. In the following discussion, we will use the term *temporal entity* to refer to date, time and combination of date and time.

The different date formats create a number of problems in information integration:

- *Semantic ambiguity* – e.g., what is "03/04/05"? There are three commonly used sequences for day (D), month (M), and year (Y); there are also two conventions of interpreting a two-digit year (e.g., "03" as 1993 or 2003). Therefore, "03/04/05" can denote six possible dates, ranging from April 5, 1903 to April 3, 2005. The ambiguity can be eliminated by using a standard (e.g., ISO 8601) or if one knows the "syntax" of how different parts of a calendar date and clock time are assembled into a single temporal entity.
- *Proliferation of conversions* – For any given date, there can be numerous representation styles. Given:
    - 3 styles for month: April, Apr, 04
    - 3 orders for parts: DMY, MDY, YMD (D: day, M: month, Y: year)
    - 3 kinds of separators: -, /, space (or no space)
    - 2 ways of writing year: 4-digit and 2-digit

[17] See http://webexhibits.org/daylightsaving/g.html for a list of DST start and end dates in different countries.

[18] Adapted from http://webexhibits.org/daylightsaving/index.html.

there are a total of 54 possible combinations out of these choices, i.e., 54 different representations for any given date. A typical database management system (DBMS) supports commonly used formats and provides conversion functions for converting between these formats. For example Microsoft SQL Server uses the following function

```
CONVERT(data_type [( length )], expression [, style])
```

for date format conversion. The *style* parameter has predefined values to represent certain formatting patterns, e.g., 101 is for "mm/dd/yy" and 107 is for "Mon dd, yy". An example of using CONVERT is given in the *Where* clause a few paragraphs below. In addition, Microsoft also supplies a DTS (Data Transformation Service) package to allow programmers to write data transformation programs. Date-Time string transformation is one of the built-in transformations. Oracle uses the following two functions to convert data formats:

```
TO_CHAR(<date>, '<format>')
TO_DATE(<string>,'<format>')
```

where *format* is a pattern string assembled using a set of predefined tokens.

- *Need for query rewriting* – Temporal entities are often represented in different formats, sometimes implemented using incompatible data types (e.g., system supported DATE type vs. STRING type). As a result, query expressed against one system cannot be directly evaluated in another system. For example, suppose Quote_date is present in an Oracle database as well as a SQL Server database using the system supported DATE type and DATETIME type, respectively. By default, date format in Oracle is "dd-MON-yy"[19], such as "03-APR-05" for April 3, 2005. When a date string with correct format is compared with a value of DATE type, Oracle first converts the string into DATE type before comparing. Thus the *Where* clause in a query to the Oracle database may look like the following:

  ```
  Where Quote_date = '03-APR-05'
  ```

  However, SQL server does not automatically convert a date string to DATETIME type. Therefore, a direct evaluation of the above *Where* clause in a SQL server will always return an empty result. To ensure correct evaluation, the above *Where* clause must be rewritten to convert data into the same type, e.g., converting DATETIME to string type like in the following:

  ```
      Where CONVERT(varchar(10), Quote_date, 101) = '04/03/05'
  ```

  As we will see next, the need for query rewriting also arises where there are structural differences and the differences in supported operations over the data types used for temporal entity representation.

Many data sources do not use system supported time related data types for temporal entities. Instead, they use other primitive types, such as string, to represent time and implement the time model in application code. When this is the case, in addition to format differences, there can be

[19] Date format template in Oracle databases is different from than in the Java programming language. In Oracle, "MON" indicates the use of abbreviated month name, "APR" for month April.

structural (i.e., schematic) and granular differences. Figure 2.5 illustrates just a few examples in clinical data (Das and Musen, 2001).

(a)

Month	Day	Year	Patient ID	Weight
1	23	3001	41288	71

(b)

Date	Length	Unit	Patient ID	Problem
Jan 23 2001	2	Week	41288	Pneumonia

(c)

Time Drawn	Time Entered	Patient ID	Creatinine
Feb 1 2001 7:01:23AM	Feb 1 2001 10:12:54AM	41288	2.3

(d)

Start Time	End time	Patient ID	Drug Name	Drug Dose	Frequency	Route
Jan 30 2001 5:40PM	Now	41288	Gentamicin	150mg	8	IV

Figure 2.5 Heterogeneities of Temporal Entities in Clinical Data (Das and Musen, 2001).

As shown in Figure 2.5, a temporal entity can be split into multiple parts as in (a) or be in one piece as in the rest, with or without time part; they can have different granularities; an interval can be represented using start time and end time in two separate columns as in (d), or using a date and a duration prior to the date as in (b). Obviously, these differences require that a query expressed against one structure to be rewritten for a different structure to ensure the executablility and correctness of the query.

Different supported operations on temporal entities (*capability heterogeneity*). Different systems often have different implementations for temporal entities. As a result, the supported operations on temporal entities differ amongst systems. We call such differences *capability heterogeneity*. For example, not all systems support the *subtraction* operation on two dates to give the duration as the number of days between the dates, *next* operation on a date to return the next day, or *overlaps* operation in temporal database query language TSQL2 (Snodgrass, 1995). Certain systems do not even support all of the usual comparison operators: <, >, >=, <=, and =. For example, to retrieve stock prices between December 21, 1998 and January 8, 1999 like those shown in Figure 2.2, one would expect to be express the date range in a SQL query like the following:

```
WHERE ... DATE>=″21-DEC-98″ and DATE=<″8-JAN-99″ and ...
```

But if the source does not support operators such as >= and =< on the DATE attribute, the query cannot be executed as expressed. As we will see in Chapter 3, Yahoo data source does have such limitations and the date range conditions have to be translated to *equality* on StartYear, StartMonth, StartDay, EndYear, EndMonth, and EndDay attributes.

The problem of capability heterogeneity exists for other types of data and has been recognized in information integration research (Yerneni, 2001). Although capability heterogeneity is a separate issue from semantic heterogeneity, a query answering system needs to resolve such heterogeneity so that when a user query violates certain source restrictions, the query answering system can rewrite the query into a form executable at the source.

2.2.3 Different Associations with Time

In the previous section, we focused on *time* itself and its representation in information systems. In this section, we look at different ways of associating time to non-temporal data and different interpretations of these associations. Temporal database literature distinguishes two dimensions of time – *valid time* and *transaction time* (Snodgrass and Ahn 1985; Jensen and Snodgrass, 1996). Valid time concerns the state of the modeled world (i.e., when a certain thing is true in the world), and transaction time concerns the state of the database (i.e., when a certain record is current/stale in the database). In the rest of the Thesis, we refer to valid time unless otherwise noted.

Time-stamping is a common practice of associating time to non-temporal data. For example, Figure 2.6 shows a number of time-stamped records in a hypothetical bank account database (adapted from Bettini et al., 2000). Account 123 has three transactions in January 2005: on January 1, 2005, there is a deposit of 100 and the resulting balance is 800; a withdrawal occurs on January 15, 2005, and the resulting balance is 750, etc.

Account	Time	Transaction	Balance
123	1/1/2005	100	800
123	1/15/2005	-50	750
123	1/31/2005	200	950

Figure 2.6 Time-stamped bank account records.

With these records and other assumptions, such as that these records represent all changes that could have affected the balance, one can infer this account *has a balance of 800 on any given day between January 1 and January 14*; but one cannot say that the account *has a transaction of 100 on any given day between January 1 and January 14*. That is, the balance-time and transaction-time associations have different interpretations.

There are several formalizations of this phenomenon. Allen (1984) distinguishes three types of entities to be associated with time: *properties*, *events*, and *processes*. In Allen's interval logic, the attribute "balance" is a *property*, which holds true over an interval *iff* it holds true over any subinterval of the interval, while the attribute "transaction" is an *event*, which can occur over an indivisible interval.

Shoham (1987) introduces a logic that eliminates the need of using special predicates for different entities and their associations with time. This is a reified[20] logic that refers to the non-temporal part as *proposition type*. For example, color(house17, red) is a proposition type in predicate TRUE(t_1, t_2, color(hourse17, red)). This predicate is equivalently represented as a record in a database table that has four attributes: <Begin, End, House, Color>. Thus, a proposition type roughly corresponds to a relational attribute. The logic is expressive enough to distinguish different proposition types: *downward hereditary*, *upward hereditary*, *liquid*. A proposition type is downward hereditary if whenever it holds over an interval it holds for all its subintervals or its internal points; it is upward hereditary if whenever it holds true for all subintervals or internal points of an interval it holds for the interval; it is liquid if it is both downward and upward hereditary. According to these

[20] Reification is the process of turning a predicate or function into an object in the language.

definitions, “balance” is liquid, and “transaction” is non-hereditary. These characteristics can be used to compact data storage (e.g., balance need not be stored for every day or every second); implicit data can be derived based the liquidity of attributes (Bettini et al., 1998).

We observe that liquidity (or lack of liquidity) is determined by the nature of the attribute and it does not vary across data sources or change over time. For example, “balance” is liquid in all data sources and stays liquid all the time.

Certain functions can be applied over the time domain to obtain an aggregate attribute. Although liquidity does not vary among sources, temporal aggregation can be different in different sources. For example, in a precipitation database that records monthly precipitation, the value could be total precipitation of each month, or the max, min or average daily precipitation within the month. In this case, the function that is applied to obtain the aggregate determines the association of the attribute with time. These different aggregates are conceptually different. Conversions between these different aggregates are possible when the data to which functions are applied are available.

Sometimes different sources aggregate data over different granularities, e.g., daily precipitation in one source and monthly precipitation in another, or monthly sales vs. quarterly sales. Conversion from fine granularity to coarse granularity can be performed via aggregation, which may or may not involve approximation (e.g., conversion from monthly sales to quarterly sales involves no approximation, while conversion from weekly sales to monthly sales does). The conversion from coarse granularity to fine granularity can only be approximated or extra information is required (e.g., approximating monthly sales from quarterly sales by taking the average).

Although both cases in the two preceding paragraphs involve aggregation, there are differences between them. In the first case, different functions are applied over the same time period, in the second case the same aggregation function is applied over different time periods. Work in temporal aggregates (Lopez et al., 2005) has mainly focused on developing efficient algorithms to compute an aggregate from data in fine temporal granularity.

2.3 Existing Approaches to Temporal Information Management

Temporal information management has been extensively researched in logic, AI, and database. In this section, we review related literature from the perspectives of temporal information representation and temporal reasoning. For representation, time can be introduced explicitly or implicitly, time can have a homogenous representation or have heterogeneous representations. For reasoning, it may or may not handle uncertain time or uncertain temporal relations; similarly, it may or may not deal with data semantics that changes over time (i.e., dynamic). These aspects are schematically shown in Figure 2.7.

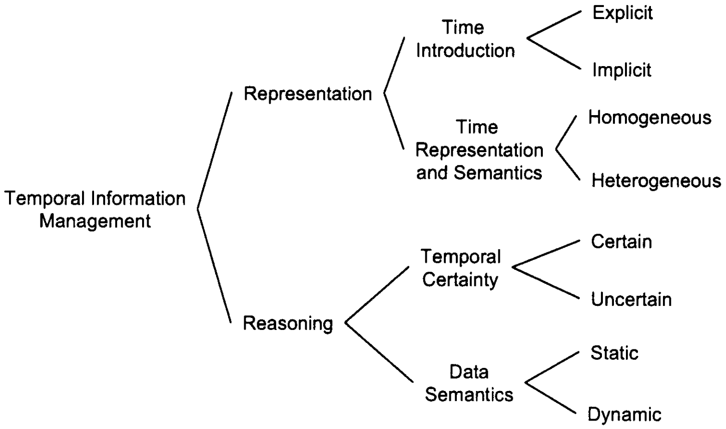


Figure 2.7 Categorization scheme for temporal information management.

Several logic formalisms introduce time implicitly. As mentioned earlier, time is implicit in situation calculus (McCarthy and Hayes, 1969) and event calculus (Kowalski and Sergot, 1986). The interval logic by Allen (1983) does not have explicit time representation either; it uses the propositions that hold true in certain time intervals as surrogates for time. Temporal modal logics (Kamp, 1968; Reichgelt, 1989; Halpen and Shoham, 1986) may or may not introduce time explicitly depending on what modal operators are supported. Non-reified (Bacchus et al., 1989), reified (Allen, 1984; McDermott, 1982; Shoham, 1987), and annotated (Frühwirth, 1996) temporal logics explicitly introduce time.

These formalisms are developed for problem solving in AI. Most of them can reason about uncertain temporal information and dynamic semantics. Uncertainty can exist in temporal relations (e.g., disjunctive temporal relations in Allen, 1983) or temporal values (e.g., Frühwirth, 1996). The formalisms are developed to particularly handle changes. However, when time is explicitly introduced, they all assume homogeneous time representation and semantics.

Temporal database research (see Özsoyoglu and Snodgrass, 1995 for a survey and Date et al., 2003 for a recent approach) focuses on efficient storing and querying large amounts of time-varying data (i.e., temporal data). One of the objectives is to offload users from managing various aspects of time, therefore, time is explicit from the system's point of view. This line of research does not concern semantic heterogeneity; it assumes that time has homogeneous representation and semantics. Earlier research deals with certain temporal information; recent work (Dyerson, 1994; Koubarakis, 1994) starts to develop techniques for managing uncertain temporal information.

Most existing information integration approaches do not systematically handle temporal semantic heterogeneity. In TSIMMIS (Garcia-Molina et al., 1995), semantic heterogeneity is identified in view definitions and a mediator is used to reconcile semantic heterogeneity through view expansion and query rewriting. Time-varying semantics can possibly be handled through view definitions that partition a data source into multiple ones. However, this is a manual process to be performed before query mediation. SCROL (Ram and Park, 2004) is a global schema approach that uses an ontology to explicitly categorize and represent predetermined types of semantic heterogeneity. Although heterogeneous representations of temporal entities can be handled by

SCROL, its ontology does not have constructs for representing time-varying semantics. COIN (Goh et al., 1997; Firat, 2003) uses a shared ontology for describing data semantics and a mediator to resolve semantic heterogeneities by query rewriting. Prior to the work described in this Thesis, its capability of dealing with temporal semantic heterogeneity is similar to that of TSIMMIS. These approaches (as well as work of this Thesis) cannot process uncertain information.

In addition to the above approaches that are intended for any application domain, there have been research efforts that aim to address semantic heterogeneity problems in specific domains. In the domain of medial applications, Shahar (1994) concerns automatic construction of interpretation contexts within which time series data can be appropriately abstracted and interpreted. For example, given a series of patient blood pressure readings, without any further information, the system may interpret the patient's blood pressure to be above normal over a 2-week period; however, if it is known that a drug with a side effect of increasing blood pressure had been administered during that period, the system will automatically construct a different interpretation context within which the same set of blood pressure data is interpreted as being normal. Although it facilitates data interpretation via abstraction and context construction, the interpretation within a context does not change over time.

Das and Musen (2001) focus on heterogeneity of temporal entities. They develop a canonical time-stamping scheme and translate timestamps in other formats into this canonical format. These translations need to be implemented manually using a number of operators specially designed for this purpose. They do not concern the semantics of non-temporal data, e.g., the unit of creatinine measurement in table (c) of Figure 2.5.

We use the categorization scheme of Figure 2.7 to summarize the related work, and present the results in Table 2.1; (Y) indicates that some, not all, approaches in the group have the indicated property or capability. The main contribution of this work is that it processes dynamic (i.e., time-varying) semantics in the presence of semantic heterogeneity in time itself; this unique combination of problems has not been addressed in previous information integration research.

Table 2.1 Various approaches to temporal information management.

	Time introduction		Time representation/semantics		Temporal uncertainty		Semantic assumptions	
	Explicit	Implicit	Homogeneous	Heterogeneous	Certain	Uncertain	Static	Dynamic
Temp. Logic	(Y)	(Y)	Y	N	Y	(Y)	Y	Y
Temp. Database	Y	N	Y	N	Y	(Y)	Y	N
Info Integration	Y	N	Y	Y	Y	N	Y	N
Med. Info Sys	Y	N	Y	Y	Y	N	Y	N
This work	Y	N	Y	Y	Y	N	Y	Y

2.4 Summary

We have identified three kinds of temporal semantic heterogeneity: (1) time-varying data semantics due to changes of implicit assumptions; (2) heterogeneous time models that include a number of aspects of time (i.e., temporal entity) itself, e.g., representations, interpretations, and operations of time; (3) different associations of non-temporal information to time. For (3), we observe that liquidity is dependent on attribute type and does not vary by source or time. Therefore, we need not deal with heterogeneous liquidity in information integration.

We reviewed related literature in logic/AI, temporal databases, and information integration. It can be seen that none of the existing research systematically addresses time-varying semantics and heterogeneous time models simultaneously. This Thesis will fill this gap. We will develop a representation formalism and reasoning mechanism to reconcile semantic heterogeneities where both changing semantics and heterogeneous time models are present. This will be developed as extensions to the existing COIN approach.

Specifically, we will focus on the integration of sources with explicit time and develop techniques to address the following issues encountered in the integration of temporal data sources:

- Time varying semantics
- Semantic heterogeneity in temporal entities
- Capability heterogeneity in terms of supported operations on temporal entities

We will focus on certain temporal information, and leave uncertain temporal information for future research.

The scope of this part of the Thesis can be further understood from the perspective of system interoperability that needs to resolve heterogeneity in all layers of information systems. With standards such as ODBC, Web Services, and XML, systems using different hardware and software platforms can exchange data and parse the exchanged data relatively easily. They essentially provide physical connectivity to allow different systems to exchange data, not the semantics of the data. For the receiver to combine data from different sources and correctly interpret the meaning of the data for various applications, schematic and semantic heterogeneities need to be resolved. Recent work on schema matching (Rahm and Bernstein, 2001; Hernandez et al, 2001; Bernstein et al., 2004; Madhavan et al., 2005; Bilke and Naumann, 2005) aim to automatically generate semantic correspondences between a pair of schemas. These algorithms are getting better at identifying similarities (e.g., *price* in schema S1 is similar to *sale_price* in schema S2), but they still lack the capability of identifying semantic heterogeneity (e.g., price in S1 includes tax but sale_price in S2 does not). This Thesis focuses on developing a formalism for representing changing semantics in various sources, and a reasoning technique for reconciling temporal semantic heterogeneity once the heterogeneity has been described using the representation formalism.

The approach developed in the Thesis and numerous other integration approaches use ontologies for representing semantics (Wache et al., 2001). Ontology also plays an important role on the Semantic Web for knowledge sharing and reasoning (Berners-Lee, et al., 2001); Firat (2003) discusses the relationship between COIN and the Semantic Web. The ontologies on the Semantic Web and in various systems may use different representation languages and terms, and have different structures. These differences need to be reconciled in large scale information integration that involves multiple ontologies. Research on ontology translation, alignment, and merging addresses this issue (Bruijn et al., 2004).

"Every general theory must work in at least one case."
– Stuart E. Madnick

Chapter 3

Resolving Temporal Semantic Heterogeneity using COIN

In the previous chapter, we identified various kinds of temporal semantic heterogeneity. Extensions to the COIN framework have been developed to reconcile these kinds of heterogeneity. The extensions do not change the architecture of COIN, which implements the framework and is briefly described below. Then we use three examples to illustrate the reconciliation of temporal semantic heterogeneity using the extended COIN. A summary of its capabilities and benefits is provided in the end. We intentionally keep the discussion intuitive, leaving the technical details to the next two chapters.

3.1 Overview of COIN Architecture

The COIN architecture, consisting of a graphic/web-based modeling tool and a mediation service, is depicted in Figure 3.1.

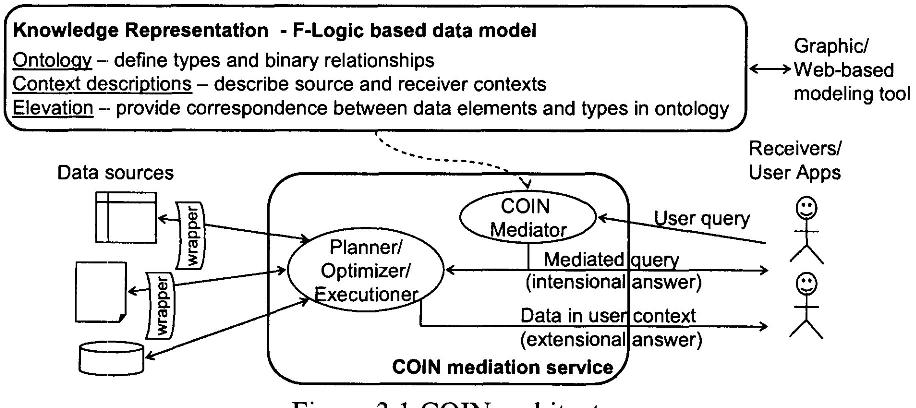

Knowledge Representation - F-Logic based data model
Ontology – define types and binary relationships
Context descriptions – describe source and receiver contexts
Elevation – provide correspondence between data elements and types in ontology
Graphic/ Web-based modeling tool
Receivers/ User Apps
Data sources
wrapper
wrapper
COIN Mediator
User query
Planner/ Optimizer/ Executioner
Mediated query
(intensional answer)
Data in user context
(extensional answer)
COIN mediation service

Figure 3.1 COIN architecture.

One of the objectives of COIN is to provide a system service so that users or user applications (i.e., receivers[21]) are not burdened with keeping track of and reconciling various kinds of semantic

[21] In this Thesis, we use *user* and *receiver* interchangeably.

heterogeneity. This is achieved by first using the modeling tool to explicitly record the implicit assumptions made by the data sources and receivers. The explicit representation of the assumptions, which we call context descriptions, is couched in an F-logic based data model that we will describe in the next chapter.

Once the various assumptions have been recorded using the modeling tool, a receiver can query all sources without the concern of semantic heterogeneity. The mediator component of the COIN mediation service intercepts the user query, compares the context descriptions of both the receiver and the data sources required to answer the query to detect differences, and outputs a mediated query (MQ) that incorporates the instructions on data conversions necessary for reconciling the differences. The query planner/optimizer/executioner (POE) then generates an optimized execution plan for the MQ and executes it to obtain the data instances with all necessary conversions performed.

COIN assumes that all data sources have a relational data model. This gives us the convenience of querying the data sources with SQL. When the actual implementation of a source is not relational, we can superimpose the relational model using wrappers. For example, we can use the Cameleon web wrapper (Firat et al., 2000) to make semi-structured web sources appear to be relational databases.

Traditionally, the MQ is called the *intensional* answer and the data instances are the *extensional* answer to the user query. Within the COIN system, the intensional answer is expressed in Datalog, which can be further translated into SQL. We assume the readers are familiar with SQL and the basics of relational calculus. An in-depth discussion on Datalog can be found in Ceri et al. (1989). Since we will use the Datalog form of the MQ in the example, a brief intuitive introduction to Datalog is provided below.

A Datalog query is the logical equivalent of an SQL query. For example, suppose there are two catalog database relations (i.e., tables) that list items and their prices: *d1(Item, Price)*, *d2(Item, Price)*. To find all items in *d1* whose prices are higher than those in *d2*, we can issue the following SQL:

```
Select d1.Item from d1, d2
Where d1.Item=d2.Item and d1.Price>d2.Price;
```

The following is an equivalent query in Datalog:

```
answer(Item) :- d1(Item,P1), d2(Item, P2), P1>P2.
```

Here, we use predicate *answer* to simulate the projection operator that outputs a list of attributes in the *Select* clause of SQL. A relation in the source is represented by a predicate, e.g., relation *d1* is represented by predicate *d1*. The attributes of a relation are referenced by the attribute names in SQL, in contrast, they are referenced by variables of arbitrary names as the arguments of the corresponding predicate in Datalog. The position of the argument determines the attribute being referenced. The same variable (e.g., *Item* in the above Datalog query) appearing in different predicates within the same Datalog query implies a *Join* condition. Conversely, variables with the same name in different Datalog queries are not related, e.g., variables *Product* and *P* in the following two queries are not related:

```
answer(Product, P) :- d1(Product, P).
answer(Product, P) :- d2(Product, P).
```

3.2 Example 1 – Yahoo Historical Stock Price

3.2.1 Integration Scenario

We have seen the Yahoo historical stock price example in Chapter 2. At the Yahoo Finance website, users can retrieve historical stock prices from most major stock exchanges by supplying the ticker symbol and the desired date range. To compare stock prices at different stock exchanges around the world, a user needs to retrieve data one exchange at a time and manually put them together. This is very time consuming. It becomes even more problematic when it comes to "understanding" the data because the prices at different exchanges may use different currencies and certain exchanges also changed currencies over time without any explicit indication. It is desirable that such temporal semantic heterogeneity is resolved by the system so that the users are not burdened with keeping track of and reconciling the differences.

For illustration purposes, let us consider a scenario that involves three stock exchanges, as depicted in Figure 3.2. Each exchange is viewed as a data source, for which we provide the schema, some sample data, and a brief description of the context. For example, YHNYSE is the source for stock prices at New York Stock Exchange (NYSE). When a source has only one relation, we use relation name and source name interchangeably. Auxiliary sources used for data transformation (i.e., conversion) are also given with necessary context descriptions.

YHNYSE

Ticker	QDate	AdjClose
IBM	4-JAN-99	87.57
IBM	31-DEC-98	88.23
IBM	30-DEC-98	89.36

Context:
Monetary value is in US dollars (as in c_usa)
Date format is d-MMM-yy (as in c_germany)

YHParis

Ticker	QDate	AdjClose
IBM.pa	4-JAN-99	75.68
IBM.pa	30-DEC-98	77.55

Context: c_france
Monetary value is in Euros
Date format is d-MMM-yy

YHFrankfurt

Ticker	QDate	AdjClose
IBM.f	4-JAN-99	75.05
IBM.f	30-DEC-98	149.14

Context: c_germany
Monetary value is
in German marks until the end of 1998
in Euros since the beginning of 1999
Date format is d-MMM-yy

Context: c_usa
Monetary value is US dollars
Date format is MM/dd/yyyy, e.g., 01/04/1999

Auxiliary sources:

Olsen (exchange rate):
<Expressed, Exchanged, Rate, Date>
c_olsen: Date format is MM/dd/yy, e.g., 01/04/99

Datecvt (date format conversion):
<date1, format1, date2, format2>

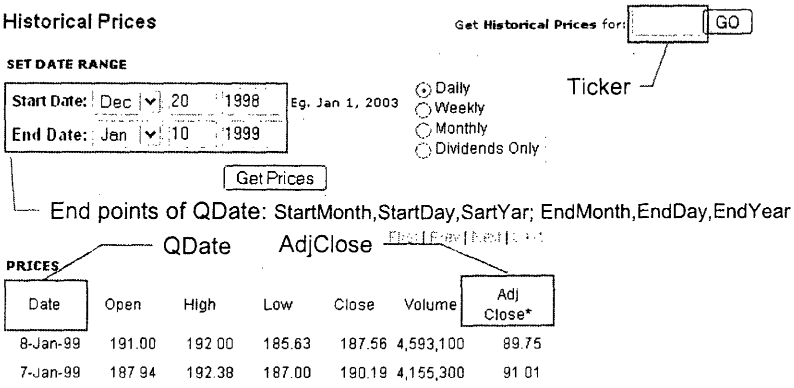
Figure 3.2 Integration of historical stock prices from multiple exchanges

A context is a set of descriptions about various aspects of the meaning of data. There are two such aspects in the example: (1) the currency used for price, which is a kind of monetary value; and (2) the date format useful for correctly interpreting date values. We use the format pattern in Java *SimpleDateFormat* class[22] for describing date format. We reference a context using a context label,

[22] See http://java.sun.com/j2se/1.5.0/docs/api/java/text/SimpleDateFormat.html for documentation.

i.e., identifier, e.g., *c_germany* is the identifier for the set of descriptions shown in the lower left corner of Figure 3.2.

We associate each column of a database table to a context label. For example, in source YHNYSE, the "QDate" column is associated to context label *c_germany* (thus we know the date format is d-MMM-yy), whereas "AdjClose" column is associated to context label *c_usa* (thus we know the currency is USD). When all columns of every table in a source are associated to the same context label, we say the source is in that particular context. For example, every column of YHFrankfurt is associated to context label *c_germany*, so we say that source YHFrankfurt is in context *c_germany*, or source YHFrankfurt's context is *c_germany*. We also associate each receiver to a context label.



Historical Prices

Get Historical Prices for: [GO]

SET DATE RANGE

Start Date: Dec 20 1998 Eg. Jan 1, 2003

End Date: Jan 10 1999

● Daily
○ Weekly
○ Monthly
○ Dividends Only

[Get Prices]

PRICES

Date	Open	High	Low	Close	Volume	Adj Close*
8-Jan-99	191.00	192 00	185.63	187.56	4,593,100	89.75
7-Jan-99	187 94	192.38	187.00	190.19	4,155,300	91 01

Ticker
End points of QDate: StartMonth,StartDay,SartYar; EndMonth,EndDay,EndYear
QDate
AdjClose

Figure 3.3 Historical stock quote interface at Yahoo Finance (adapted from Yahoo).

All the sources for historical stock prices are actually from the web site of Yahoo Finance (at finance.yahoo.com). We also use Olsen (at Oanda.com) as a source for historical exchange rates, and a servlet application called Datecvt for date format conversions. We use the Cameleon web wrapper to provide an SQL interface to these web sources so that they can be queried using SQL.

As an example, Figure 3.3 shows the actual user interface of historical stock prices at Yahoo Finance. Using the web wrapper, we can superimpose the following relational schema on the source:

```
<Ticker, QDate, AdjClose, StartMonth, StartDay, StartYear, EndMonth, EndDay, EndYear>
```

The last six attributes corresponds to the *month*, *day*, and *year* of "Start Date" and "End Date" and are necessary only because the data source does not accept inequalities on the "QDate" attribute. Not all sources have such restrictions. This is an example of capability heterogeneity discussed in Chapter 2. Like the semantic heterogeneity, capability heterogeneity should be handled by the integration technology, not the users. Therefore, users can specify the date range using >= and =< operators on "QDate" attribute and the COIN mediation service will convert them to the corresponding comparisons on the last six attributes in the schema.

This scenario illustrates several kinds of heterogeneity. First, the date formats are different in several contexts. This is a simple example of time model heterogeneity. Second, different currencies are assumed in different contexts; in the context of YHFrankfurt source, the currency also changed from German Marks (i.e., Deutschmarks) to Euros. In addition, the sources also have capability restrictions that other sources may not have. Therefore, we can use this simple scenario to illustrate the following features:

- Resolving time-varying semantic heterogeneity in the presence of time model heterogeneity
- Automatic query rewriting for source capability heterogeneity

3.2.2 Using COIN in this Example

Suppose a receiver in the context *c_usa*, as depicted in Figure 3.1, is interested in IBM stock price in Frankfurt during December 20, 1998 and January 10, 1999. To retrieve this information, the receiver can issue the following SQL:

Q1:

```
select QDate, AdjClose from YHFrankfurt
where Ticker="IBM.f" and QDate >="12/20/1998" and QDate =<"01/10/1999";
```

This query cannot be executed as is because of the source's inability in evaluating inequalities on "QDate"; even if it could, it does not return meaningful data to the user. Comparing the context definitions for the source and the user in Figure 3.2, we notice that there are currency and date format differences. The currency assumed in the source also changed within the specified date range. These capability restrictions, semantic differences and the change of semantics are recognized by the COIN mediator, which subsequently generates the following mediated Datalog query:

MDQ1:

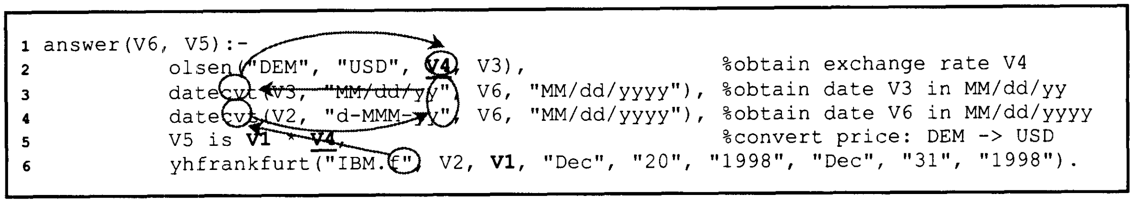
```
answer(V6, V5):-
        olsen("DEM", "USD", V4, V3),
        datecvt(V3, "MM/dd/yy", V6, "MM/dd/yyyy"),
        datecvt(V2, "d-MMM-yy", V6, "MM/dd/yyyy"),
        V5 is V1 * V4,
        yhfrankfurt("IBM.f", V2, V1, "Dec", "20", "1998", "Dec", "31", "1998").

answer(V6, V5):-
        olsen("EUR", "USD", V4, V3),
        datecvt(V3, "MM/dd/yy", V6, "MM/dd/yyyy"),
        datecvt(V2, "d-MMM-yy", V6, "MM/dd/yyyy"),
        V5 is V1 * V4,
        yhfrankfurt("IBM.f", V2, V1, "Jan", "1", "1999", "Jan", "10", "1999").
```

The corresponding mediated SQL query is:

```
MQ1:   select datecvt.date2, (yhfrankfurt.adjClose*olsen.rate)
       from   (select 'DEM', 'USD', rate, ratedate
               from   olsen
               where  exchanged='DEM'
               and    expressed='USD') olsen,
              (select date1, 'MM/dd/yy', date2, 'MM/dd/yyyy'
               from   datecvt
               where  format1='MM/dd/yy'
               and    format2='MM/dd/yyyy') datecvt,
              (select date1, 'd-MMM-yy', date2, 'MM/dd/yyyy'
               from   datecvt
               where  format1='d-MMM-yy'
               and    format2='MM/dd/yyyy') datecvt2,
              (select 'IBM.f', qDate, adjClose, 'Dec', '20', '1998', 'Dec', '31', '1998'
               from   yhfrankfurt
               where  Ticker='IBM.f'
               and    StartMonth='Dec' and StartDay='20' and StartYear='1998'
               and    EndMonth='Dec' and EndDay='31' and EndYar='1998') yhfrankfurt
       where  datecvt2.date1 = yhfrankfurt.qDate
       and    datecvt.date2 = datecvt2.date2
       and    olsen.ratedate = datecvt.date1
       union
       select datecvt3.date2, (yhfrankfurt2.adjClose*olsen2.rate)
       from   (select 'EUR', 'USD', rate, ratedate
               from   olsen
               where  exchanged='EUR'
               and    expressed='USD') olsen2,
              (select date1, 'MM/dd/yy', date2, 'MM/dd/yyyy'
               from   datecvt
               where  format1='MM/dd/yy'
               and    format2='MM/dd/yyyy') datecvt3,
              (select date1, 'd-MMM-yy', date2, 'MM/dd/yyyy'
               from   datecvt
               where  format1='d-MMM-yy'
               and    format2='MM/dd/yyyy') datecvt4,
              (select 'IBM.f', qDate, adjClose, 'Jan', '1', '1999', 'Jan', '10', '1999'
               from   yhfrankfurt
               where  Ticker='IBM.f'
               and    StartMonth='Jan' and StartDay='1' and StartYear='1999'
               and    EndMonth='Jan' and EndDay='10' and EndYear='1999') yhfrankfurt2
       where  datecvt4.date1 = yhfrankfurt2.qDate
       and    datecvt3.date2 = datecvt4.date2
       and    olsen2.ratedate = datecvt3.date1
```

The SQL syntax is a bit verbose, so we will examine the more concise Datalog query MDQ1. It has two sub-queries: one for the time period from <u>December 20, 1998</u> to <u>December 31, 1998</u>, the other for the time period from <u>January 1, 1999</u> to <u>January 10, 1999</u>. This is because the currency assumed in the source is German Mark in the first period and is Euro in the second period, each needing to be processed separately.

Let us focus on the first sub-query for the moment, which is reproduced in Figure 3.4 with line numbers and annotations added. Line 6 queries the YHFrankfurt source. Notice that the date range has been translated to equalities of the six attributes of *month*, *day*, and *year* of start date and end date of the actual schema; the values for month are now in the format required by the source, i.e., “Dec” for December. Variable V2 corresponds to “QDate”, V1 corresponds to “AdjClose”. None of them are in line 1 to be reported back to the user; the code in lines 2-5 has the instructions on how to transform them to V6 and V5 as values to be returned to the user.



```
1 answer(V6, V5):-
2         olsen("DEM", "USD", V4, V3),                   %obtain exchange rate V4
3         datecvt(V3, "MM/dd/yy", V6, "MM/dd/yyyy"),     %obtain date V3 in MM/dd/yy
4         datecvt(V2, "d-MMM-yy", V6, "MM/dd/yyyy"),     %obtain date V6 in MM/dd/yyyy
5         V5 is V1 * V4,                                 %convert price: DEM -> USD
6         yhfrankfurt("IBM.f", V2, V1, "Dec", "20", "1998", "Dec", "31", "1998").
```

Figure 3.4 Data transformations in MDQ1

The procedural reading of the code is:

- *line 4* converts "QDate" (V2) from the source format to the format expected by user (V6), i.e., from "d-MMM-yy" format (e.g., 20-Dec-98) to "MM/dd/yyyy" format (e.g, 12/20/1998);
- *line 3* converts V6 (from *line 4*) to V3 so that V3 has the format expected by source *olsen*, i.e., it converts date format from "MM/dd/yyyy" (e.g, 12/20/1998) to "MM/dd/yy" (e.g, 12/20/98);
- *line 2* queries the *olsen* source to obtain exchange rate (V4) between Deutschmark (DEM) and U.S. dollar (USD) for the date given by V3; and
- *line 5* converts "AdjClose" (V1) to USD using the exchange rate (V4) from *line 2*.

The second sub-query is almost the same except that it deals with a different date range within which the currency difference is EUR v. USD instead of DEM v. USD.

When the mediated query is executed, the user receives data instances[23] as shown in the left pane of Figure 3.5. For comparison, we also show the "raw" data from the source; notice that the unusual abrupt price drop in the raw data (which is actually due to the change in currencies) no longer appears in the mediated data.

Mediated results

QDate	AdjClose
01/08/1999	91.65
01/07/1999	90.10
01/06/1999	92.94
01/05/1999	88.28
01/04/1999	88.61
12/30/1998	89.27
12/29/1998	90.54
12/28/1998	90.14
12/23/1998	88.06
12/22/1998	84.84
12/21/1998	84.96
(user format)	(in USD)

Non-mediated results (original data)

QDate	AdjClose	
8-Jan-99	78.67	EUR
7-Jan-99	77.22	
6-Jan-99	79.15	
5-Jan-99	74.81	
4-Jan-99	75.05	
30-Dec-98	149.13	DEM
29-Dec-98	151.54	
28-Dec-98	151.54	
23-Dec-98	147.2	
22-Dec-98	141.89	
21-Dec-98	141.41	
(original format)		

Figure 3.5 Mediated and non-mediated data instances.

The mediated query for Q1 has two sub-queries because the currency in the source changed within the specified date range. When the user specifies a different date range within which there is

[23] Mediated results are rounded for easy reading.

no such change, the mediator will generate a mediated query accordingly. For example, when the user issues the following query to obtain more recent data:

```
Q2:    select QDate, AdjClose from YHFrankfurt
       where Ticker="IBM.f" and QDate >="05/01/2005" and QDate =<"05/13/2005";
```

The mediated Datalog query becomes

```
MDQ2:  answer(V6, V5):-
               olsen("EUR", "USD", V4, V3),
               datecvt(V3, "MM/dd/yy", V6, "MM/dd/yyyy"),
               datecvt(V2, "d-MMM-yy", V6, "MM/dd/yyyy"),
               V5 is V1 * V4,
               yhfrankfurt("IBM.f", V2, V1, "May", "1", "2005", "May", "13", "2005").
```

Notice that this mediated Datalog query contains only one sub-query similar to the second sub-query in MDQ1 – during the specified date range, the currency difference is EUR v. USD.

When the user wants to compare stock prices at the three stock exchanges for the same date range as in Q1, the following query can be issued to the mediator:

```
Q3:    select YHFrankfurt.QDate, YHFrankfurt.AdjClose, YHParis.AdjClose, YHNYSE.AdjClose
       from YHFrankfurt, YHParis,YHNYSE
       where YHParis.Ticker="IBM.PA"and YHNYSE.Ticker="IBM" and YHFrankfurt.Ticker="IBM.f"
       and YHFrankfurt.QDate=YHNYSE.QDate and YHFrankfurt.QDate=YHParis.QDate
       and YHFrankfurt.QDate >="12/20/1998" and YHFrankfurt.QDate =< "01/10/1999";
```

The mediated Datalog query MDQ3 (see below) looks similar to MDQ1, except that it queries three sources instead of only one. The corresponding mediated SQL query MQ3 is shown in the appendix at the end of the chapter.

```
MDQ3:  answer(V10, V9, V8, V7):-
               V9 is V6 * V5,
               olsen("EUR", "USD", V4, V3),
               datecvt(V3, "MM/dd/yy", V10, "MM/dd/yyyy"),
               olsen("DEM", "USD", V5, V3),
               datecvt(V2, "d-MMM-yy", V10, "MM/dd/yyyy"),
               V8 is V1 * V4,
               yhnyse("IBM", V2, V7, "Dec", "20", "1998", "Dec", "31", "1998"),
               yhparis("IBM.PA", V2, V1, "Dec", "20", "1998", "Dec", "31", "1998"),
               yhfrankfurt("IBM.f", V2, V6, "Dec", "20", "1998", "Dec", "31", "1998").

       answer(V9, V8, V7, V6):-
               datecvt(V5, "MM/dd/yy", V9, "MM/dd/yyyy"),
               olsen("EUR", "USD", V4, V5),
               V8 is V3 * V4,
               datecvt(V2, "d-MMM-yy", V9, "MM/dd/yyyy"),
               V7 is V1 * V4,
               yhnyse("IBM", V2, V6, "Jan", "1", "1999", "Jan", "10", "1999"),
               yhparis("IBM.PA", V2, V1, "Jan", "1", "1999", "Jan", "10", "1999"),
               yhfrankfurt("IBM.f", V2, V3, "Jan", "1", "1999", "Jan", "10", "1999").
```

The data instances are shown in Figure 3.6[24]. The currency has been converted into USD for all prices; the date is also expressed in user preferred format. In this uniform context, it is easy to compare the prices at different exchanges. Further analyses can be performed much more easily, e.g., one can calculate the maximum price differential across the three exchanges and find it to be $2.73 on 12/22/1998.

[24] In theory, the values in YHFrankfurt.AdjClose column in Figure 3.6 should be the same as those in the left pane of Figure 3.5, but we notice that they differ by up to a few cents. This is because that sometimes Yahoo Finance returns slightly different values when it is queried at a different time.

YHFrankfurt.QDate	YHFrankfurt.AdjClose	YHParis.AdjClose	YHNYSE.AdjClose
01/08/1999	91.67	91.10	89.75
01/07/1999	90.12	91.24	91.01
01/06/1999	92.95	92.50	90.32
01/05/1999	88.29	88.52	90.74
01/04/1999	88.62	89.36	87.57
12/30/1998	89.28	90.83	89.36
12/29/1998	90.55	91.26	89.54
12/28/1998	90.14	91.76	90.56
12/23/1998	88.07	88.94	88.52
12/22/1998	84.85	87.58	87.21
12/21/1998	84.97	86.25	84.40

Figure 3.6 Price comparison for IBM stock, 1998-1999

With the support of the mediation service, it becomes much easier to perform price comparison and other analyses. Users can use the sources without the burden of keeping track of their contexts and how the contexts change over time. They can also use the sources without being concerned about source capability differences.

3.3 Example 2 – Economic and Environmental Development in Yugoslavia

The Yugoslavia example in Chapter 2 shows that temporal semantic heterogeneity can arise when the real world undergoes changes. Based on the example, we construct a scenario that involves two data sources and one receiver, as shown in Figure 3.7. The sources share the same context.

Context *c_src*

1. *Monetary values* are in official currency of the country, with a scale factor of 1M;
2. *Mass* is a rate of tons/year with a scale factor of 1000;
3. All *other numbers* have a scale factor of 1;
4. All values are aggregated by country.

Schema of source 1:

```
Statistics(Country, Year, GDP, Population)
```

Schema of source 2:

```
Emissions(Country, Year, CO2)
```

Context *c_receiver*

1. *Monetary values* are always in USD, with a scale factor of 1M;
2. *Mass* is in tons/year;
3. *Other numbers* have a scale factor of 1;
4. If country code is "YUG", aggregate values for the region of the former Yugoslavia.

Figure 3.7 Sources and receiver contexts in Yugoslavia example.

In the scenario, the receiver is interested in the longitudinal economic and environmental changes in the region of the former Yugoslavia; however, the sources organize the data by sovereign country. Before the former Yugoslavia was broken up into five countries in 1992, the sources and the receiver coincided on how data is organized; from 1992 onward, data for the five countries need to be aggregated to meet the receiver need. In other words, the receiver wants to continue to use "Yugoslavia" to refer to the region within the boundaries of the former country "Yugoslavia". In addition, the sources and the receiver also make other implicit assumptions in terms of currencies and scale factors, as shown in Figure 3.7. The five countries and their official currencies are shown in Table 3.1.

Table 3.1 Countries and their currencies of former Yugoslavia

Country	Code	Currency	Currency Code
Yugoslavia[25]	YUG	New Yugoslavian Dinar	YUM
Bosnia and Herzegovina	BIH	Marka	BAM
Croatia	HRV	Kuna	HRK
Macedonia	MKD	Denar	MKD
Slovenia	SVN	Tolar	SIT

With the support of the mediation service, the receiver can issue the following query to retrieve the data:

Q4:

```
Select S.Country,S.Year,GDP,CO2
From Statistics S, Emissions E
Where S.Country=E.Country and S.Year=E.Year and S.Country=”YUG”;
```

The mediator generates the following mediated query to ensure that semantic differences both before and after the break-up of the region are correctly reconciled:

MDQ4:

```
answer(V8, V7, V6, V5) :-
     V5 is V4 * 1000.0,
     olsen("YUM", "USD", V3, V7),
     statistics("YUG", V7, V2, V1),
     emissions("YUG", V7, V4),
     V7 =< 1991,     V6 is V2 * V3.
answer(V96, V95, V94, V93) :-
     V92 is V91 * 1000.0, V90 is V89 * 1000.0,
     V88 is V87 * 1000.0, V86 is V85 * 1000.0,
     V84 is V83 * 1000.0, V82 is V90 + V92,
     V81 is V88 + V82, V80 is V86 + V81,
     V93 is V84 + V80, V79 is V78 * V77,
     V76 is V75 * V74, V73 is V72 * V71,
     V70 is V69 * V68, olsen("SIT", "USD", V67, V95),
     Statistics("SVN", V95, V66, V65),
     olsen("MKD", "USD", V68, V95),
     statistics("MKD", V95, V69, V64),
     olsen("HRK", "USD", V71, V95),
     statistics("HRV", V95, V72, V63),
     olsen("BAM", "USD", V74, V95),
     statistics("BIH", V95, V75, V62),
     olsen("YUM", "USD", V77, V95),
     emissions("SVN", V95, V83),
     emissions("MKD", V95, V85),
     emissions("HRV", V95, V87),
     Emissions("BIH", V95, V89),
     statistics("YUG", V95, V78, V61),
     emissions("YUG", V95, V91),
     1992 =< V95,     V60 is V66 * V67,
     V59 is V76 + V79, V58 is V73 + V59,
     V57 is V70 + V58, V94 is V60 + V57.
```

The first sub-query reconciles semantic differences for the time period before the break-up: information in the source is organized the same way the receiver expects, but there are currency and scale factor differences. The second sub-query is more complicated because what the receiver expects corresponds to the sum of the data of the five countries in each data source; currency and scale factor differences in these records need to be reconciled as well.

[25] The Federal Republic of Yugoslavia was renamed Serbia and Montenegro in 2003. We will not encode this change in the example to simplify illustration.

3.4 Example 3 – Company Financials

In the preceding examples, only one semantic aspect changes (and it only changes once) during the relevant time period. The temporal semantic heterogeneities resulting from such a "simple" change are by no means "simple", as illustrated by the complex mediated queries. Furthermore, the mediation service is not limited to "one-time change of one thing" at all. We will show an example where multiple semantic aspects change more than once.

The example involves a receiver who is interested in annual financial data of a certain company. The context definitions and source schema are shown in Figure 3.8. Notice that profit data in the source makes assumptions on currency, scale factor, and profit definition in terms whether taxes are included; all of the assumptions changed over time, not necessarily at the same time. The scale factor changed twice, whereas currency and profit definition changed only once. The scale factor for *num_employee* also changed once. Receiver context can change over time, as well. For simplicity, we illustrate with a receiver whose context is static in the example.

Context *c_src*

1. All *monetary values* are in French Francs until 2000 and in Euros afterwards;
2. All *monetary values* have a scale factor of 1M until 1999, 1K until 2001, and 1M from 2002
3. *Profit* is tax excluded until 2000, tax included afterwards
4. All *other numbers* have a scale factor of 1 until 2001 and 1K afterwards

Schema:

```
   Financials(Year, Num_Employee, Profit, Tax)
```

Context *c_receiver*

1. All *monetary values* are always in USD;
2. All *monetary values* always have a scale factor of 1K
3. *Profit* is always tax included
4. All *other numbers* always have a scale factor of 1K

Figure 3.8 Source and receiver contexts in the company financials example.

Again, with the support of the mediation service, the receiver need not worry about context differences and how contexts evolve. For example, the receiver can issue the following query to retrieve company annual data from year 2000 and onward:

```
Q5:     Select Year,Num_Employee,Profit
        From Financials
        Where 2000=<Year;
```

The following mediated query is generated to reconcile all semantic differences for the specified time range:

```
MDQ5:  answer(2000, V21, V20) :-
             V21 is V19 * 0.001,               % adjusting num_employee to 1K
             financials(2000, V19, V18, V17),
             olsen("FRK", "USD", V16, 2000),   % V16 is exchange rate
             V15 is V18 * V16,                 % converting profit to USD
             V14 is V17 * V16,                 % converting tax to USD
             V20 is V15 + V14.                 % summing up to get tax-included profit
       answer(2001, V13, V12) :-
             V13 is V11 * 0.001,               % adjusting num_employee to 1k
             financials(2001, V11, V10, V9),
             olsen("EUR", "USD", V8, 2001),    % V8 is exchange rate
             V12 is V10 * V8.                  % converting tax-included profit to USD
       answer(V7, V6, V5) :-                 % (no difference for num_employee)
             olsen("EUR", "USD", V4, V7),      % V4 is exchange rate
             financials(V7, V6, V3, V2),
             2002 =< V7,                       % year>=2002
             V1 is V3 * V4,                    % converting profit to USD
             V5 is V1 * 1000.0.                % adjusting for scale factor
```

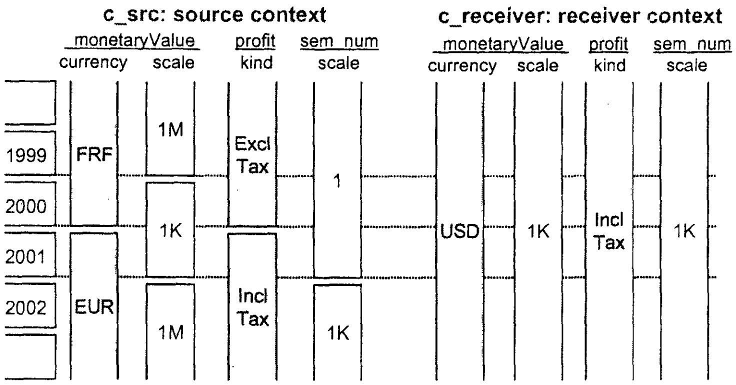


Figure 3.9 Graphical comparison of source and receiver contexts

To explain MDQ5, we need the assistance of Figure 3.9, which graphically presents the context descriptions of Figure 3.8. In the figure, horizontal lines are drawn to indicate context changes. Time line is on the left; each column shows how a modifier value evolves along the time line. A monetary value, such as profit or tax, makes assumptions about currency and scale factor; a profit also makes assumption on whether taxes are included; the number of employees also makes assumption about scale factor.

For the year 2000 and beyond, we see three time periods separated by the horizontal lines; each sub-query in MDQ5 corresponds to a time period. For example, when Year = 2000, the profit data in the source is in FRF and it excludes taxes, while the receiver expects it to be in USD and with taxes included; both the source and receiver assume the scale factor is 1K. In addition, the number of employees has different scale factors: 1 in the source, 1K in the receiver. The annotation in MDQ5 shows that these differences are correctly reconciled. The other two sub-queries can be understood with the annotations and Figure 3.9.

If the user issues a query like Q5 but without the WHERE clause, the mediated query will consist of four sub-queries: in addition to the three in MDQ5, there is a fourth sub-query that reconciles the semantic differences for Year=<1999:

```
answer(V32, V31, V30) :-
    V29 is V28 * 1000.0,
    V31 is V27 * 0.001,
    olsen("FRK", "USD", V26, V32),
    V28 is V25 * V26,
    financials(V32, V27, V25, V24),
    V32 =< 1999,
    V23 is V24 * V26,
    V22 is V23 * 1000.0,
    V30 is V29 + V22.
```

If the user is only interested in historical employee data and issues the following query,

```
Q6:    Select Year,Num_Employee
       From Financials;
```

the mediated query will consist of only two sub-queries:

```
MDQ6:  answer(V9) :-
               financials(V8, V7, V6, V5),
               V8 =< 2001,
               V9 is V7 * 0.001.
       answer(V4) :-
               financials(V3, V4, V2, V1),
               2002 =< V3.
```

the first of which reconciles the scale factor difference (1 in the source, 1K in the receiver); the second of which does not perform any transformation because there is no difference between the source context and the receiver context (both have a scale factor of 1K).

3.5 Summary

As mentioned earlier, one of the main objectives of COIN is to relieve users from keeping track of and reconciling semantic heterogeneities. This is accomplished by providing a system service that records contexts and reconciles semantic differences automatically. With this service, any user whose context has been recorded can use all sources as if they were in the user context.

The mediator rewrites the user query into a mediated query that includes instructions on how to reconcile semantic differences; the mediated query can be optimized and executed to obtain data instances with all necessary transformations performed. The user can get the mediated query as the intensional answer and the data instances as the extensional answer. This two-step process has several benefits (Imielinski, 1987; Goh, et al., 1999). First, the intensional answer contains useful information that the extensional answer does not have – the data transformations in the MQ allow one to examine and infer what semantic differences are there and how they are reconciled[26]. Second, the mediated query can be saved for repeated uses or can be used by the administrator to create views[27]. This feature increases the usability of the mediation service. And lastly, it allows for the opportunity of applying existing query optimization techniques to optimize the mediated query so that the query can be executed efficiently.

The following features are due to the work reported in this Thesis.

In line with the primary objective of COIN, users also need not worry about how data semantics changes over time in the data sources. The system automatically determines the semantic differences in different time periods and reconciles them accordingly.

The mediated query usually consists of multiple sub-queries, one for each time period. In effect, these sub-queries create partitions for the data sources; within each partition, the data semantics does not change. Although partitioning can be done manually by the administrator, COIN dynamically adjusts the partitions according to the user query. This automatic and dynamic portioning offers certain advantages over manual and static partitioning.

First, the manual approach would generate more data sources, which complicates the task of writing user queries. In Example 3, to ensure that there is no time-varying semantics in each partition, the company financials data source must be partitioned into four (virtual) sources with the following conditions on "Year" attribute: (1) Year ≤1999; (2) Year=2000; (3) Year=2001; and (4) Year≥2002.

[26] A table can be generated by the mediation service to summarize all the aspects in which the source and the receiver are different.

[27] A view is a virtual table based on the result of a query.

Second, the manual approach sometimes creates too many partitions with respect to certain user queries, which adds unnecessary complexity. For example, four partitions are more than is necessary if the user is only interested in the number of employees; two partitions are enough because scale factor for number of employees changed only once in history. With four partitions, the user has to write four sub-queries, three of which deal with the same scale factor difference in the same way. In contrast, the COIN approach generates only two necessary partitions automatically for this particular query.

The automatic and dynamic partitioning also constitutes a form of semantic query optimization. In MDQ5, the mediator uses the condition introduced in the WHERE clause to prune away sub-queries that would conflict with this condition. In MDQ6, the first sub-query can be considered as a result of coalescing three adjoining periods that would have been created by a manual partitioning. These queries are said to be semantically optimized because for the first case it avoids unnecessary access to data sources only to generate an empty answer, and for the second case it avoids repeated accesses to the data source.

The mediator is responsible for rewriting user queries. Although the main goal of this rewriting is to reconcile semantic differences, we can also let it reconcile source capability heterogeneities. This capability is demonstrated in Example 1.

Other features of COIN and the extensions will be discussed after we have presented the technical details in the next chapter.

Appendix 3.1 – Mediated SQL Query MQ3

Mediated SQL query corresponding to MDQ3.

MQ3:

```
select datecvt.date2, (yhfrankfurt.adjClose*olsen2.rate),
       (yhparis.adjClose*olsen.rate), yhnyse.adjClose
from   (select 'EUR', 'USD', rate, ratedate
        from   olsen
        where  exchanged='EUR'
        and    expressed='USD') olsen,
       (select date1, 'MM/dd/yy', date2, 'MM/dd/yyyy'
        from   datecvt
        where  format1='MM/dd/yy'
        and    format2='MM/dd/yyyy') datecvt,
       (select 'DEM', 'USD', rate, ratedate
        from   olsen
        where  exchanged='DEM'
        and    expressed='USD') olsen2,
       (select date1, 'd-MMM-yy', date2, 'MM/dd/yyyy'
        from   datecvt
        where  format1='d-MMM-yy'
        and    format2='MM/dd/yyyy') datecvt2,
       (select 'IBM', qDate, adjClose, 'Dec', '20', '1998', 'Dec', '31', '1998'
        from   yhnyse
        where  Ticker='IBM'
        and    StartMonth='Dec'
        and    StartDay='20'
        and    StartYear='1998'
        and    EndMonth='Dec'
        and    EndDay='31'
        and    EndYear='1998') yhnyse,
       (select 'IBM.PA', qDate, adjClose, 'Dec', '20', '1998', 'Dec', '31', '1998'
        from   yhparis
        where  Ticker='IBM.PA'
        and    StartMonth ='Dec'
        and    StartDay ='20'
        and    StartYear ='1998'
        and    EndMonth ='Dec'
        and    EndDay ='31'
        and    EndYear ='1998') yhparis,
       (select 'IBM.f', qDate, adjClose, 'Dec', '20', '1998', 'Dec', '31', '1998'
        from   yhfrankfurt
        where  Ticker='IBM.f'
        and    StartMonth ='Dec'
        and    StartDay ='20'
        and    StartYear ='1998'
        and    EndMonth ='Dec'
        and    EndDay ='31'
        and    EndYear ='1998') yhfrankfurt
where  datecvt2.date1 = yhnyse.qDate
and    yhnyse.qDate = yhparis.qDate
and    yhparis.qDate = yhfrankfurt.qDate
and    datecvt.date2 = datecvt2.date2
and    olsen.ratedate = datecvt.date1
and    datecvt.date1 = olsen2.ratedate
union
select datecvt3.date2, (yhfrankfurt2.adjClose*olsen3.rate),
       (yhparis2.adjClose*olsen3.rate), yhnyse2.adjClose
from   (select date1, 'MM/dd/yy', date2, 'MM/dd/yyyy'
        from   datecvt
        where  format1='MM/dd/yy'
        and    format2='MM/dd/yyyy') datecvt3,
       (select 'EUR', 'USD', rate, ratedate
        from   olsen
        where  exchanged='EUR'
        and    expressed='USD') olsen3,
       (select date1, 'd-MMM-yy', date2, 'MM/dd/yyyy'
        from   datecvt
        where  format1='d-MMM-yy'
        and    format2='MM/dd/yyyy') datecvt4,
       (select 'IBM', qDate, adjClose, 'Jan', '1', '1999', 'Jan', '10', '1999'
        from   yhnyse
        where  Ticker='IBM'
        and    StartMonth='Jan'
        and    StartDay='1'
        and    StartYear='1999'
        and    EndMonth='Jan'
        and    EndDay='10'
        and    EndYear='1999') yhnyse2,
       (select 'IBM.PA', qDate, adjClose, 'Jan', '1', '1999', 'Jan', '10', '1999'
        from   yhparis
        where  Ticker='IBM.PA'
        and    StartMonth ='Jan'
        and    StartDay ='1'
        and    StartYear ='1999'
        and    EndMonth ='Jan'
        and    EndDay ='10'
        and    EndYear ='1999') yhparis2,
       (select 'IBM.f', qDate, adjClose, 'Jan', '1', '1999', 'Jan', '10', '1999'
        from   yhfrankfurt
        where  Ticker='IBM.f'
        and    StartMonth ='Jan'
        and    StartDay ='1'
        and    StartYear ='1999'
        and    EndMonth ='Jan'
        and    EndDay ='10'
        and    EndYear ='1999') yhfrankfurt2
where  datecvt4.date1 = yhnyse2.qDate
and    yhnyse2.qDate = yhparis2.qDate
and    yhparis2.qDate = yhfrankfurt2.qDate
and    datecvt3.date2 = datecvt4.date2
and    datecvt3.date1 = olsen3.ratedate
```

"There is nothing more practical than a good theory."
– Kurt Lewin

Chapter 4

Representation and Reasoning with Temporal Semantic Heterogeneity

This chapter provides the detail about the techniques that underpin the COIN capabilities described in the previous chapter. Since the new representation and reasoning capabilities are developed as extensions to the existing COIN, most of the formal descriptions of COIN in Goh (1997) and Firat (2003) are still valid. To make the Thesis as self-contained as possible, we present the theoretical foundation of COIN in the first half of the chapter, followed by a formal presentation of the extensions for temporal representation and reasoning.

In presenting the material of this chapter, we assume the readers have background knowledge in several areas related to logic programming. The following pointers should be helpful if reader needs further information about several useful topics. We suggest Lloyd (1987) for the theories of logic programming, Bratko (2001) for introductory Prolog and its applications in AI, Sterling and Shapiro (1994) for advanced Prolog programming, and Frühwirth and Abdennadher (2003) for constraint logic programming and its implementation using constraint handling rules (CHR) (Frühwirth, 1998).

The rest of the chapter is organized as follows. In Section 4.1, we briefly introduce the effort of formalizing context and discuss how it is related to the COIN approach. In Section 4.2, we describe the representation formalism of COIN. Several important notions that are missing in prior COIN work are provided in this section. In Section 4.3, we give background information on Abductive Constraint Logic Programming (ACLP) (Kakas et al., 2000) and CHR (Frühwirth, 1998), and then we describe the COIN reasoning mechanism. In Sections 4.4 and 4.5, we present the extensions to the representation formalism and the reasoning mechanism respectively. In Section 4.6, we discuss several issues that need further investigation. A brief summary is given in the last section.

4.1 Context and Semantic Interoperability

According to MaCarthy (1987), the truth or falsity of a statement can only be determined by referencing a given context. For example, *Holmes is a detective* is true in the context of *Sherlock Holmes stories*, whereas *Holmes is a Supreme Court Justice* is true in the context of *U.S. legal* *history*. These can be formalized using a logic of context that contains the following basic relations (McCarthy and Buvač, 1997):

$$c_0: \ ist(c,p)$$
$$c_0: \ value(c,e)$$

The first relation means that the p is true in context c, this statement itself being asserted in an outer context c_0. While contexts can be nested infinitely, most practical problems often need only one layer of outer context. Here, p can be a variable-free *proposition* or a first order logic (FOL) *formula* with quantified variables. We refer to p as a *statement* when this distinction is not necessary. The second relation is a *function* that returns the value of term e in context c.

Lifting axioms are used to describe the relationship between terms and formulas in different contexts. Guha (1995) provides several kinds of lifting axioms, most of which have one of the following forms:

$$c_0: \ ist(c_1,p_1) \leftrightarrow ist(c_2,p_2)$$
$$c_0: \ ist(c_1,p_1) \rightarrow ist(c_2,p_2)$$

which establishe the logical equivalence or logical implication relationship between statements p_1 and p_2 in contexts c_1 and c_2, respectively (e.g., the second lifting axiom states "if p_1 is true in c_1, then p_2 is true in c_2").

This notion of context is useful for understanding semantic heterogeneities between sources and receivers. The tuples in relational sources can be seen as logical statements, asserting that certain relationships exist between things represented by pieces of data. However, even if these statements are true in the context associated with the source, they may be false in the context of a given receiver. *Lifting*, or *transformation*, or *conversion*, is necessary to derive true statements for the receiver from statements true in the source contexts. For example, the sample tuples in the YHFrankfurt source in Figure 3.2 are equivalent to the following two statements:

$$yhfrankfurt(\text{"IBM.f"}, \text{"4 - JAN - 99"}, 75.05).$$
$$yhfrankfurt(\text{"IBM.f"}, \text{"30 - DEC - 98"}, 149.14).$$

The statements are true in context *c_germany*; in the logic of context, they are written as:

$$c_0: \ ist(c\_germany, yhfrankfurt(\text{"IBM.f"}, \text{"4 - JAN - 99"}, 75.05)).$$
$$c_0: \ ist(c\_germany, yhfrankfurt(\text{"IBM.f"}, \text{"30 - DEC - 98"}, 149.14)).$$

(4.1)

However, the two statements are not true in the receiver context *c_usa*; they have to be transformed, to the following statements:

$$yhfrankfurt(\text{"IBM.f"}, \text{"01/04/1999"}, 88.61).$$
$$yhfrankfurt(\text{"IBM.f"}, \text{"12/30/1998"}, 89.27).$$

which become true in the receiver context, i.e.,

$$c_0: \ ist(c\_usa, yhfrankfurt(\text{"IBM.f"}, \text{"01/04/1999"}, 88.61)).$$
$$c_0: \ ist(c\_usa, yhfrankfurt(\text{"IBM.f"}, \text{"12/30/1998"}, 89.27)).$$

(4.2)

The primary objective of semantic interoperability technology is to provide a transformation service so that statements true in one context can be correctly restated in another context. The outer context c_0 is the context of the service, i.e., the *integration context*. Descriptions about inner contexts (e.g., "In *c_usa*, currency of monetary value is USD") and definitions on conversions between inner contexts are all stated in this integration context c_0.

The transformation can be declaratively defined using lifting axioms in logic of context (McCarthy and Buvač, 1997; Guha, 1995). For example, it is possible to specify the following lifting axioms to enable the transformation from (4.1) to (4.2):

$$
\begin{aligned}
&c_0 : (\forall Stock, Date, Price)\\
&\quad ist(c\_germany, yhfrankfurt(Stock, Date, Price) \wedge Date \leq \text{"31 - Dec - 1998"}) \rightarrow\\
&\qquad\qquad ist(c\_usa, yhfrankfurt(Stock,\\
&\qquad\qquad\qquad f(Date, \text{"d - MMM - yy"}, \text{"MM/dd/yyyy"}),\\
&\qquad\qquad\qquad g(Price, Date, \text{"}DEM\text{"}, \text{"}USD\text{"})).
\end{aligned}
$$

$$
\begin{aligned}
&c_0 : (\forall Stock, Date, Price)\\
&\quad ist(c\_germany, yhfrankfurt(Stock, Date, Price) \wedge Date \geq \text{"1 - JAN - 1999"}) \rightarrow\\
&\qquad\qquad ist(c\_usa, yhfrankfurt(Stock,\\
&\qquad\qquad\qquad f(Date, \text{"d - MMM - yy"}, \text{"MM/dd/yyyy"}),\\
&\qquad\qquad\qquad g(Price, Date, \text{"}EUR\text{"}, \text{"}USD\text{"})).
\end{aligned}
$$

where f and g are conversion functions that can be defined using the logic or implemented in an external program to return a value when all the parameters become instantiated. Notice that the axioms are expressed over *predicates* with universally quantified variables. For semantic interoperability, this is equivalent to expressions over the *relations* in data sources.

Lifting axioms are often specified pair-wise for all contexts, thus the number of lifting axioms in this approach grows rapidly with the increase of the number of different contexts. COIN takes a different strategy to overcome this problem. The technical details and the merits of the COIN strategy will be covered in the rest of the chapter and the next chapter. Here, we only provide a brief summary about how it differs from the strategy that relies on the expressiveness of logic of context. The differences are mainly in how context is modeled and described, and how context is associated with statements.

There has been no consensus on what *context* is. Particularly, McCarthy has insisted on not giving context a definition. In McCarthy and Buvač (1997), a *poor* context can be fully described, while a *rich* context cannot be fully described at all. In Guha (1995), contexts are regarded as rich objects that capture *everything* that is necessary to make a set of statements true. For example, he says:

> *"In the formula ist(NaiveMoneyMt A1), the context denoted by the symbol NaiveMoneyMt is supposed to capture everything that is not in A1 that is required to make A1 a meaningful statement representing what it is intended to state. This includes assumptions made by A1 about the domain, assumptions about the applicability of A1, how A1 might relate to statements in other contexts, etc."*

In his formulation, contexts are described using rules that specify the scope of a theory, certain properties common to a class of contexts, and lifting rules that relate statements in one context to those in another. In other words, a context is described indirectly by all the statements that are true in the context and directly by the rules that relate these statements to those in the other contexts. Such formulations are often very expressive but at a price of sacrificing computational tractability. Even with certain restrictions such as assuming the same symbols denote the same objects across contexts in Buvač (1996) and disallowing context nesting and quantification over contexts in Nayak (1994), the resulting logics are still only semi-decidable (Guha et al., 2004). In addition, there will be a proliferation of lifting axioms when there are a large number of contexts.

In COIN, contexts are described using the vocabulary from a shared ontology, namely by assigning values to a set of modifiers present in the ontology. We will explain the concepts of ontology and modifier later in the chapter. Although using just modifiers for context description has limited expressiveness[28], it provides a *systematic* and *structured* way of describing contexts. We can exploit the structure by devising more efficient reasoning algorithms to replace the generic inference mechanism built into the various logics of contexts. Another benefit is that the number of lifting axioms can be significantly reduced.

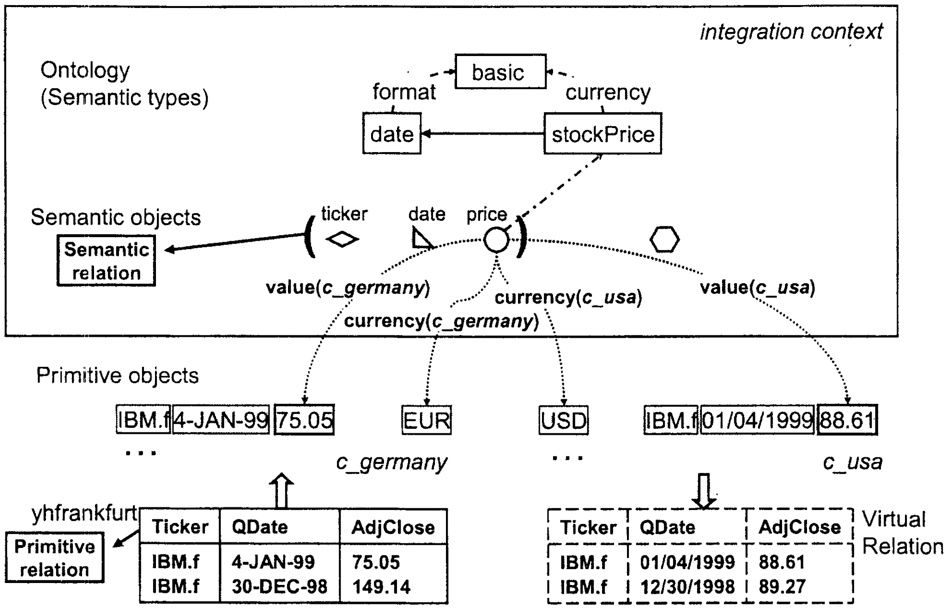
In Guha (1995), lifting axioms are specified for individual predicates, functions, and formulas so that other formulas involving these specified components can be *lifted* automatically through composition supported by the inference mechanism. In COIN, we go even further by allowing the association of contexts with individual terms, which corresponds to the arguments in a predicate, and the arguments correspond to attributes (i.e., columns) of a relation (i.e., table) in the relational model. An example of associating attributes to contexts is given in Figure 4.2 of Section 4.2. The finer granularity in COIN enhances context sharing and reduces the number of lifting axioms to be specified.

Next, we describe the COIN framework, which consists of a representation formalism and a reasoning mechanism for describing contexts and using the descriptions to automatically generate instructions for transforming data from source contexts to receiver context.

4.2 The Representation Formalism of COIN

The purpose of knowledge representation in COIN is to have a formal way of describing various assumptions about data and what each data element in the sources and receivers denote. Referring to the statements in (4.1) above for the record *yhfranfurt*("IBM.f", "4-JAN-99", 75.05), the fact that the value "75.05" denotes a *stock price* given in the *currency* EUR needs to be somehow represented. A receiver may need the same *stock price* in a different *currency*, say, USD. Once such knowledge is represented using a certain formalism, it can be manipulated by the mediator (i.e. software component that implements the reasoning mechanism of the COIN framework) to automatically convert data values in the source to the values desired by the receiver, e.g., deriving "88.61"as in (4.2) for the receiver from "75.05" in the source. The representation formalism used by COIN is based on an object-oriented data model. In the model, we introduce semantically "rich" data types, called *semantic types* (e.g., stockPrice) to distinguish them from *primitive types* (e.g., integer, string). The instances of these types are respectively called *semantic objects* and *primitive objects*; the relations of these objects are respectively called *semantic relations* and *primitive relations*. The key concepts of this data model are illustrated in Figure 4.1 and explained below.

[28] We have not encountered any problem in various prototype applications we developed for testing the approach. If more expressive power becomes necessary, we should introduce richer constructs in ontology or look into other alternatives.



Ticker	QDate	AdjClose
IBM.f	4-JAN-99	75.05
IBM.f	30-DEC-98	149.14

Ticker	QDate	AdjClose
IBM.f	01/04/1999	88.61
IBM.f	12/30/1998	89.27

Figure 4.1 Object-oriented data model in COIN

Each semantic type can have various aspects that may be different in different contexts. These aspects are described for each context. For example, *currency* is an aspect of *stockPrice*, and for each context we describe what currency is used. To access these descriptions, we construct semantic objects that correspond to the data in the sources and receivers. For example, corresponding to data value "75.05" in the source (and its corresponding context *c_germany*), there is a semantic object (shown as a circle in Figure 4.1) that is an instance of *stockPrice* semantic type. Since this object is created from a value in the context of *c_germany*, we call *c_germany* the *original* (or *originating*) context of the object. The context descriptions are accessed through methods (shown as labeled arrows pointing downward in Figure 4.1) of the semantic object, e.g., through *currency* method, the mediator can determine that the *currency* of the semantic object is "EUR" in *c_germany* and "USD" in *c_usa*. By applying appropriate currency conversion, the value "88.61" in *c_usa* can be derived.

Each semantic object is internally identified by a unique *object-id* (OID) constructed using *Skolemization*, which we present in Section 4.2.3. As seen in the example, a semantic object can have different values in different contexts. The *value* of a semantic object in a given context is returned by calling the *value* method. To avoid confusion, we sometimes call the "value" returned by the *value* method *primitive value*, which is in fact a primitive object. For a primitive object, we do not distinguish between its OID and its *value*.

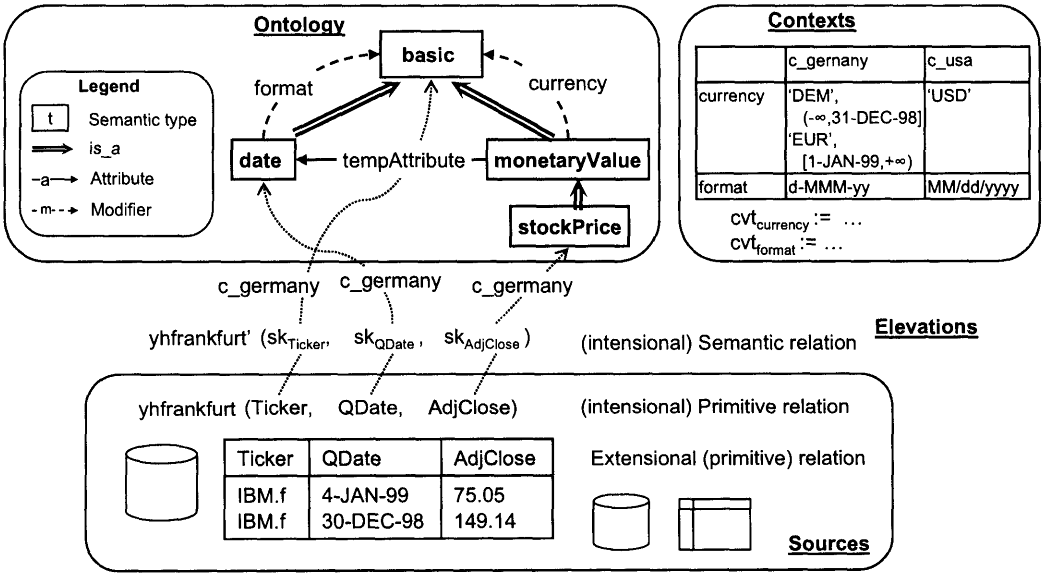
In the above discussion, for explications purposes we used data instances (e.g., "75.05") in data sources to illustrate the distinction between a semantic object and a primitive object. In the COIN approach, we actually do not create a semantic object for each data instance in the source. Instead, we create a semantic object for each attribute in the source relation, e.g., there will be *one* semantic object for "AdjClose" attribute. Such objects are meta-objects. The *value* method of a meta-object does not return data instances, instead, it returns either the attribute (if no conversion is needed) or a series of conversions applied to the attribute (e.g., currency conversion for converting "AdjClose" attribute from EUR to USD) in the form of a query (which we call mediated query). Data instances are obtained when the query is executed. However, there are cases where we do create a semantic object based on a particular data instance. These points will become clear when we discuss *Skolemization* in Section 4.2.3.

4.2.1 The Concepts of Knowledge Representation

Loosely speaking, COIN implements the objected-oriented data model using the following four components:

- an *ontology*, also called a *domain model*, that provides the *semantic types* in the application domain and a vocabulary for describing contexts;
- a *context* set that has a collection of context descriptions using the vocabulary provided by the ontology. The context descriptions capture all *aspects* of data interpretation that may vary and specify conversion rules for each of the aspects;
- a *source* set that specifies the source schemas by identifying the relations and their integrity constraints. These relations are called *primitive relations*; and
- an *elevation* set that creates a *semantic relation* for each primitive relation by mapping each attribute of the primitive relation to a corresponding type in the ontology and associating each attribute with a context.

We will give detailed explanations of these components in the rest of the section. After providing an informal description of the main concepts, we present the formal specification of each component.
We will use Figure 4.2 to explain the four components. The Figure consists of a graphical representation of the ontology, certain informal context descriptions, one data source from the Yahoo historical price example in Chapter 3, and the mappings between the relational attributes and the types in the ontology.


Ontology
basic
format
currency
date
tempAttribute
monetaryValue
stockPrice
Legend
t Semantic type
is_a
-a→ Attribute
-m-→ Modifier
Contexts
c_germany
c_germany
c_germany
yhfrankfurt' (sk_{Ticker}, sk_{QDate}, $sk_{AdjClose}$)
(intensional) Semantic relation
Elevations
yhfrankfurt (Ticker, QDate, AdjClose)
(intensional) Primitive relation
Extensional (primitive) relation
Sources

	c_gernany	c_usa
currency	'DEM', (-∞,31-DEC-98] 'EUR', [1-JAN-99,+∞)	'USD'
format	d-MMM-yy	MM/dd/yyyy

$cvt_{currency}$:= ...
cvt_{format} := ...

Ticker	QDate	AdjClose
IBM.f	4-JAN-99	75.05
IBM.f	30-DEC-98	149.14

Figure 4.2 Components of Knowledge Representation in COIN

Ontology

From the conceptual modeling perspective, the ontology consists of all major concepts in the application domain and the relationships between the concepts. These are high level concepts, the precise interpretation of which needs to be further specified by each source and receiver according to their respective contexts. Thus the terms representing the concepts still have certain ambiguities, which are disambiguated in context descriptions. This approach simplifies ontology development by requiring minimal ontological commitment on a much smaller set of terms. For example, the sources and the receivers only need to agree that there is general notion of *monetary value*, having their freedom of choosing the *currency*, the *scale factor*, and any other aspects that together disambiguate the general notion. A further discussion about the benefit of this approach can be found in Chapter 5.

We distinguish three kinds of relationship: *is_a*, *attribute*, and *modifier*. The *is_a* relationship indicates that a concept is more specific (or conversely more general) than another, and is shown with a double shafted arrow in Figure 4.2. For example, stock price (*stockPrice*) is a more specific concept than monetary value (*monetaryValue*). Since all concepts have a concrete representation, we use *basic* for the concrete representation and all the other concepts are connected to it either directly or via the transitive closure of *is_a* relation.

A concept can be an *attribute* of another concept, e.g., the *date* concept is an attribute (*tempAttribute*) of the *monetaryValue* concept. Similar to relations in Entity-Relation modeling where a relationship (also called a role) can be described in both directions (e.g., the relationship between a person and a company can be "works_for" or "employs"), *monetaryValue* can be the *dateFor* attribute of *date*, although we did not include this attribute in the ontology. Attributes often reflect certain structural relationships between data elements in the sources and provide a mechanism of accessing them for inference purposes.

A *modifier* is a special kind of attribute whose purpose is to capture one aspect of the interpretation of a concept that may vary in the sources and receivers. We include two modifiers in the sample ontology in Figure 4.2: *currency* modifier for *monetaryValue* concept, indicating that a monetary value may be in different currencies;*format* modifier for *date* concept, indicating that there may be different representation formats. Each concept can have more than one modifier. For example, in addition to *currency*, we could also include a *scaleFactor* modifer for *monetaryValue* concept to indicate that monetary values can be expressed using different scale factors. Using these modifiers we can explicitly capture those assumptions that are implicitly made by the sources and the receivers. We do so by assigning values to the modifiers for each context. These assignments are part of the context descriptions and are used to disambiguate the high level concepts in the ontology.

From an object-oriented modeling point of view, the concepts in the ontology are *classes*, which we call *semantic types* (or *types* for short). Thus, the ontology provides a typing system that organizes the types in a generalization hierarchy with provision of inheritance. Both *structural* inheritance and *behavioral* inheritance are supported. Through structural inheritance, a type inherits all the attributes and modifiers declared for its parent types in the *is_a* relation. Through behavioral inheritance, a type also inherits the value assignments and conversion specifications of the parent types with optional overriding, where a type has value assignments different from those of its parent types. The attributes and modifiers are equivalent to the properties of an object, with the attributes corresponding to the properties explicitly present in the data sources and the modifiers corresponding to the properties implicitly assumed by the sources and receivers.

As mentioned earlier, instances of the semantic types are *semantic objects*. When at least one of the modifiers, including the inherited ones[29], of a semantic object has different values in different contexts, the semantic object has different values in these contexts. Conversely, if all modifiers of a semantic object have the same values in different contexts, the object has the same value in these contexts. If a semantic object has no modifier, it has the same value in all contexts.

Context

Context descriptions disambiguate the concepts in the ontology by explicitly capturing the implicit assumptions in the sources and the receivers. These descriptions come in two parts. (1) The first part uses declarative rules, i.e., axioms, to assign values to modifiers or specify how modifier values can be obtained dynamically through accessing other semantic objects. A context is a collection of value assignments or specifications of all modifiers in the ontology and is referenced by a unique identifier. In Figure 4.2, two contexts, *c_germany* and *c_usa*, are shown with the value assignments for the two modifiers in the ontology. (2) The second part of context description is a collection of conversion rules that specify, for each modifier, how to convert the value of a semantic object from one context to another that has a different modifier value. These conversion rules are used for the same purposes as McCarthy's lifting axioms, but they are much finer grained and more structured than lifting axioms. The rules are specified for each modifier and are called component conversion rules; when the data elements of interest involve multiple semantic types, each potentially having multiple modifiers, all relevant component rules are identified to compose a composite conversion. As discussed in Chapter 5, this mechanism significantly reduces the number of conversion rules to be manually specified and maintained.

Source

Sources in COIN are treated uniformly as relational sources. Without loss of generality, we assume the relations in all sources have unique names; this can be ensured by prefixing each relation with a unique identifier maintained by a name space mechanism. The sources are represented by a collection of predicates (with non-ground arguments) that correspond to the relational schemas and a set of rules for the integrity constraints in the sources. We call the predicates intensional primitive relations, or *primitive relations* for short, because their arguments correspond to the primitive values in the sources. For example, *yhfrankfurt(Ticker, QDate, AdjClose)* is a primitive relation in Yahoo Frankfurt data source. The actual data records in the source are called the *extensional relations*. Integrity constraints are usually key constraints, foreign key constraints, and other constraints that are useful for query pruning and optimization.

[29] Inheritance is an intrinsic characteristic of object orientation. In the ensuing discussion, unless noted otherwise, the attributes and modifiers of an object always include those inherited from parent types.

Elevation

The data elements in each source are meaningful only in their own contexts. To make them also meaningful in other contexts, we need to identify necessary conversions and generate the instructions on how to perform these conversions. This is accomplished by mapping these data elements to the semantic types and associating them with corresponding context descriptions. These mappings and associations are generated using a collection of elevation axioms that create a semantic relation for each primitive relation. In a semantic relation, each argument is a semantic object, which is created using a structure that indicates (1) the attribute in the primitive relation, (2) its corresponding semantic type, and (3) its context. The intuition of elevation can be explained using the links shown in Figure 4.2. For example, by indicating that "AdjClose" attribute is of *stockPrice* type and is in *c_germany* context, the subsequent reasoning mechanism can determine the modifiers and their value assignments, and incorporate appropriate conversion rules to generate the instructions for transforming the values to conform to the receiver context. Therefore, elevation axioms play a vital role of enriching primitive data elements in the sources with the explicit data semantics captured by the ontology and context descriptions.

4.2.2 The COIN Data Model

The knowledge representation concepts introduced above are formalized with a logical object-oriented data model. It is logical because it uses mathematical logic as a way of representing knowledge and expressing operations on various constructs of knowledge. It is also object-oriented because it supports the object-orientation features discussed earlier.

The language of the COIN data model is a subset of F(rame)-logic (Kifer et al., 1995), which is a first order logic (FOL) with syntactic sugar for object orientation. We choose this formalism because of its support for inheritance. Semantic types can be expressed using type declarations. Attributes, modifiers, and the conversion rules for modifiers are expressed as methods for semantic types.

COIN does not use the query answering and deductive inference mechanism of the F-logic language; instead, we take advantage of its equivalence to FOL by translating F-logic expressions into FOL expressions, and then implement inheritance and query answering in Prolog. This approach combines the strengths from the two language paradigms: the succinct syntax of F-logic to support object-orientation and the industrial strength programming platform of Prolog to support prototyping and implementation. In addition, F-logic was not implemented when the first COIN prototype was developed. Although there have been two implementations of F-logic[30] since then, they only have limited support for constraint handling. As the implementation improves, we will reevaluate the feasibility of using F-logic directly in the future.

In the rest of this sub-section, we describe the COIN data model in terms of its language. A number of important concepts that are missing or not clear in Goh (1997) and Firat (2003) are provided. In the next sub-section, we introduce the correspondence between the COIN constructs and a set of FOL predicates.

[30] Currently there are two implementations of F-Logic: FLORA-2 (http://flora.sourceforge.net/) and FLORID (http://www.informatik.uni-freiburg.de/~dbis/florid/). Both lack integration with constraint solvers essential to the Abductive Constraint Logic Programming techniques (discussed in Section 4.3) that we use in COIN.

The COIN Language

The alphabet of the language consists of the following:

- a set of *type* symbols to represent semantic types; *basic* is a reserved semantic type; $\tau_\perp$ represents all primitive data types;
- a set of *constant* symbols for representing the OIDs of primitive objects and a set of *variable* symbols; recall that the OID and the value of a primitive object are the same;
- a set of *function* symbols and *predicate* symbols; *srcContext* and *srcValue* are two reserved functions;
- a set of *method* symbols corresponding to attributes and modifiers[31]; *cvt* is a built-in method for value conversions and *value* is built-in method to derive the primitive value of a semantic object in a given context;
- the usual logical *connectives*, *quantifiers*, and *auxiliary* symbols, e.g., $\wedge, \forall, \vee, \exists, \neg, (,), [,], :, ::, \rightarrow, \leftarrow, \Rightarrow$, etc. $\rightarrow$ is a domain mapping symbol; $\leftarrow$ is a logical implication symbol; $\mapsto$ and ";" are used for *if-then-else* conditional implication, e.g., $A \mapsto X; Y$ means if *A then X, else Y*; and
- a set of *context symbols* , of the distinguished object type called *ctx*.

A *term* is either a constant, a variable, or a function of the form $f(t_1, \ldots, t_n)$, where f is a function symbol and $t_1, \ldots, t_n$ are terms. The OIDs for semantic objects are Skolem functions which we describe later. A predicate, function, or method is said to be *n-ary* if it expects *n* arguments.

In the following discussion, we use τ , with or without certain super- or sub-script, as a type symbol; it can be the special type *basic*. Similarly, we use *t*, with or without certain super- or sub-script, as a term symbol.

Definition 4.1 A *declaration* is defined as follows:

- $\tau :: \tau'$ is a *type declaration*; we say that τ is a *subtype* of τ', and conversely, that τ' is a *supertype* of τ . For any type τ'' such that $\tau' :: \tau''$, τ is also a subtype of τ''; this is true recursively. *basic* is supertype of all the other types.

  Example: *stockPrice::monetaryValue* is a declaration for the *is_a* relationship shown in Figure 4.2

- $t : \tau$ is a semantic *object declaration*; we say that t is an instance of type τ . If τ' is a supertype of τ , then t is said to be of *inherited* type τ'. Similarly, $t_\perp : \tau_\perp$ is a *primitive object declaration.*

  Example: *X:stockPrice* declares that variable *X* represents a object of type *stockPrice.*

[31] An attribute (or modifier) of a semantic object is another semantic object, which is returned when the attribute (or modifier) method is called.

- if p is predicate symbol, then $p(\tau_1, \ldots, \tau_n)$ is a *predicate declaration*; the signature of predicate p is $\tau_1 \times \cdots \times \tau_n$; a predicate with all arguments being primitive type can be declared similarly.

  Example: *yhfrankfurt*(*basic*, *date*, *stockPrice*) is a declaration for *yhfrankfurt* relation.

- if a is an attribute symbol, then $\tau[a \Rightarrow \tau']$ is an *attribute declaration*, which means that semantic type τ has attribute a; the signature of the attribute is $\tau \to \tau'$.

  Example: $monetaryValue[tempAttribute \Rightarrow date]$ declares the *tempAttribute* attribute shown in Figure 4.2.

- if m is a modifier, then $\tau[m@ctx \Rightarrow \tau']$ is a *modifier declaration*. We say that m is a modifier of semantic type τ and has a signature of $\tau \to \tau'$; the parameter of m is of type *ctx*.

  Example: $monetaryValue[currrency \Rightarrow basic]$ declares *currency* modifier.

- $\tau[cvt@m, ctx, \tau_1 \mid \tau_\perp, \tau_2 \mid \tau_\perp, \tau_\perp \Rightarrow \tau_\perp]$ is a *component conversion declaration* for modifier m; symbol "|" indicates that argument can be one and only one of the types separated by "|"; the signature of the component conversion is $\tau \times \tau_\perp \to \tau_\perp$.

  Example: $monetaryValue[cvt@currency, ctx, string, string, float \Rightarrow float]$ declares the component conversion for *currency* modifier of semantic type *monetaryValue.*

- $\tau[cvt@ctx, \tau_\perp \Rightarrow \tau_\perp]$ is a *composite conversion declaration*; the signature of the composite conversion is $\tau \times \tau_\perp \to \tau_\perp$.

  Example: $monetaryValue[cvt@ctx, float \Rightarrow float]$ declares the composite conversion for semantic type *monetaryValue.*

- $\tau[value@ctx \Rightarrow \tau_\perp]$ is a *value* declaration; its signature is $\tau \to \tau_\perp$.

  Example: $monetaryValue[value@ctx \Rightarrow float]$ declares the *value* method for semantic type *monetaryValue.*

- $srcContext(\tau) \Rightarrow ctx$ declares the *srcContext* function, whose signature is $\tau \to ctx$; $srcValue(\tau) \Rightarrow \tau_\perp$ declares the *srcValue* function, whose signature is $\tau \to \tau_\perp$.

  Example: $srcContext(monetaryValue) \Rightarrow ctx$ declares the *scrContext* function for semantic type *monetaryValue*; similarly $srcValue(monetaryValue) \Rightarrow float$ declares the *srcValue* function for semantic type *monetaryValue*.

Declarations for attributes, modifiers, conversions, and the built-in method value are collectively referred to as *method declarations.* ■

In the following definition, we will not include examples. Plenty of examples about this definition can be found in Sections 4.4 and 4.5.

Definition 4.2 An atom is defined as follows:

- if p is an n-ary predicate symbol with signature of $\tau_1 \times \cdots \times \tau_n$ and $t_1, \cdots, t_n$ are of (inherited) types $\tau_1, \cdots, \tau_n$ respectively, then $p(t_1, \ldots, t_n)$ is a predicate atom; similarly for a predicate with primitive arguments.
- if a is an attribute symbol with signature $\tau \rightarrow \tau'$, t, t' are of (inherited) types τ, τ' respectively, then $t[a \rightarrow t']$ is an attribute atom. Similarly, if m is a modifier symbol and c is a context symbol, then $t[m@c \rightarrow t']$ is a modifier atom.
- if m is a modifier symbol, its component conversion has signature $\tau \times \tau_\perp \rightarrow \tau_\perp$, t is of (inherited) type τ, t_{ms} and t_{mr} are of type τ_m, $t_{\perp 1}$ and $t_{\perp 2}$ are of type $\tau_\perp$, and t_c is a context term (i.e., it can be a constant or a variable of type *ctx*), then $t[cvt@m, t_c, t_{ms}, t_{mr}, t_{\perp 1} \rightarrow t_{\perp 2}]$ is a component conversion atom. Similarly, $t[cvt@t_c, t_{\perp 1} \rightarrow t_{\perp 2}]$ is a composite conversion atom.
- if c_2 is a context symbol, t and $t_{\perp 2}$ are of type τ and $\tau_\perp$ respectively, then $t[value@c_2 \rightarrow t_{\perp 2}]$ is a *value* atom.

The atoms corresponding to attributes, modifiers, and built-in methods *cvt* and *value* are referred to collectively as *method atoms*. ■

For notational convenience, atoms can be combined to form *molecules* using the following syntactic sugar:

- $t[m_1 \rightarrow t_1; \cdots; m_k \rightarrow t_k]$ as a shorthand for the conjunction $t[m_1 \rightarrow t_1] \wedge \cdots \wedge t[m_k \rightarrow t_k]$;
- $t[m \rightarrow t_1[m_1 \rightarrow t_2]]$ as a shorthand for $t[m \rightarrow t_1] \wedge t_1[m_1 \rightarrow t_2]$; when t_1 is not referenced elsewhere, we can replace it with _ as in Prolog: $t[m \rightarrow \_[m_1 \rightarrow t_2]]$; and
- $t : \tau[m \rightarrow t']$ as a shorthand for $t : \tau \wedge t[m \rightarrow t']$.

Well formed formulas (*wff_s*) are defined inductively as follows:

- an atom is a formula;
- if ϕ and φ are formulas, so are $\neg\phi$, $\phi \wedge \varphi$, and $\phi \vee \varphi$; and
- if ϕ is a formula and X is a variable occurring in ϕ, then both ($\forall X\ \phi$) and ($\exists X\ \phi$) are formulas.

As suggested in Goh (1997), it is customary to restrict wff_s to Horn clauses to have better computational tractability. A Horn clause has the following form:

$$\Gamma \vdash \quad A \leftarrow B_1, \cdots, B_n.$$

where A is an atom, and is called the *head* of the clause; $B_1, \cdots, B_n$ is a conjunction of atoms and is called the *body* of the clause. The body can be empty; when it is empty, the implication symbol can be omitted. Γ is a collection of type declarations. When A is a method atom, all oid-terms in A must be declared in Γ; otherwise, Γ may be omitted.

The *value* and *cvt* Methods

As discussed earlier, a semantic object can have different primitive values in different contexts. The *value* and *cvt* methods provide the means of obtaining these primitive values of a semantic object. These two methods are important in the COIN approach, but they are not well defined in either Goh (1997) or Firat (2003). Below we provide definitions for them.

As mentioned in the beginning of Section 4.2 and will be seen in Section 4.2.3, a semantic object is created from a relational attribute a given source and is associated with a particular context. We call this context the *original context* of the object.

The *value* method of a semantic object derives the primitive value of the object in a given context. This is a built-in method of COIN and is evaluated internally. If the type of the object is *basic* or any other semantic type that does not have any modifier, it returns the primitive value of the object in its original context:

$$\begin{array}{l} t : basic \vdash \;\; t[value@t_c \to t_\perp] \leftarrow t_\perp = srcValue(t). \\ t : \tau \vdash \;\; t[value@t_c \to t_\perp] \leftarrow \neg t[m@t_c \to t_a], t_\perp = srcValue(t). \end{array} \qquad (4.3)$$

Otherwise, *value* method invokes the composite conversion of the object to generate the instructions on how to derive the primitive value t_k in (receiver) context c_k from the primitive value t_0 in original (source) context *c*:

$$t : \tau \vdash \;\; t[value@c_k \to t_k] \leftarrow t_0 = srcValue(t), \;\; t[cvt@c_k, t_0 \to t_k]. \qquad (4.4)$$

From now on, we drop the subscript $_\perp$ for terms representing primitive objects because they can be inferred from the signatures of *value* and *cvt* methods. For example, both t_k and t_0 in formula (4.4) are primitive types.

In COIN, the composite conversion of semantic object *t* is not defined *a priori*; instead, it is composed at query mediation time by the mediator that consecutively invokes the component conversions of all modifiers (including the inherited modifiers) of *t*. If the original context of *t* (i.e., the source context) is c, and *t* has *k* modifiers m_1, ..., m_k, the composite conversion can be obtained according to the following formula:

$$\begin{array}{l} t : \tau \vdash \\ \quad t[cvt@c_k, t_0 \to t_k] \leftarrow \\ \qquad t[m_1@c \to \_[value@c_k \to t_{1s}]], t[m_1@c_k \to \_[value@c_k \to t_{1r}]], \\ \qquad\quad (t_{1s} == t_{1r} \mapsto t_1 = t_0; t[cvt@m_1, c_k, t_{1s}, t_{1r}, t_0 \to t_1]), \\ \qquad t[m_2@c \to \_[value@c_k \to t_{2s}]], t[m_2@c_k \to \_[value@c_k \to t_{2r}]], \\ \qquad\quad (t_{2s} == t_{2r} \mapsto t_2 = t_1; t[cvt@m_2, c_k, t_{2s}, t_{2r}, t_1 \to t_2]), \\ \qquad \ldots, \\ \qquad t[m_k@c \to \_[value@c_k \to t_{ks}]], t[m_k@c_k \to \_[value@c_k \to t_{kr}]], \\ \qquad\quad (t_{is} == t_{ir} \mapsto t_k = t_{k-1}; t[cvt@m_k, c_k, t_{ks}, t_{kr}, t_{k-1} \to t_k]). \end{array} \qquad (4.5)$$

The head of formula (i.e., $t[cvt@c_k, t_0 \to t_k]$) indicates that given the semantic object *t* and its value t_0 in its original context, the composite conversion derives t_k as the value of *t* in context c_k. The body of the rule specifies the derivation procedure, which is roughly as follows: (1) for each modifier, obtain the value assignments in the original context of *t* and the receiver context c_k; (2) then compare modifier values: if they are the same, do not change the primitive value, otherwise, introduce a component conversion to convert the primitive value. Below we explain this procedure using the two lines that contain modifier m_2 in (4.5):

- the first molecule (i.e., $t[m_2 @ c \rightarrow \_[value @ c_k \rightarrow t_{2s}]]$) obtains the assignment for modifier m_2 in (source) context c. The assignment is an unnamed semantic object represented by "_". Then use value method to derive this object's primitive value t_{2s} in (receiver) context c_k;
- the second molecule (i.e., $t[m_2 @ c_k \rightarrow \_[value @ c_k \rightarrow t_{2r}]]$) is similar to the first one, but it obtains m_2's assignment in (receiver) context c_k, then derives its primitive value t_{2r} also in (receiver) context c_k; and
- in the second line containing m_2, it tests to see if the modifier has same values (i.e., if t_{2s} equals t_{2r}): if so, assign t_1 to t_2, otherwise, invoke the component conversion for m_2 to derive the primitive value t_2 from t_1.

After the invocation of the k^{th} component conversion, we derive semantic object t's value t_k in (receiver) context c_k.

We also provide an alternative explanation of the composite conversion. As we noted earlier, each modifier captures one aspect of interpretation that may vary across contexts. After invoking a component conversion, the semantic object is lifted to a (intermediate) context that is the same as the receiver context in terms of this aspect; by invoking the component conversions consecutively, the semantic object is lifted through a series of intermediate contexts, each having one less aspect being different from the receiver context; it reaches the receiver context in the end when no aspect is different from the receiver context.

In the explanation of (4.5) we assumed the invocation of the component conversions is in the order of their appearance in (4.5). Does the order matter? The answer to this question depends on the *orthogonality* of the modifiers, which we discuss next.

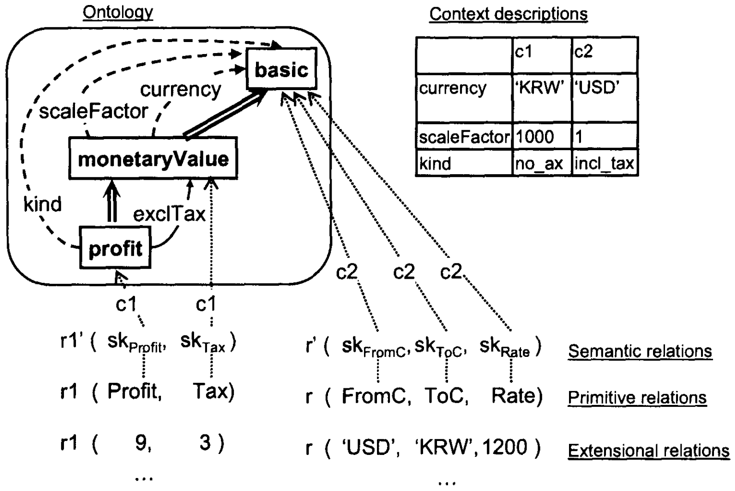
Orthogonality of Modifiers

The modifiers of the (inherited) type τ are orthogonal if the value derived from its composite conversion is not affected by the order in which the component conversions are invoked. For example, the *currency* and *scaleFactor* modifiers of type *monetaryValue* are orthogonal: we derive the same value either by converting currency first followed by scale factor adjustment or by adjusting scale factor first followed by currency conversion.

For a pair of modifiers that are not orthogonal, the component conversions need to be invoked in a particular order to ensure the correctness of the composite conversion; we call this order a *correct* order. The order often depends on how the component conversion is specified. We will use the following example to illustrate this.

Suppose a source contains profit and tax data, expressed in 1000's of South Korean Won (KRW); the profit is net profit not including taxes. A receiver needs gross profit data that includes the taxes, expressed in 1's of USD. These context descriptions are illustrated in Figure 4.3, which also shows the source r_1(*Profit*, *Tax*) and the auxiliary source r(*FromC*, *ToC*, *Rate*) for currency conversion.

Profit data is mapped to *profit* type, tax data is mapped to *monetaryValue* type, data in the auxiliary source are mapped to type *basic*.



	c1	c2
currency	'KRW'	'USD'
scaleFactor	1000	1
kind	no_ax	incl_tax

Figure 4.3 Example for Illustrating Modifier Orthogonality

Type *profit* has three modifiers: *scaleFactor*, *currency*, and *kind*, the first two of which are inherited from its supertype *monetaryValue*. The component conversions for these modifiers are given below (here we follow the logic programming convention where terms started with capital letters are variables):

$$
\begin{array}{l}
M : monetaryValue \vdash \\
\quad M[cvt@scaleFactor, C_r, M_s, M_r, U \to V] \leftarrow V = U * M_s / M_r.
\end{array}
$$

$$
\begin{array}{l}
M : monetaryValue \vdash \\
\quad M[cvt@currency, C_r, M_s, M_r, U \to V] \leftarrow \\
\quad\quad r(M_s, M_r, R), V = U * R.
\end{array}
$$

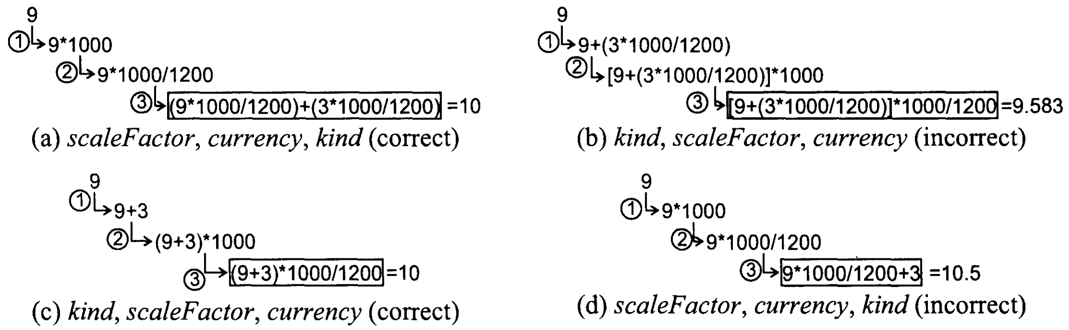
$$
\begin{array}{l}
P : profit \vdash \\
\quad P[cvt@kind, C_r, 'no\_tax', 'incl\_tax', U \to V] \leftarrow \\
\quad\quad P[exclTax \to T], \\
\quad\quad T[value@C_r \to T_p], \\
\quad\quad V = U + T_p.
\end{array}
$$

As in logic programming, variables with the same name appearing in different clauses are not related. Each of the component conversions converts from (source) context C_s to (receiver) context C_r to derive primitive value *V* from *U*. In the *conversions* for *scaleFactor* and *currency*, the modifier values in source and receiver contexts are M_s and M_r; these parameters are variables, indicating that the component conversions can convert between any arbitrary pair of contexts. The *profit* conversion can convert from any context whose *kind* modifier has a value of '*no_tax*' to any other context whose *kind* modifier has a value of '*incl_tax*'.

As we mentioned earlier, modifiers *scaleFactor* and *currency* are orthogonal. In fact, we can illustrate this using the component conversions shown above: starting with a primitive value *x*, applying *scaleFactor* conversion followed by *currency* conversion we derive $x \cdot \frac{M_s}{M_r} \cdot R$; we derive

$x \cdot R \cdot \frac{M_s}{M_r}$ when *currency* conversion is applied before *scaleFactor* conversion. The two derived values are equal.

The component conversion for *kind*, as specified above, must be invoked after those for *scaleFactor* and *currency*. The component conversion first obtains the semantic object T representing the taxes excluded from the profit data using the attribute method *exclTax*; then it calls the *value* method of T to obtain its primitive value T_p in (receiver) context C_r; and finally it adds T_p to the initial value U to arrive at value V. T is of type *monetaryValue* and has *scaleFactor* and *currency* modifiers; the component conversions of the two modifiers are invoked when the *value* method of T is called, i.e., T_p is the result of transforming the tax data in the source with scale factor adjustment and currency conversion. Adding T_p to U entails the assumption that U has been transformed for scale factor and currency before the addition takes place; otherwise T will be transformed twice for scale factor and currency, resulting in wrong data. The above data transformation sequence is shown in Figure 4.4(a) using the sample data record in the extensional relation. As shown in Figure 4.4(b), if the component conversion for *kind* is invoked first, the result will be incorrect because the tax data will be adjusted for *scaleFactor* and *currency* differences twice.



(a) *scaleFactor*, *currency*, *kind* (correct)

(b) *kind*, *scaleFactor*, *currency* (incorrect)

(c) *kind*, *scaleFactor*, *currency* (correct)

(d) *scaleFactor*, *currency*, *kind* (incorrect)

Figure 4.4 Different Sequences of Data Transformation.

An alternative specification of the component conversion for modifier *kind* is to replace C_r with C_s in the *value* method for semantic object T: the derived value T_p will be the primitive value of tax data in (source) context C_s. In this case, adding T_p to value U requires the assumption that U is the primitive value of profit data in source context C_s and the component conversion for *kind* modifier should be invoked before those for *scaleFactor* and *currency*. This sequence is illustrated in Figure 4.4(c) using the sample data. With this specification of the component conversion for *kind*, if the conversion is invoked last, the result will be incorrect because the tax data will not be adjusted for *scaleFactor* and *currency* differences. This incorrect sequence is shown in Figure 4.4(d).

When a type has non-orthogonal modifiers, the precedence of modifiers should be specified so that the component conversions can be applied in correct order. For example, with the component conversions as shown earlier, we can specify the precedence of the three modifiers as the following:

[*scaleFactor*, *currency*] $\prec$ [*kind*]

where modifiers enclosed in the same brackets are orthogonal. The above specification means that component conversion for *kind* must be applied after those for *scaleFactor* , whereas the order between *scaleFactor* and *currency* does not matter. This can be implemented as a list structure for each semantic type (using a list imposes an ordering for orthogonal modifiers, but this does not affect the correctness because any order amongst orthogonal modifiers is correct). Before applying the composite conversion procedure, modifiers are first sorted according to the list to ensure that component conversions are introduced in correct order.

4.2.3 COIN Constructs to FOL Translation

In this section, we introduce the FOL predicates corresponding to the COIN constructs in the F-logic based COIN language. These correspondences provide the foundation for implementing the COIN data model using mature Prolog programming platform; they are also helpful to readers of COIN literature by bridging the gap between the different presentations, such as those in Goh (1997) and Firat (2003). We also introduce a *Skolem* function that is used in the construction of semantic relations.

Table 4.1 summarizes these correspondences. Notice that COIN declarations are simplified: (1) type τ' in attribute and modifier declarations are not indicated in the corresponding predicates, instead, it is implied by the term t' in the respective atom definition; (2) predicate, conversion (*cvt*), and *value* declarations are omitted; and (3) object declarations are omitted and can be inferred from atom definitions. Composite conversion declarations and atom definitions are omitted; as we mentioned earlier, composite conversions are composed dynamically by the mediator.

Table 4.1 Correspondences between COIN Constructs and FOL Predicates

	COIN Construct	**FOL Predicate**	**Comment**
Type declaration	$\tau :: \tau'$	$is\_a(\tau, \tau')$	
Attribute declaration	$\tau[a \Rightarrow \tau']$	$attributes(\tau, [a_1, \cdots, a_k])$	$[a_1, \cdots, a_k]$ is a list of all attributes of type τ ; can be an empty list
Modifier declaration	$\tau[m@ctx \Rightarrow \tau']$	$modifiers(\tau, [m_1, \cdots, m_k])$	$[m_1, \cdots, m_k]$ is a list of all modifiers to type τ ; can be an empty list
Attribute atom	$t[a \rightarrow t']$	$attr(t, a, t')$	
Modifier atom	$t[m@c \rightarrow t']$	$modifier(\tau, t, m, c, t')$	
Component conversion	$t[cvt@m, t_c, t_{ms}, t_{mr}, t_1 \rightarrow t_2]$	$cvt(\tau, t, m, t_c, t_{ms}, t_1, t_{mr}, t_2)$	
value method	$t[value@c_2 \rightarrow t_2]$	$value(t, c_2, t_2)$	

As mentioned before, elevation axioms create semantic relations for primitive relations. Each argument in a semantic relation is a semantic object denoted by a *Skolem term.* In mathematical logic, *Skolemization* is a technique for eliminating existentially qualified variables, often by introducing function symbols over constants or universally quantified variables to construct *unique* terms called *Skolem* constants or *Skolem* functions. Below is the template for creating such *Skolem* terms in COIN:

$$skolem(\tau_i, A_i, c_i, i, r_l(\cdots, A_i, \cdots))$$

(4.6)

where τ_i is the semantic type of the attribute A_i, A_i is the i^{th} attribute of primitive relation r_l, c_i is the context of the attribute and can be different for different attribute. Terms constructed with this template are guaranteed to be unique because of the uniqueness of primitive relation. The following is an example of using the template to specify the semantic relation corresponding to primitive relation $r_1(Profit, Tax)$ in Figure 4.3:

$$\begin{aligned} & r_1'(skolem(profit, Profit, c_1, 1, r_1(Profit, Tax)), \\ & \quad skolem(monetaryValue, Tax, c_1, 2, r_1(Profit, Tax)) \\ & \quad) \leftarrow r_1(Profit, Tax). \end{aligned} \tag{4.7}$$

For brevity, we will refer to a semantic relation by a predicate that post-fixes the relation name and argument names of the corresponding primitive relation with the prime symbol, e.g., we use $r_1'(Profit', Tax')$ to refer to the semantic relation as defined by the above rule.

Sometimes we need to create a semantic object for a particular primitive object in a certain context. For example, later we will see that we need a semantic object for "31-DEC-98", which is a primitive object in context *c_germany*. This is done by using template (4.6), but replacing the attribute reference with the primitive object, and replacing the primitive relation with the special predicate *cste*/1, which is always evaluated true. Below is the semantic object constructed for "31-DEC-98":

$skolem(date, "31-DEC-98", c\_germany, 1, cste("31-DEC-98"))$

Sources are represented by the predicates corresponding to the primitive relations. In addition, they are further described using the following 5-ary *relation* predicate:

$relation(src\_Name, rel\_Name, purposeID, attrList, capability)$

where the first two arguments are source name and relation name, *purposeID* can be one either "*i*" or "*e*", with "*i*" indicating that the relation can appear in mediated query, and "*e*" indicating that the relation is an internally defined view not appearing in mediated query; *attrList* is a list of attributes of the relation, each element of list is a [*Attribute_name*, *Primitive_type*] pair; and *capability* is an argument indicating the capability record of the relation[32].

Rules are used to describe various constraints that hold in the source. These can be key constraint on one relation, foreign key constraints between two relations, and other constraints that are useful for query optimization. These rules are expressible using FOL *wff*s. For example, if K is the key attribute in relation $r(K, A_1, \ldots, A_n)$, the following *wff* expresses the key constraint:

$A_1=B_1, \ldots, A_n=B_n \leftarrow r(K, A_1, \ldots, A_n), r(K, B_1, \ldots, B_n).$

This is a non-clausal form rule; in our prototype system, such rules are expressed using the Constraint Handling Rules (CHR) language, which allows non-clausal formulae.

[32] A capability record indicates the operators on an attribute supported in the source. It helps the POE to generate executable query plan. See Alatovic (2002) for details. In current COIN prototype, the capability record is given in an external file. For backward compatibility, we still keep the argument, but it can be left unspecified using symbol "_" in Prolog.

4.2.4 Summary

We described the knowledge representation formalism of COIN in this section. The data model can represent the four components introduced earlier; the components are collectively referred to by the quadtuple <*O*, *C*, *E*, *S*>, where

O ::= *ontology*: a set of semantic types and the binary relations over the types: *is_a*, *attribute*, and *modifier*.
C ::= *context* descriptions: a set of rules that specify modifier values or how to derive values for modifiers and another set of rules specifying conversions for each modifier
E ::= *elevation* axioms: map each a relational attribute in the primitive relation to a type in the ontology and assign it a context
S ::= source descriptions: a set of rules describing the source schemas and constraints

4.3 The Reasoning Mechanism of COIN

As mentioned before, although the COIN approach can be understood with McCarthy's formal treatment of context and has a data model that is based on F-logic, we do not perform reasoning using the machinery in these logics. Instead, we implement a reasoning algorithm in the mediator; the structure provided by the ontology allows the algorithm to perform specialized and more efficient reasoning. In particular, the mediator automatically composes the composite conversions according to (4.5) using the component conversions of the modifiers of relevant semantic types.

The algorithm is implemented using Abductive Constraint Logic Programming (ACLP) (Kakas et al., 2000), which combines Abductive Logic Programming (ALP) and Constraint Logic Programming (CLP). Constraints are processed using solvers written in the constraint language called Constraint Handling Rules (CHR) (Frühwirth, 1998). In this section, we first provide the background on ACLP and CHR, and then we describe the mediation implementation and query answering mechanism in COIN.

4.3.1 Abductive Constraint Logic Programming (ACLP)

Abduction inference was first introduced by the philosopher Peirce (1903). It is a form of hypothetical and non-monotonic reasoning that derives possible explanations for a given observation from a set of rules. In this reasoning paradigm, a set of predicates need to be declared to be *abducible predicates*, also called *abducibles* for short. Intuitively, abducibles should be something "interesting" and can serve as explanations in a particular application domain. For example, given the following rules:

wet_grass ← *sprinkler_was_on.*

wet_grass ← *rained.*

and knowing that *sprinkler_was_on* and *rained* are abducibles, for the observation *wet_grass*, through abduction we can derive {*springler_was_on*} and {*rained*} as possible explanations. In Appendix 4.1 we show how the explanations are derived using the SLD+Abduction procedure first introduced in Cox and Pietrzykowski (1986). Abduction is non-monotonic because if later we learn that it did not rain last night, we can retract *rained* from the explanations.

Abductive Logic Programming (ALP) is the extension of logic programming to support abductive reasoning. It has been used as a problem solving technique in many AI applications, e.g., planning and scheduling, natural language understanding, database view updates (Kakas et al., 1993, Kakas et al., 2000, and references therein). The use of ALP in COIN for information integration, first proposed by Goh (1997), is a novel application of ALP. In these applications, the goals to be solved are seen as observations to be explained by abduction, e.g., the mediated query is viewed as an explanation for the intended user query.

Formally, the ALP framework is defined as a triple <*T*, *A*, *I*>, where *T* is a theory defined by a logic program, *A* is a set of predicates designated as abducible predicates, or *abducibles* for short, and *I* is the set of constraints (Eshghi and Kowalski, 1989; Denecker and Kakas, 2002). Given a query (i.e., observation) *Q*, the abductive task is characterized by the following:

$$<T, A, I>, Q \vdash_{abd} \Delta$$

that is, the task is to derive a set of abducibles Δ as an explanation for *Q* such that

$T \cup \Delta \models Q$;

$T \cup \Delta$ satisfies *I*; and

$T \cup \Delta$ is consistent

The abducibles can be either ground or non-ground with variables existentially qualified, e.g., $\exists \vec{X}\ a(\vec{X})$; the latter case is known as *constructive abduction* (Kakas and Mancarella, 1993).

Constraint Logic Programming (CLP) is an extension of logic programming with a set of constraint predicates and the corresponding dedicated constraint solvers. These solvers are implemented with special algorithms to gain computational efficiency.

The similarity between ALP and CLP is observed by Kakas, et al. (2000): in both frameworks an answer to a query is constructed from a set of special predicates – abducible predicates in ALP and constraint predicates in CLP, which are constrained either by integrity constraints in ALP or by a constraint theory in CLP. Abductive constraint logic programming (ACLP) aims to unify these two frameworks by posting abducible predicates and constraint predicates into a *constraint store*; we use the term *constraint store* to informally refer to a collection of constraints that need to be solved and checked for consistency. The inconsistency of the store is used as immediate feedback to the abduction procedure, which promptly abandons hopeless search branches to reduce search space.

There are numerous constraint solving techniques; the book by Marriott and Stuckey (1998) is a good and extensive introduction to this topic, and the book by Frühwirth and Abdennadher (2003) provides a concise introduction and numerous examples using a high level and declarative solver specification language called Constraint Handling Rules (CHR). The constraint theory in COIN is implemented using CHR, which is briefly introduced in the next sub-section.

4.3.2 Constraint Handling Rules (CHR)

Traditional constraint solvers were implemented in low-level languages as "black boxes" that are difficult to debug, maintain, and explain. Most built-in constraint solvers in many CLP systems are still implemented in this fashion. CHR was introduced as a declarative language for writing solvers for user defined constraints; it offers significantly more flexibility than the traditional approach and is much easier to develop solvers for fast prototyping. More details about CHR can be found in Frühwirth (1998), here we give a brief introduction on its syntax and informal semantics.

CHR distinguishes two kinds of constraints: built-in constraints and CHR (i.e., user defined) constraints. Built-in constraints are handled by the CLP system that hosts CHR; CHR constraints are defined by a CHR program, which consists of a finite set of rules as defined below.

There are three types of rules:

- simplification rule: $E \Leftrightarrow C \mid G$
- propagation rule: $E \Rightarrow C \mid G$
- simpagation rule: $E \setminus F \Leftrightarrow C \mid G$, which is a shorthand for $E \wedge F \Leftrightarrow C \mid E \wedge G$

where *E* (or *F*), called the *head*, is a conjunction CHR constraints; *C*, called the *guard*, is a conjunction of built-in constraints; and *G*, called the *goal*, can be *true*, *fail*, or a conjunction of built-in and CHR constraints. Here *true* and *fail* are trivial built-in constraints.

Before we present the *operational* and *declarative* semantics of CHR, let us first look at an example of a CHR program consisting of simplification and propagation rules.

Example (CHR program for ≤): Given the built-in constraints *true* and =, below is a CHR program for partial order relation $\leq$:

$X \leq Y \Leftrightarrow X = Y \mid true$ (r1: reflectivity)

$X \leq Y \wedge Y \leq X \Leftrightarrow X = Y$ (r2: anti-symmetry)

$X \leq Y \wedge Y \leq Z \Rightarrow X \leq Z$ (r3: transitivity) □

Operational semantics. Intuitively, when a simplification rule is applied, the constraints in the constraint store that match the rule *head* are replaced with the *goal* of the rule. Take r1 above as an example: *guard X*=*Y* will be first checked; if guard checking returns true, head *X*≤*Y* will be replaced with *goal true*; if guard checking returns false, the rule will not be applied. Like r1 and r2 in the example above, simplification rules often have a smaller number of constraints in the *goal* than in the head, therefore they tend to reduce the number of constraints to be solved.

When a propagation rule is applied, the constraints in the *goal* are added to the store without removing the ones matching the *head.* Propagation rules (e.g., r3 above) appear to introduce certain redundant constraints, but this redundancy may be useful to trigger the application of other rules that can further reduce the number of constraints.

As is usually done for programming languages, the operational semantics of CHR is formally given through a state transition system. A state is a pair <*G*, *C*>, where *G* is a goal, and *C* is a built-in constraint. The initial state is of the form <*G*, *true*>. A state transitions to the next by applying a *simplify*, *propagate*, or *solve* transition rule. The *simplify* and *propagate* transition rules are similar to the intuitive explanation to simplification and propagation rules above. The *solve* transition rule updates a state (e.g., consolidating built-in constraints in a state) to a normal form as defined in Abdennadher et al. (1999). A formal specification of these transition rules can be found in (Frühwirth and Abdennadher, 2003). A successful final state is reached when there no applicable rule, and the state is of the form <*G*, *C*> where *C* is not *false*; <*G*, *false*> is a failed state.

Let us continue to use the partial order ($\leq$) example to illustrate the use of transition rules. Given the goal $A{\leq}B \wedge C{\leq}A \wedge B{\leq}C$, the following shows the state transitions that solve the goal; the CHR constraints that are considered in the current transition step are underlined:

$$
\begin{array}{ll}
< \underline{A{\leq}B} \wedge \underline{C{\leq}A} \wedge B{\leq}C, true > & \\
< A{\leq}B \wedge C{\leq}A \wedge \underline{B{\leq}C} \wedge \underline{C{\leq}B}, true > & (propagate, r3) \\
< \underline{A{\leq}B} \wedge \underline{B \leq A}, B = C > & (simplify, r2) \\
< \underline{A = B}, B = C > & (simplify, r2) \\
< top, A{=}B \wedge B{=}C > & (solve)
\end{array}
$$

where the state in the line is a successful final state and "*top*" in the state represents an empty goal.

Declarative semantics. The *declarative semantics* of CHR is given by the following FOL formulas corresponding to simplification and propagation rules:

$$E \Leftrightarrow C \mid G \;\equiv\; \forall(C \rightarrow (E \leftrightarrow \exists\vec{y}\,G))$$

$$E \Rightarrow C \mid G \;\equiv\; \forall(C \rightarrow (E \rightarrow \exists\vec{y}\,G))$$

where $\vec{y}$ are the variables that appear only in the body G.

The soundness and completeness of the derivation procedure specified by the operational semantics can be established using the above declarative semantics (Abdennadher et al., 1999).

4.3.3 Query Answering in COIN

Figure 4.5 illustrates how queries are evaluated in COIN. It is a two step process: the user query is first mediated to generate a mediated query as the *intensional* answer; the mediated query is then optimized and executed to retrieve the data instances as the *extensional* answer. We focus on the mediation step only in this Thesis.

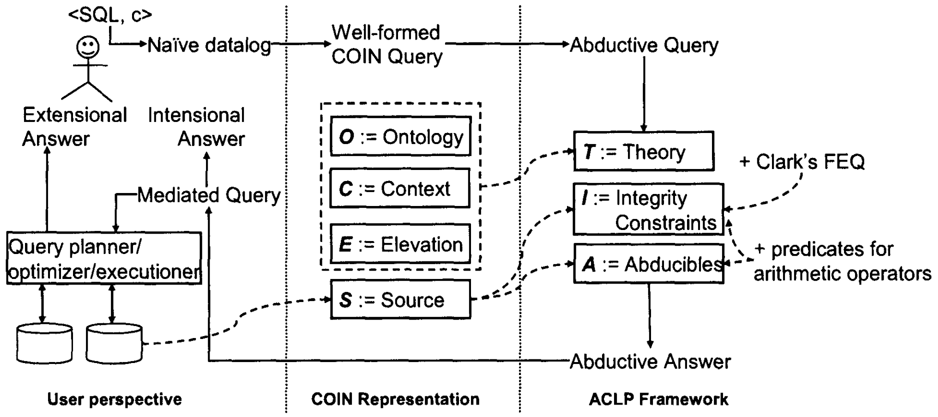


Figure 4.5 Query Answering in COIN

The user query in SQL along with the user context *c* are submitted to the system. The SQL is first translated into a clausal form (i.e., datalog) using predicates corresponding to primitive relations and the conditions specified in user query. This query is called naïve datalog query because it does not consider the semantic differences between the source contexts and the user context. For example, when <Q1, *c_usa*> is submitted, where Q1 is the first sample query as shown below:

Q1:
```
select QDate, AdjClose from YHFrankfurt
where Ticker="IBM.f" and QDate >="12/20/1998" and QDate =<"01/10/1999";
```

the system generates the following naïve datalog query:

```
answer(QDate,AdjClose) :-
      yhfrankfurt(Ticker,QDate,AdjClose,
                    StartMonth,StartDay,Startyear,EndMonth,EndDay,EndYear),
      Ticker="IBM.F", QDate>="12/20/1998", QDate =<"01/10/1999".
```

Then the system translates the naïve datalog query into a so called well-formed COIN query. Both Goh (1997) and Firat (2003) discuss the procedure for this translation, which systematically renames the variables and introduces *value* predicate so that all primitive values are in the user context. The well-formed COIN query of the above example is:

```
answer(V10,V11) :- yhfrankfurt’(V1’,V2’,V3’,V4’,V5’,V6’,V7’,V8’,V9’),
                   value(V1’, c_usa, "IBM.f"),
                   value(V2’, c_usa, V10), V10>="12/20/1998", V10 =<"01/10/1999",
                   value(V3’, c_usa, V11).
```

Notice that the first predicate in the body is a semantic relation with arguments V1’ through V9’ representing semantic objects. The types and contexts of semantic objects V1’, V2’ and V3’ are illustrated earlier in Figure 4.2; semantic objects V4’ through V9’ are of *basic* type. As discussed in Section 4.2, the *value* predicate introduces the composite conversion for the semantic object; the composite conversion as specified in formula (4.5) then introduces component conversions specified for the semantic object. This process is automated by the mediator, which is implemented in ACLP.

In Figure 4.5 we illustrate the approximate correspondences between the COIN representation components <*O*, *C*, *E*, *S*> and the ACLP components <*T*, *A*, *I*>. Specifically:

- Ontology *O*, context descriptions *C*, and elevation axioms *E* of COIN correspond to logic theory *T* in ACLP;
- Abducible predicates *A* include the predicates corresponding to source relations (e.g., *yhfrankfurt*/9, *olsen*/4, etc.)[33], predicates corresponding to externally defined functions, and built-in predicates corresponding to arithmetic and relational (comparison) operators;
- Integrity constraints *I* include the constraints defined in sources *S*, Clark’s Free Equality (Clark, 1978) axioms, and symbolic equation solving constraints (Firat, 2003).

Abduction is implemented with an extended *SLD+Abduction* procedure as described in Goh (1997); constraint handling is implemented with CHR. Abducible predicates encountered in the procedure are put into a constraint store whose consistency is maintained by a CHR program. The mediated query is constructed from the abducibles in the constraint store after the CHR program has reached a successful final state. More details will be provided in Section 4.5.

4.4 Representation for Temporal Semantic Heterogeneity

We described the existing COIN approach in two previous sections. We gave a procedural definition for the *value* method not previous defined in COIN literature. For a semantic object, this method

[33] We follow the Prolog convention that uses predicate/arity as a shorthand notation to refer to a predicate without explicitly listing all its arguments; arity indicates the number of arguments the predicate expects.

derives the primitive value of the object in any given context. We also clarified the notion of modifier orthogonality that is only briefly mentioned in Goh (1997) and Firat (2002).

Starting from this section, we will present the temporal extensions developed for the existing COIN. This section focuses on the extensions to the representation formalism, and the next section focuses on the extensions to the reasoning mechanism.

As outlined in Chapter 2, we need to address the kind of temporal semantic heterogeneity caused by time-varying semantics, which inevitably involves heterogeneous time models that include such things as different representations (e.g., Month/Year *v.* Year-Month), operations capable to perform (e.g., not all sources support ≤ operation on temporal entities), and implicit assumptions about temporal concepts (local time *v.* UTC time).

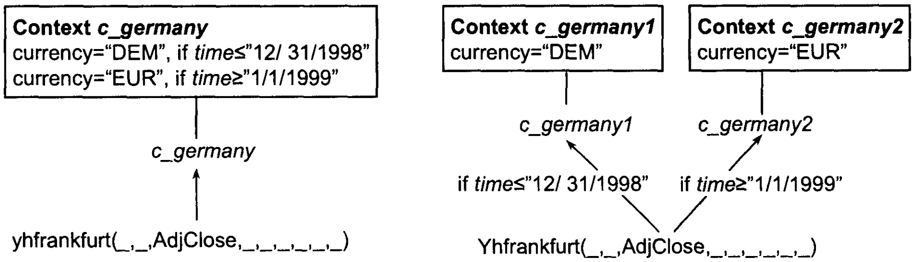
Similar to the case of non-temporal concepts, the different representations and implicit assumptions about temporal concepts can be described using ontology and contexts. Depending on the integration scenario, the ontology may need to include a single temporal concept (e.g., temporal entity, date, etc) or all concepts of a full-blown time ontology such as the DAML time ontology (Hobbs, 2002). What modifiers are needed for describing contexts (i.e., possible variations of interpretation of a concept) also depends on the integration scenario. For example, for integration of certain time series data, a date format modifier for describing different formats of date values my be enough, while for integration of flight information, we may also need a modifier for time offset (i.e., combination of time zone and whether daylight saving time is in effect) in addition to date format.

4.4.1 Dynamic Modifier v. Dynamic Context Association

We identify two possible ways of representing time-varying semantics:

- **Approach 1** (*dynamic modifier*): allow modifiers to have non-static values within a context referenced by a context identifier, i.e., a modifier can have different values at different times in a context.
- **Approach 2** (*dynamic context association*): allow modifiers to only have static values within a context referenced by a context identifier, but associate an argument (i.e., attribute) of a primitive relation with different contexts at different times.

In Figure 4.6 below, we illustrate the two approaches when they are used to describe time-varying currencies of the "AdjClose" argument in primitive relation *yhfrankfurt*/9.



(a) dynamic modifier (b) dynamic context association

Figure 4.6 Two Approaches to Describing Time-varying Semantics

With the dynamic modifier approach (Figure 4.6 (a)), we use *c_germany* to refer to the context within which modifier *currency* has a value of "DEM" until December 31, 1998 and a value of "EUR" since January 1, 1999. Then we simply associate the "AdjClose" argument of *yhfrankfurt*/9 with context *c_germany*. The parameter *time* in both (a) and (b) of Figure 4.6 will be bound to temporal entities in sources after a source attribute is associated with the context.

With the dynamic context association approach (Figure 4.6 (b)), we describe two contexts: in context *c_germnay1*, modifier *currency* has a value of "DEM"; in context *c_germany2*, modifier *currency* has value of "EUR". Then we dynamically associate the "AdjClose" argument of *yhfrankfurt*/9 with the two contexts: "AdjClose" is associated with *c_germany1* until December 31, 1998, and it is associated with *c_germany2* since January 1, 1999.

The following two definitions will be useful in the ensuing discussion about the two approaches.

Definition 4.3 (*Static Context*) A context is a *static context* iff every modifier has a single static value in the context.

Definition 4.4 (*Temporal Context*) A context is *temporal context* iff at least one modifier has time-varying values in the context.

The apparent differences between the two approaches are where time-dependent conditions are specified: with the dynamic modifier approach the time-dependent conditions are given in the description of temporal context, thus context association is simpler; with the dynamic context association approach the time-dependent conditions are given in the context association, thus only static context descriptions are necessary.

We select the dynamic modifier approach to representing time-varying semantics for the following two reasons.

First, the existing COIN can be straightforwardly extended to admit dynamic modifiers. We will present this extension in Section 4.4.2. In contrast, it is much more involved to develop the extensions for the dynamic context association approach. Recall from Section 4.2.3 (especially formulas 4.6 and 4.7) that a *Skolem* function is used to associate a relational attribute to a context. We have to introduce more complicated structures to include time-dependent context associations. In addition, since each semantic object has an original context, now multiple semantic objects need to be created for a relational attribute (e.g., for "AdjClose", we need a semantic object with original context *c_germany1*, and another one with original context *c_germany2*). Consequently, we need to cerate multiple semantic relations for a primitive relation. Further research is needed to find means for reducing or avoiding such complications.

Second, when multiple sources share the same context, the dynamic modifier approach avoids repeated specifications of the time-dependent conditions. For example, in addition to Frankfurt, all the other exchanges in Germany switched currency in the beginning of 1999, thus the temporal context *c_germany* can be shared among them. However, with the dynamic context association, the time-dependent conditions have to be specified repeatedly for all exchanges despite that share the two static contexts *c_germany1* and *c_germany2*. Managing these redundancies can be cumberson and error-prone.

We shall point out that a potentially useful feature may be developed by extending the notion of dynamic context association to dynamic elevation, i.e., a relational attribute in a primitive relation can be elevated to different semantic types at different times. The feature is useful when certain fields in a relational source are recycled to represent different things at different times. Field recycling is a form of domain evolution exemplified in Ventrone and Heiler (1991). Future research should further examine the merit of dynamic context association as well dynamic elevation.

In a temporal context the value of at least one modifier is dependent on time. In some cases the modifier value may depend on factors other than time. Thus we can extend the notion of temporal context to *dynamic context*, of which temporal context is a special case.

Definition 4.5 (*Dynamic Context*) A context is a *dynamic context* iff at least one modifier has multiple values in the context.

Goh (1997) gives an example of dynamic context where company financials are reported in the official currencies of the countries where the companies are incorporated; the value of *currency* modifier is determined by first obtaining the *company* attribute, then using it to find the country in which the *company* is incorporated, and lastly finding the *official currency* of the country and assigning it to the modifier. Country of incorporation information is in the data source and can be determined only at query execution step. Goh et al. (1999) provides another example where in a context the *scaleFactor* modifier depends on the *currency* modifier and the currency modifier depends on the country of incorporation like in the other example; *scaleFactor* has a value 1000 if the *currency* is "JPY", and a value 1 otherwise.

The dependency of modifier on factors that cannot be determined during the mediation step complicates the provision of component conversions and the mediated query has to consider all possibilities. Temporal context is a special case because the modifier value depends on time only, not other arbitrary factors. We can take advantage of this special case by systematically processing the time dependency; in Section 4.5, we show how this is done to simplify (i.e., semantically optimize) the mediated query.

4.4.2 Representation of Time-Dependent Dynamic Modifiers

A dynamic modifier can have multiple values over the time span in a temporal context, with each individual value being valid for a particular time period within the span. The previously defined COIN language allows single-valued modifiers only; certain extension is necessary to allow for multi-valued modifiers. There are two alternative approaches to the extension: (1) treating *time* as a distinguished type (much like type *ctx* for context) in the COIN data model; and (2) treating *time* as a user defined type (i.e., a semantic type in the domain ontology). Below discuss these two alternatives.

Approach 1*: Time as a Distinguished Type.* This approach introduces a new object type *tmp* for symbols representing temporal entities. The modifier declaration and the modifier atom can be augmented with an argument of type *tmp* as below:

$\tau[m@ctx, tmp \Rightarrow \tau']$

$t[m@c, t_m \rightarrow t']$, where c:*ctx*, and t_m:*tmp*.

Because a modifier can be dynamic in one context and static in another, declarations with one parameter and two parameters are both needed, which is supported by the polymorphism of the language. This seems to be a plausible extension. But there are at least two issues that lie in the new object type *tmp*. First, the purpose of introducing context is to use it as a mechanism of capturing the sources of variance in data interpretation; *time* is just another such source, so it should be part of context instead of being a special type that is of the same significance as context. Second, as we discussed in Chapter 2, *time* is a complex data type. Using a symbol to represent an instance of a complex data type requires the language to be committed to a predefined model for the complex type (e.g., time point or time period, single- or multi-granularity, a calendar system for referencing time line, etc.). Such commitment not only limits the expressiveness of the language, it may be even impossible to suit the diverse needs for time representation in different integration scenarios.

***Approach 2**: Time as a User Defined Type*. Instead of making *time* a distinguished type, we can treat it like any other semantic type, which means that the semantics of the type is specified externally by the ontology and the context descriptions, not by the COIN language. This alternative approach allows for greater flexibility in handling time. We can do so by extending the language to allow a modifier method to return multiple semantic objects with the following modifier declaration:

$\tau[m@ctx \Rightarrow\Rightarrow \tau']$

where the double-arrow indicates that modifier *m* in context *ctx* has multiple objects of type τ'. Since these objects have different values in the same context, we sometimes call a dynamic modifier a *multi-valued* modifier. For example, we use the following to declare that the *currency* modifier in certain context is multi-valued:

$monetaryValue[currency@ctx \Rightarrow\Rightarrow basic]$

We distinguish a *multi-valued* method here from a *set-valued* method in F-logic. The validity of different objects returned by a multi-valued modifier is dependent on different (temporal) conditions; all elements in the set returned by a set-valued method are valid under the same set of conditions.

The different conditions of a dynamic modifier are specified using multiple rules. When describing time-varying semantics, the conditions inevitably contain comparisons of temporal entities. For example, when quote date is =< "December 31, 1998", the currency is "DEM" in context *c_germany*. Here the comparison is between temporal entities in a source and the ground temporal entity "December 31, 1998". Since we can only compare primitive values in the same context, we need to decide the context in which the compassion is performed. There are two alternative choices of the context: (1) the source context (more precisely is the context to be described); and (2) the receiver context, which is known when a query is submitted for mediation. We discuss each choice below.

Temporal Comparison in Source Context. In this case, the ground temporal entity in the comparing can be a primitive object in the source, e.g., we can use "31-DEC-98" for "December 31, 1998". The following rules illustrate how to define modifier *currency* for context *c_germany* using *c_germany* as the context of comparison:

$$
\begin{array}{l}
O: monetaryValue, currency(c\_germany, O): basic \vdash \\
\quad O[currency@c\_germany \rightarrow currency(c\_germany, O)] \leftarrow \\
\qquad currency(c\_germany, O)[value@c\_germany \rightarrow "DEM"], \\
\qquad O[tempAttribute \rightarrow T], T[value@c\_germany \rightarrow T_v], T_v \leq "31\text{-}DEC\text{-}98".
\end{array}
$$

(4.8)

$$
\begin{array}{l}
O: monetaryValue, currency(c\_germany, O): basic \vdash \\
\quad O[currency@c\_germany \rightarrow currency(c\_germany, O)] \leftarrow \\
\qquad currency(c\_germany, O)[value@c\_germany \rightarrow "EUR"], \\
\qquad O[tempAttribute \rightarrow T], T[value@c\_germany \rightarrow T_v], T_v \geq "1\text{-}JAN\text{-}99".
\end{array}
$$

(4.9)

In the above rules, $currency(c\_germany, O)$ is a *Skolem* function representing the semantic object returned by the method of *currency* modifier. In the body of each rule, the primitive value of the object is given, the remaining atoms first obtain the temporal attribute *T* of semantic object *O*, then derive the primitive value of *T* in the context to be described (i.e., *c_germany*), and specify certain constraint the primitive value needs to satisfy using comparison predicates such as $\leq$ and $\geq$.

These comparison predicates are abducible constraint predicates and are abducted during query mediation. For example, when rule (4.8) is selected during mediation, $T_v \leq "31\text{-}DEC\text{-}98"$ is abducted into the constraint store. When the user query involves multiple contexts, constraints with primitive values in these contexts will be abducted into the constraint store. Thus the constraint store will consist of constraints of primitive values in different contexts (e.g., “31-DEC-98”, “12/31/1998”, “1998.12.31”, 31-12-1998”, etc. when the involved contexts have different date formats as shown with these sample values). As we will see in the next section, there are several challenging difficulties in solving constraints in multiple contexts. Therefore, we need to consider the choice of performing comparison only in the *receiver context*.

Temporal Comparison in Receiver Context. In this case, the ground temporal entity in the comparison should be a semantic object from the source. This semantic object is created using a *Skolem* function and a special predicate *cste*/1 that is always evaluated *true*. To access the receiver context when a user query is submitted, we need to add a fact about the receiver context C: *rcvContext(C)*. The following rules illustrate how to define modifier *currency* for context *c_germany* using receiver context *C* as the context of comparison:

$$
\begin{array}{l}
O: monetaryValue, currency(c\_germany, O): basic \vdash \\
\quad O[currency@c\_germany \rightarrow currency(c\_germany, O)] \leftarrow \\
\qquad currency(c\_germany, O)[value@c\_germany \rightarrow "DEM"], \\
\qquad rcvContext(C), O[tempAttribute \rightarrow T], T[value@C \rightarrow T_v], \\
\qquad skolem(date, "31\text{-}DEC\text{-}98", c\_germany, 1, cste("31\text{-}DEC\text{-}98"))[value@C \rightarrow T_c] \\
\qquad T_v \leq T_c.
\end{array}
$$

(4.10)

$$\begin{array}{l}
O: monetaryValue, currency(c\_germany, O): basic \vdash \\
\quad O[currency@c\_germany \rightarrow currency(c\_germany, O)] \leftarrow \\
\qquad currency(c\_germany, O)[value@c\_germany \rightarrow \text{"}EUR\text{"}], \\
\qquad rcvContext(C), O[tempAttribute \rightarrow T], T[value@C \rightarrow T_v], \\
\qquad skolem(date, \text{"}1\text{–}JAN\text{–}99\text{"}, c\_germany, 1, cste(\text{"}1\text{–}JAN\text{–}99\text{"}))[value@C \rightarrow T_c] \\
\qquad T_v \geq T_c.
\end{array}$$

(4.11)

With this form of representation, primitive values in all constraints abducted into the constraint store are in the receiver context. For example, if the receiver context is *c_usa*, the *value* method for semantic object created for the ground temporal entity in (4.10) will derive primitive value "12/31/1998" for T_c.

Time-varying Semantics of Temporal Entities

In the preceding discussion we used examples where the interpretations of semantic objects other than temporal entities are dependent on the comparisons of temporal entities. Such dependency is specified using temporal entities in the context to be described in both forms of representation: one uses the primitive objects, the other uses the semantic objects. The implicit assumption is that the interpretations of these temporal entities do not change over time, i.e., the modifiers of the temporal entities should be static. We will discuss the implications of the non-static case and suggest potential solutions in Section 4.6.

4.4.3 Backward Compatibility

The representation extensions presented above do not affect how static modifiers are defined. For example, the following much simpler rule specifies the currency modifier in context *c_usa*:

$$\begin{array}{l}
O: monetaryValue, currency(c\_usa, O): basic \vdash \\
\quad O[currency@c\_usa \rightarrow currency(c\_usa, O)] \leftarrow \\
\qquad currency(c\_usa, O)[value@c\_usa \rightarrow \text{"}USD\text{"}].
\end{array}$$

(4.12)

which states that in context *c_usa* currency is always USD.

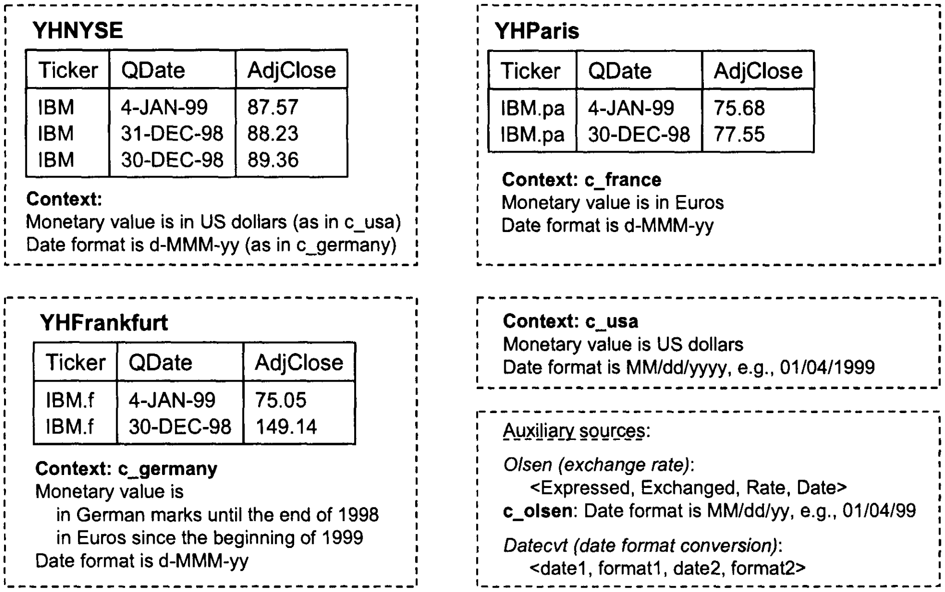
As we will see in the next section, abducible constraint predicates appearing in dynamic modifier definitions are abducted into a constraint store; by solving the constraints of all modifiers involved in the user query, we obtain a set of simplified common constraints under which all modifiers have a single valid value. This means that we do not need to change how component conversions are specified.

4.5 Reasoning about Temporal Semantic Heterogeneity

We have chosen to use dynamic modifiers to represent time-varying semantics and treat temporal entity types like other semantic types when the interpretations of temporal entities do not change over time. This representation formalism makes it possible to use the existing COIN query answering mechanism to reason about temporal semantic heterogeneity. We first illustrate the use of existing COIN for reconciling temporal semantic heterogeneity. Certain deficiencies of existing COIN are identified. Then we present the improvements made in constraint handling to overcome these deficiencies.

4.5.1 Reconciling Temporal Semantic Heterogeneity with Existing COIN

We will use the historical stock price example in Chapter 3 for illustration. We need Figure 3.2 for an informal description of the contexts and Figure 4.2 for the ontology. For ease in reading, we reproduce the figures below.



YHNYSE

Ticker	QDate	AdjClose
IBM	4-JAN-99	87.57
IBM	31-DEC-98	88.23
IBM	30-DEC-98	89.36

Context:
Monetary value is in US dollars (as in c_usa)
Date format is d-MMM-yy (as in c_germany)

YHParis

Ticker	QDate	AdjClose
IBM.pa	4-JAN-99	75.68
IBM.pa	30-DEC-98	77.55

Context: c_france
Monetary value is in Euros
Date format is d-MMM-yy

YHFrankfurt

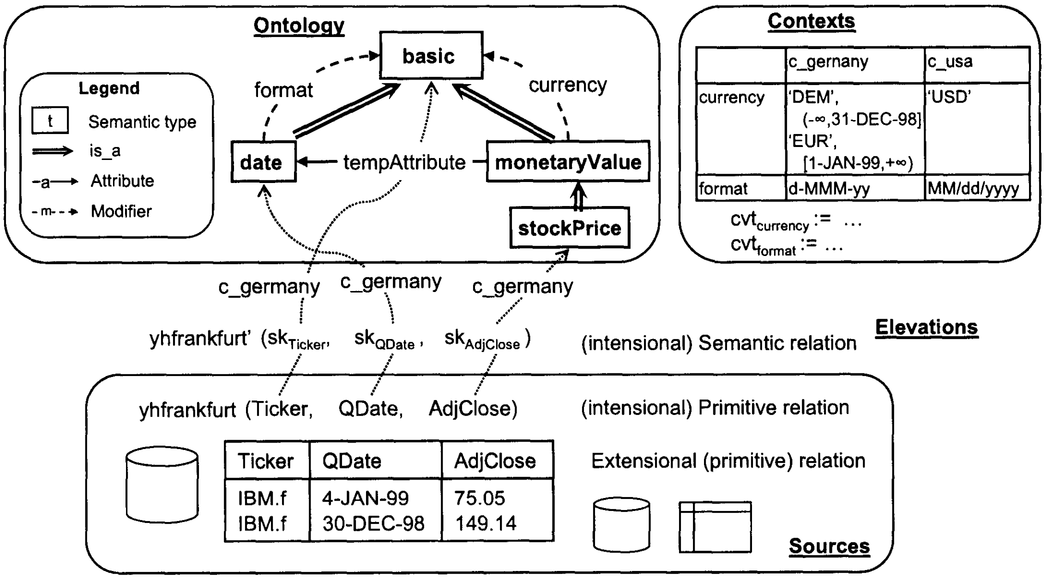
Ticker	QDate	AdjClose
IBM.f	4-JAN-99	75.05
IBM.f	30-DEC-98	149.14

Context: c_germany
Monetary value is
in German marks until the end of 1998
in Euros since the beginning of 1999
Date format is d-MMM-yy

Context: c_usa
Monetary value is US dollars
Date format is MM/dd/yyyy, e.g., 01/04/1999

Auxiliary sources:
Olsen (exchange rate):
<Expressed, Exchanged, Rate, Date>
c_olsen: Date format is MM/dd/yy, e.g., 01/04/99
Datecvt (date format conversion):
<date1, format1, date2, format2>

Figure 3.2 Integration of historical stock prices from multiple exchanges (reproduced)



	c_gernany	c_usa
currency	'DEM', (-∞,31-DEC-98] 'EUR', [1-JAN-99,+∞)	'USD'
format	d-MMM-yy	MM/dd/yyyy

Ticker	QDate	AdjClose
IBM.f	4-JAN-99	75.05
IBM.f	30-DEC-98	149.14

Figure 4.2 Components of Knowledge Representation in COIN (reproduced)

There are two modifiers in the ontology: *format* for type *date* and *currency* for type *monetaryValue*. The component conversions for the modifiers are specified as follows:

$$\begin{array}{l} D : date \vdash \\ \quad D[cvt@format, C_r, M_s, M_r, U \rightarrow V] \leftarrow \\ \qquad datecvt(U, M_s, V, M_r). \end{array} \tag{4.13}$$

$$\begin{array}{l} M : monetaryValue \vdash \\ \quad M[cvt@currency, C_r, M_s, M_r, U \rightarrow V] \leftarrow \\ \qquad M[tempAttribute \rightarrow T], T[value@C_r \rightarrow T_v], olsen'(F_c', T_c', R', D'), \\ \qquad F_c'[value@C_r \rightarrow M_s], T_c'[value@C_r \rightarrow M_r], D'[value@C_r \rightarrow T_v], \\ \qquad R'[value@C_r \rightarrow R_v], V = U * R_v. \end{array} \tag{4.14}$$

Rule (4.13) specifies date format component conversion. In its body, predicate *datecvt*(*Date*$_1$, *Format*$_1$, *Date*$_2$, *Format*$_2$) corresponds to the primitive relation (implemented as a function) that converts a date value from one given format to another given format; e.g., the query

```
:- datecvt("1-JAN-99", "d-MMM-yy", V, "MM/dd/yyyy").
```

would return `V="01/01/1999"`. In the rule body we use the primitive relation rather than the semantic relation because the terms appearing in the head and as the arguments of *datecvt*/4 are all primitive objects.

Rule (4.14) specifies currency component conversion. It uses *olsen*(*From_currency*, *To_currency*, *Rate*, *Date*) to retrieve the exchange rate between a pair of currencies on a given date. We use the semantic relation of *olsen*/4, indicated with prime post-fixed predicate and arguments, to ensure that date values in *olsen* and of the temporal attribute in the source (obtained via *tempAttribute*) are converted into the same context before they are equated. Since the first three arguments in *olsen* are of type *basic* so their values are context invariant, we can specify the component conversion alternatively by converting the date value of *tempAttribute* to *olsen*'s context and use the primitive relation of *olsen*:

$$\begin{array}{l} M : monetaryValue \vdash \\ \quad M[cvt@currency, C_r, M_s, M_r, U \rightarrow V] \leftarrow \\ \qquad M[tempAttribute \rightarrow T], T[value@c\_olsen \rightarrow T_v], olsen(M_s, M_r, R, T_v), \\ \qquad V = U * R. \end{array} \tag{4.15}$$

Let us continue to use the sample query Q1 with *c_usa* being the receiver context. As shown before, the well-formed COIN query for Q1 is:

```
answer(V10,V11) :- yhfrankfurt'(V1',V2',V3',V4',V5',V6',V7',V8',V9'),
                   value(V1', c_usa, "IBM.f"),
                   value(V2', c_usa, V10), V10>="12/20/1998", V10 =<"01/10/1999",
                   value(V3', c_usa, V11).
```

As a reminder, the first three arguments for *yhfrankfurt*/9 are "Ticker", "QDate", and "AdjClose", which are mapped to *basic*, *date*, and *stockPrice* types respectively; *stockPrice* is a subtype of *monetaryValue*.

When the ACLP based mediator receives this query, it tries to prove each predicate in the body. We illustrate this mediation process in Figure 4.7, where for each step we underline the predicate to be resolved and show the constraint store in a shaded box. Near the bottom of the figure we show one of the two branches that correspond to the two rules specifying *currency* modifier in *c_germany*.

Recall form Section 4.2.2 that the *value* method first finds modifier assignments, then applies component conversions of the modifiers. For ease in reading, we show in the figure one of the two specifications for *currency* modifier and the rule for *currency* component conversion. Further explanation of the mediation process is given below.

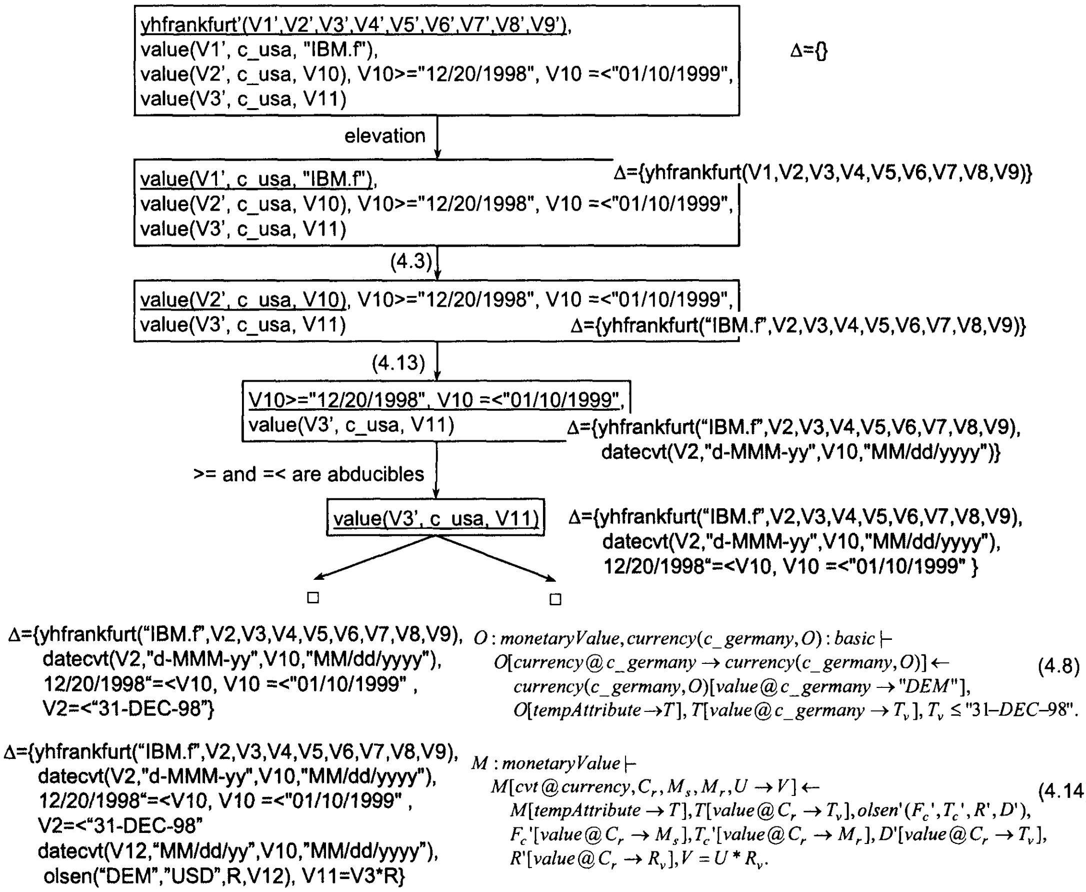

yhfrankfurt'(V1',V2',V3',V4',V5',V6',V7',V8',V9'),
value(V1', c_usa, "IBM.f"),
value(V2', c_usa, V10), V10>="12/20/1998", V10 =<"01/10/1999",
value(V3', c_usa, V11)
Δ={}
elevation
value(V1', c_usa, "IBM.f"),
value(V2', c_usa, V10), V10>="12/20/1998", V10 =<"01/10/1999",
value(V3', c_usa, V11)
Δ={yhfrankfurt(V1,V2,V3,V4,V5,V6,V7,V8,V9)}
(4.3)
value(V2', c_usa, V10), V10>="12/20/1998", V10 =<"01/10/1999",
value(V3', c_usa, V11)
Δ={yhfrankfurt("IBM.f",V2,V3,V4,V5,V6,V7,V8,V9)}
(4.13)
V10>="12/20/1998", V10 =<"01/10/1999",
value(V3', c_usa, V11)
Δ={yhfrankfurt("IBM.f",V2,V3,V4,V5,V6,V7,V8,V9), datecvt(V2,"d-MMM-yy",V10,"MM/dd/yyyy")}
>= and =< are abducibles
value(V3', c_usa, V11)
Δ={yhfrankfurt("IBM.f",V2,V3,V4,V5,V6,V7,V8,V9), datecvt(V2,"d-MMM-yy",V10,"MM/dd/yyyy"), 12/20/1998"=<V10, V10 =<"01/10/1999" }
□
□
Δ={yhfrankfurt("IBM.f",V2,V3,V4,V5,V6,V7,V8,V9), datecvt(V2,"d-MMM-yy",V10,"MM/dd/yyyy"), 12/20/1998"=<V10, V10 =<"01/10/1999" , V2=<"31-DEC-98"}
O : monetaryValue, currency(c_germany, O) : basic ⊢ O[currency@c_germany → currency(c_germany, O)] ← currency(c_germany, O)[value@c_germany → "DEM"], O[tempAttribute → T], T[value@c_germany → T_v], T_v ≤ "31-DEC-98".
(4.8)
Δ={yhfrankfurt("IBM.f",V2,V3,V4,V5,V6,V7,V8,V9), datecvt(V2,"d-MMM-yy",V10,"MM/dd/yyyy"), 12/20/1998"=<V10, V10 =<"01/10/1999" , V2=<"31-DEC-98" datecvt(V12,"MM/dd/yy",V10,"MM/dd/yyyy"), olsen("DEM","USD",R,V12), V11=V3*R}
M : monetaryValue ⊢ M[cvt@currency, C_r, M_s, M_r, U → V] ← M[tempAttribute → T], T[value@C_r → T_v], olsen'(F_c', T_c', R', D'), F_c'[value@C_r → M_s], T_c'[value@C_r → M_r], D'[value@C_r → T_v], R'[value@C_r → R_v], V = U * R_v.
(4.14)

Figure 4.7 Illustration of Mediation Process for Query Q1.

The first predicate in the body is a semantic relation, which is resolved with the elevation axiom for primitive relation *yhfrankfurt*/9. This binds argument variables V1' through V9' with the semantic objects (i.e., *Skolem* terms in this case) corresponding to the arguments in the primitive relation and produces *yhfrankfurt*/9 as the new goal to resolve. Since predicates of primitive relations are abducible predicates, *yhfrankfurt*/9 is abducted into the constraint store. The constraint store now has one predicate: `{yhfrankfurt(V1,V2,V3,V4,V5,V6,V7,V8,V9)}`.

Then it tries to resolve the first *value* predicate. Object V1' is of type *basic*; according to formula (4.3), the argument in primitive relation corresponding to this object is bound to "IBM.F". Thus the constraint store becomes `{yhfrankfurt("IBM.f",V2,V3,V4,V5,V6,V7,V8,V9)}`.

When it tries to resolve the second *value* predicate, because V2' is of type *date* and has modifier *format*, the composite conversion for V2' is generated according to the template in formula (4.5). The composite conversion invokes the component conversion for *format*, which resolves to *datecvt*/4 according to the conversion specified in (4.13). Since *datecvt*/4 corresponds to a primitive relation, it is abducted into the constraint store. The remaining two predicates (>= and =<) in the same line are constraint predicates, which are abducibles and are abducted. There is a simplification rule that rewrites >= to =<. Now the constraint store becomes:

```
{yhfrankfurt("IBM.f",V2,V3,V4,V5,V6,V7,V8,V9),
 datecvt(V2,"d-MMM-yy",V10,"MM/dd/yyyy"),
 "12/20/1998"=<V10, V10 =<"01/10/1999" }
```

The last *value* predicate introduces composite conversion for object corresponding to *AdjClose* in the source. It is of inherited type *monetaryValue* and has a *currency* modifier. Following the template of (4.5), the mediator first finds the modifier definitions in *c_germany* and *c_usa* contexts. In context *c_germany*, modifier *currency* has different values in two time periods, so we use two rules to specify the values and the corresponding temporal constraints. We provided two alternatives to specifying the constraints: one specifies constraints using *primitive objects* in the context the be described, the other specified uses the *semantic objects* in the context to be described.

Let us consider the first alternative: rule (4.8) is selected first and rule (4.9) (not shown in Figure 4.7) will be used on backtracking. In processing the rule body of (4.8), modifier *currency* is assigned value "DEM"; because the context parameter of the *value* method of temporal attribute *T* is the same as *T*'s context, it binds T_v to *QDate* (represented by variable V2) in the primitive relation; the next predicate is a constraint predicate and is abducted, i.e., V2 =<"31-DEC-98" is added to the constraint store.

For context *c_usa*, rule (4.12) is used to assign value "USD" for modifier *currency*. Then the component conversion for currency is introduced. The two alternative specifications (4.14) and (4.15) are equivalent. Suppose we chose to use (4.14): the *value* method on *T* generates another *datacvt*/4 and is subsequently removed by the key constraint CHR rule for *datacvt*/4; the semantic relations osen'/4 produces the primitive relation *olsen*/4; the *value* methods on the three objects that are of type *basic* generates the binding for the three argument in *olsen*/4; and the value method on object *D*' on generates the abducible *datecvt*/4, with date format parameters being the values of modifier *format* in *c_germany* and *c_olsen*. As a result, datecvt(V12,"MM/dd/yy",V10,"MM/dd/yyyy") is added to the constraint store; primitive relation *olsen*/4 is abducted, with last parameter being bound to V12: olsen("DEM","USD",R,V12). And finally, the price data is converted with the exchange rate; because "=" is also an abducible, V11=V3*R is put into the constraint store. Now the constraint store contains the following:

```
{yhfrankfurt("IBM.f",V2,V3,V4,V5,V6,V7,V8,V9),
 datecvt(V2,"d-MMM-yy",V10,"MM/dd/yyyy"),
 "12/20/1998"=<V10, V10 =<"01/10/1999",
 V2 =<"31-DEC-98",
 datecvt(V12,"MM/dd/yy",V10,"MM/dd/yyyy"),
 olsen("DEM","USD",R,V12), V11=V3*R }
```

There are no CHR rules that further process these constraints to simplify them or detect any inconsistency among them. Thus this branch of proof ends successfully. A sub-query can be constructed using the predicates in the constraint store. On backtracking, the second rule (i.e., rule (4.9)) for modifier *currency* in *c_germany* is used, generating constraint "1-JAN-99"=<V2; the rest is the same as in the case when the first rule is used. Similarly, another sub-query can be constructed

from the predicates in the constraint store. The mediated query $\mathrm{MDQ1}_{pr}$[34] consisting of the two sub-queries is shown in Figure 4.8. Certain constraints are renamed, e.g., `V11=V3*R` is rewritten as `V11 is V3*R`.

With the alternative that uses the semantic objects in temporal constraints, a *value* method is used to derive the primitive value of the semantic object in the receiver context. In the example, two rules are used to specify modifier *currency*. When the first rule (4.10) is selected, the constraint store contains the following predicates at the end of the proof branch:

```
{yhfrankfurt("IBM.f",V2,V3,V4,V5,V6,V7,V8,V9),
 datecvt(V2,"d-MMM-yy",V10,"MM/dd/yyyy"),
 "12/20/1998"=<V10, V10 =<"01/10/1999",
 datecvt("31-DEC-98","d-MMM-yy",V13,"MM/dd/yyyy"), V10=<V13, "12/20/1998"=<V13,
 datecvt(V12,"MM/dd/yy",V10,"MM/dd/yyyy"),
 olsen("DEM","USD",R,V12), V11=V3*R }
```

(4.16)

Notice that all primitive values in constraints are in the receiver context *c_usa*. Variable V13 represents the primitive value in context *c_usa* that corresponds to `"31-DEC-98"` in context *c_germany*. Constraint `"12/20/1998"=<V13` is generated by the transitivity rule for ≤ from the two underlined constraints. The transitivity rule is shown in Section 4.3.2.

Similarly, the second rule (4.11) is used on backtracking. The mediated query $\mathrm{MDQ1}_{so}$[35], constructed from the predicates in the constraint store, also consists of two sub-queries and is presented in Figure 4.8.

$\mathrm{MDQ1}_{pr}$

```
answer(V10,V11) :-
        yhfrankfurt("IBM.F",V2,V3,V4,V5,V6,V7,V8,V9),
        datecvt(V2,"d-MMM-yy",V10,"MM/dd/yyyy"),
        datecvt(V12,"MM/dd/yy",V10,"MM/dd/yyyy"),
        V2 =<"31-DEC-98", V10>="12/20/1998", V10=<"01/10/1999",
        olsen("DEM","USD",R,V12), V11 is V3*R.
answer(V10,V11) :-
        yhfrankfurt("IBM.F",V2,V3,V4,V5,V6,V7,V8,V9),
        datecvt(V2,"d-MMM-yy",V10,"MM/dd/yyyy"),
        datecvt(V12,"MM/dd/yy",V10,"MM/dd/yyyy"),
        "1-JAN-99"=<V2, V10>="12/20/1998", V10=<"01/10/1999",
        olsen("EUR","USD",R,V12), V11 is V3*R.
```

$\mathrm{MDQ1}_{so}$

```
Answer(V10, V11):-
    yhfrankfurt("IBM.f",V2,V3,V4,V5,V6,V7,V8,V9),
    datecvt(V2,"d-MMM-yy",V10,"MM/dd/yyyy"),
    "12/20/1998"=<V10, V10 =<"01/10/1999",
    datecvt("31-DEC-98","d-MMM-yy",V13,"MM/dd/yyyy"),V10=<V13,"12/20/1998"=<V13,
    datecvt(V12,"MM/dd/yy",V10,"MM/dd/yyyy"),
    olsen("DEM","USD",R,V12), V11 is V3*R
answer(V10, V11):-
    yhfrankfurt("IBM.f",V2,V3,V4,V5,V6,V7,V8,V9),
    datecvt(V2,"d-MMM-yy",V10,"MM/dd/yyyy"),
    "12/20/1998"=<V10, V10 =<"01/10/1999",
    datecvt("1-JAN-99","d-MMM-yy",V13,"MM/dd/yyyy"),V13=<V10,V13=<"01/10/1999",
    datecvt(V12,"MM/dd/yy",V10,"MM/dd/yyyy"),
    olsen("EUR","USD",R,V12), V11 is V3*R
```

Figure 4.8 Mediated Queries

The two sub-queries of the mediated query generated by either approach (i.e., one performs temporal comparison in source context and uses primitive objects for ground temporal entities, the

[34] Here "pr" standards for "primitive object", indicating that the ground temporal entity is represented by a primitive object in the source context.

[35] Here "so" standards for "semantic object", indicating that the ground temporal entity is represented by a semantic object from the source context.

other performs temporal comparison in receiver context and users semantic objects for ground temporal entities) retrieves and converts data of two time periods: [“December 20, 1998”, “December 31, 1998”] in the first sub-query, [“January 1, 1999”, “January 10, 1999”] in the second sub-query. Each sub-query reconciles a set of semantic differences unique to its respective time period.

As intentional answers to the user query, both mediated queries are correct in that they reconcile the semantic differences that exist between the source and receiver contexts. But problems arise at the query execution step when the mediated queries are evaluated to obtain extensional answers. These problems are caused by the deficiency of the existing COIN that we discuss next.

4.5.2 Deficiency of Existing COIN

The main deficiency in the existing COIN is that it does not have constraint handling rules for temporal constraints and certain source capability constraints. As a result, temporal constraints often remain unsolved in the mediation step, and the certain source limitations are not considered by the mediated query, which lead to the following three problems:

- Lack of Semantic Optimization
- Inefficient Query Plan
- Unexecutable Query

Lack of Semantic Optimization: Undetected Inconsistent Temporal Constraints

Multiple rules are used to define a dynamic modifier in a temporal context. When a user query involves multiple temporal contexts, the multiple rules of each modifier in each context are considered combinatorially during mediation, and each combination produces a sub-query. The number of sub-queries is:

$$\prod_{c}\prod_{i} n_{ic} \tag{4.17}$$

where n_{ic} is the number of rules for the i^{th} modifier in context c.

The user query in the example involves two modifiers (i.e., *format* and *currency*) and two contexts (i.e., *c_germany* and *c_usa*). One rule is used for each modifier in each context except for modifier *currency* that has two rules in context *c_germany*. According to (4.17), there are two sub-queries: $(2\times 1)\times(1\times 1)=2$.

In the example, although both the sub-queries are semantically correct, they contain redundant temporal constraints. In other cases, certain sub-queries may contain inconsistent constraints that are not detected during mediation; a semantically optimized query should eliminate such sub-queries. For example, if the *currency* in the receiver context *c_usa* also changed on the same date as in the source context and primitive temporal values are used in the modifier definitions, the mediated query would consist of four sub-queries that respectively have the following constraints in the body:

```
{…, datecvt(V2,"d-MMM-yy",V10,"MM/dd/yyyy", V2=<″31-DEC-98″, V10=<″12/31/1998″, … }
{…, datecvt(V2,"d-MMM-yy",V10,"MM/dd/yyyy", V2=<″31-DEC-98″, ″01/01/1999″=<V10, … }
{…, datecvt(V2,"d-MMM-yy",V10,"MM/dd/yyyy", ″1-JAN-98″=<V2, V10=<″12/31/1998″, … }
{…, datecvt(V2,"d-MMM-yy",V10,"MM/dd/yyyy", ″1-JAN-98″=<V2, ″01/01/1999″=<V10, … }
```

the two in the middle contain inconsistent constraints that are not detected by the mediator. The number of such sub-queries can be large when there are many modifiers that have time-varying values. In the case of the company financial example in Chapter 3, when the temporal constraints are not solved during mediation, the mediated query for the example user query would consist of 24 sub-queries, out of which 21 sub-queries have inconsistent constraints and should be eliminated.

Inefficient Query Plan

Recall that two steps are involved in query answering in COIN: (1) mediation; and (2) query planning/optimization/execution (POE). The mediation step is responsible for generating the instructions on how to perform appropriate conversions to reconcile semantic differences; the POE step is responsible for generating efficient query plan and executing the plan to obtain data instances. For optimization purposes, certain constraints expressed in the receiver context or in other source contexts may need to be converted into a particular source context. For example, in the first sub-query of MDQ1_{pr}, to minimize or avoid the retrieval of unnecessary data, the constraint `V10=<″12/31/1998″` in the receiver context needs to be converted to the source context so that only data within the date range of [“December 20, 1998”, “December 31, 1998”] is retrieved. But the mediation step did not generate the instructions on how to perform this conversion. Therefore, the only possible plan that can be generated by the POE given the mediated query is to retrieve all data that is on or after December 20, 1998. It is even worse with MDQ1_{so}, with which the only possible plan to be generated is to retrieve all data from the source to find out later that only a small fraction is needed.

This is a COIN weakness not specific to temporal constraints. When a user introduces constraints, ground primitive values in the user context are used and the mediated query only introduces conversions from the source contexts to the receiver context, not the other way around. For example, for a query that retrieves products with `price<10` in the query condition, the mediated query would include all conversions to convert price data in the source context to the receiver context, and then compare the converted values with 10, e.g., `price*0.001*exchangrate<10`. If the source has the capability to evaluate the comparison, it is up to the source's query optimizer to determine how the comparison is evaluated. If it evaluates the arithmetic operation on the left of “<”, then all price data in the sources will be converted before comparing to 10, which is inefficient. If the optimizer is “smart” enough to rewrite the comparison as `price<10/0.001/exchangerate`, and knows to first evaluate the arithmetic operations on the right of “<”, it perhaps can leverage the index on “price” to quickly find only the price data that satisfy the condition without needing to convert all the price data. Had 10 in the receiver context been converted to the source context by the mediator, the source would receive `price<10/0.001/exchangerate`, which is the form that is more likely to be evaluated efficiently.

While a source may be able to manipulate arithmetic operations, it is impossible for a source manipulate other user defined operations. For example, when a user in context *c_usa* issues a query with `Date<″12/31/1998″` to a source whose “Date” attribute is in `″dd-MMM-yy″` format as in context *c_germany*, the mediated query will contain `datecvt(Date, ″dd-MMM-yy″, D, ″MM/dd/yyyy″) and D<″12/31/1998″`, which can be evaluated only by first converting all “Date” data in the source. We should not expect the source to know `datecvt` enough to generate `datecvt(D,″dd-MMM-yy″, ″12/31/1998″, ‶MM/dd/yyyy″) and Date<D.`

Unexecutable Query

The problem of query evaluation becomes much more severe when the source has certain capability constraints. In the example, the source requires that the beginning and ending date must be given in the form of equality on year, month, and day elements. This means that it requires not only that constraints expressed in the receiver context be converted to the source context, but also that constraint ≤ be rewritten as a set of equality constraints. Without such rewriting, the query cannot be executed at all, unless a more capable wrapper is developed for the source.

4.5.3 Extensions with Temporal Constraint Handling Rules

The problems discussed above can be solved by extending the existing COIN mediator with the following capabilities:

- solving temporal constraints in multiple contexts;
- executing conversions for ground temporal values between any contexts during mediation; and
- accommodating certain source capability limitations during mediation.

Before describing the solutions, let us first explain why these capabilities are needed.

From the problems discussed in the previous section, we know that the temporal constraints in sub-queries need to be solved during mediation to eliminate the sub-queries that have inconsistent constraints. We introduced two representation alternatives to defining dynamic modifiers: one uses primitive objects and the other uses semantic objects to represent ground temporal entities. When primitive objects are used in dynamic modifier definitions, the mediated query can contain constraints in multiple contexts. Therefore a multi-context constraint solver is needed. The alternative approach uses semantic objects in modifier definitions to ensure that all temporal values are converted to the receiver context. It seems that this approach only needs a temporal constraint solver in the receiver context, therefore eliminating the need for a multi-context constraint solver. However, any context can be the receiver context in the COIN framework. Thus, we still need a constraint solver that can solve constraints in any defined context.

Traditionally, all conversions are executed for data instances in the query execution step, not in the mediation step. For example, `datecvt("31-DEC-98","d-MMM-yy",V13,"MM/dd/yyyy")` in $MDQ1_{so}$ is a predicate for converting the ground value (i.e., constant) `"31-DEC-98"` to another value represented by variable V13 in a different context. It is not executed during mediation, therefore, the value of variable V13 is not known until the query execution step. Without knowing the value of V13, the constraints involving it and other ground values cannot be solved during mediation. Therefore, when a conversion is for a ground value, the conversion should be executed during mediation instead of being deferred to the query execution step.

In the COIN system, the mediator generates the instructions on necessary data conversions in the form of a mediated query, and the POE component executes the mediated query by marshalling parts of the mediated query to data sources. The POE accommodates certain source limitations by properly sequencing different parts; if a source cannot evaluate certain expressions, the POE can compensate for the source by evaluating the expressions itself. Some limitations of a source may be such that instructions on data conversions are not included in the mediated query, e.g., a source may require a begin date, therefore the begin date specified by the receiver (e.g., "12/20/1998") needs to be converted to the source context (e.g.,"20-DEC-98"), but the existing COIN does not generate conversions from the receiver context to source contexts. In this case, we need to handle such limitations during mediation so that the instructions on how to perform the conversions can be generated by the mediator.

Solution Approach

When ground primitive values in different contexts are posted into the constraint store, constraints over these values cannot be solved until they are converted into the same context. To perform such conversions, the constraint solver needs to have a cumbersome mechanism for keeping track of the contexts of the ground values. This mechanism is not needed when semantic objects are used because all ground values are in the receiver context. Therefore, we choose the representation that uses semantic objects in modifier definitions.

Another requirement for solving the temporal constraints is that the constraint solver should be able to determine a set of binary relations, $<, \leq, =, >, etc.$, between the ground temporal values in all defined contexts. For example, the constraint solver needs to know that in context *c_usa* $"12/31/1998" < "01/01/1999"$ is true, it also needs to know that in context *c_germany* $"31-DEC-98" < "1-JAN-99"$ is true, and so on. Although we can implement these binary relations for each context, it is more efficient to adopt a model (which we call *internal model*) for temporal values in the integration context, map ground temporal values in various contexts to values of this model, and implement the relations only for this model. This model for temporal entities is within the integration context; the mappings of ground temporal values are just bidirectional conversions between the integration context and all the other contexts. The model and the binary conversion can be implemented in a library accessible by the mediator. It is often desirable that the internal model is general and flexible enough to meet the needs of a variety of integration scenarios.

The overall solution approach consists of the following:

- an internal model for time, M_T, with binary relations for temporal values;
- a conversion function, f_{IC}, that provides bidirectional conversions for ground values between the internal model and all defined contexts;
- a set of constraint handling rules that solve the temporal constraints expressed in any given receiver context; and
- a set of constraint handling rules that address source capability limitations.

The internal time model M_T can be implemented in several ways, e.g., using the Calendar class in the Java programming language, or using date and time types supported in various relational database management systems. Another possibility is based on a *temporal system* described in Appendix 4.2.

Below, we provide the general characteristics of the bidirectional conversion and the constraint handling rules for temporal constraints. Later we use examples to illustrate how they may be implemented. Constraint handling rules for source capability limitations are specific to the limitation, thus we illustrate them using an example in the next section.

Definition 4.6 (Bidirectional conversion function f_{IC}) If $t_{IC} : M_T, c : ctx, t : \tau_{\perp}$, $f_{IC}(t_{IC}, c, t)$ is a predicate atom whose procedural reading can be either of the two functions: (1) given a ground primitive value $t_{\perp}$ in context *c*, return the corresponding value t_{IC}; the signature of this function is $ctx \times \tau_{\perp} \rightarrow M_T$; and (2) given a ground value t_{IC} of the internal model, return its corresponding primitive value $t_{\perp}$ in a given context c; the signature of this function is $M_T \times ctx \rightarrow \tau_{\perp}$. ■

When the temporal entity type has only one modifier, the modifier value can uniquely identify a context. In this case, we can use the modifier value in lieu of the context identifier in the definition of f_{IC} and the type symbol *ctx* in the signature declarations should be replaced by the primitive type $\tau_{\perp}$. Predicate f_{IC} is treated as a built-in constraint, thus can be used in the guard of a CHR rule.

Temporal constraints are the conjunction of a set of binary relations over the temporal entity domain in the receiver context. The binary relations include $\{<, \leq, =, \geq, >\}$. In the ACLP framework, these relations are treated as user defined constraints. The constraint = provides a means of variable binding. Constraints ≥ and > are simplified into constraints ≤ and < respectively using the following simplification rules:

$A \geq B \;\Leftrightarrow\; B \leq A$ (simplification for $\geq$)

$A > B \;\Leftrightarrow\; B < A$ (simplification for $>$)

With these rules, we only need to specify CHR rules for ≤ and <

For constraint ≤ , the CHR rules for reflectivity, transitivity, and anti-symmetry given in Section 4.3 are applicable in the temporal entity domain as well. Similar rules can be specified for <. We introduce new rules for temporal constraints that that involve more than one ground temporal values. A predicate *groundTmp*/1 can be used to test if a term is a ground temporal value; since we use strings to represent ground primitive values in the receiver context, we can use *string*/1 for testing purpose. For readability, we continue to use *groundTmp*/1 in the following discussion. There are three types of rules that process the conjunction of two constraints at a time:

- built-in rule – to eliminate the constraint between two ground values
- inconsistency rule – to detect inconsistency of the constraints
- subsumption rule – to eliminate the constraint that is less stringent

All of the rules use the predicate *tr*/3 to convert ground primitive values in receiver context to the integration context and evaluate the given predicate in the integration context. The predicate is defined below:

$$tr(R, T_1, T_2) \leftarrow rcvContext(C), f_{IC}(M_1, C, T_1), f_{IC}(M_2, C, T_2), R(M_1, M_2). \qquad (4.18)$$

where $R \in \{\leq, <\}$, T_1 and T_2 are ground primitive temporal values in the receiver context. The evaluation of predicate $R(M_1, M_2)$ is provided by the internal time model M_T.

A built-in rule directly evaluate the constraint $\leq$ or $<$:

$X < Y \Leftrightarrow groundTmp(X), groundTmp(Y) \mid tr(<, X, Y).$ (built-in)

(4.19)

$X \leq Y \Leftrightarrow groundTmp(X), groundTmp(Y) \mid tr(\leq, X, Y).$ (built-in)

(4.20)

When these rules are applied, the constraint in the rule *head* will be removed if the rule *goal* is evaluated true, otherwise, inconsistency is present and failure is signaled. For example, when constraint `"12/20/1998"=<"12/31/1998"` is posted into the constraint store, rule (4.20) applies; because the rule *goal* `tr(=<, "12/20/1998", "12/31/1998")` will return *true*, `"12/20/1998"=< "12/31/1998"` will be removed from the constraint store.

The inconsistency rule is straightforwardly defined as follows,

$M{<}X, X{<}N \Leftrightarrow groundTmp(M), groundTmp(N), tr(<, N, M) \mid fail.$ (inconsistency)

(4.21)

The intuition of subsumption rules is that when one constraint is more stringent than the other, we can remove the less stringent constraint from the constraint store. For example, given X<"01/01/1999" $\wedge$ X<"10/11/1999", we can remove X<"01/01/1999" because "01/01/1999" <"10/11/1999". We use simpagation rules to specify the subsumption rules:

$X{\leq}M \setminus X{\leq}N \Leftrightarrow groundTmp(M), groundTmp(N) \mid tr(<, M, N).$ (subsumption)

(4.22)

$N{\leq}X \setminus M{\leq}X \Leftrightarrow groundTmp(M), groundTmp(N) \mid tr(<, M, N).$ (subsumption)

(4.23)

The other six rules can be obtained by replacing either one of all of the $\leq$'s with < in the head of rules (4.22) and (4.23).

For temporal entities, we also introduce a composite conversion predicate f_{cc}/4 to convert ground primitive values between two contexts. This predicate is treated as a CHR constraint and is handled by the following rules:

$f_{cc}(T_1, C_1, T_2, C_2) \Leftrightarrow groundTmp(T_1), nonground(T_2) \mid f_{IC}(M, C_1, T_1), f_{IC}(M, C_2, T_2).$

(4.24)

$f_{cc}(T_1, C_1, T_2, C_2) \Leftrightarrow nonground(T_1), groundTmp(T_2) \mid f_{IC}(M, C2, T_2), f_{IC}(M, C_1, T_1).$

(4.25)

Illustrative Example

We illustrate an implementation of the solution approach and show how it addresses the deficiency of the existing COIN use using the Yahoo historic stock price example.

For illustration purpose, we implement the internal time model as a library using a *temporal system* described in Appendix 4.2. The bidirectional conversion f_{IC} is also implemented in the library; f_{IC} is accessible to the mediator using the following predicate:

```
date_string(IM, C, P)
```

where *IM* is the argument for internal model and *C* is the context of primitive value *P*. Argument C is an input; arguments *IM* and *P* cannot be output argument at the same time. Thus, for conversion purposes, the predicate can be used to derive the internal representation given ground primitive value or to derive the ground primitive value from an internal representation.

Since there is only one modifier for temporal entity type in the Yahoo historical quote example, the conversion predicate *datecvt*/4 for the modifier is functionally equivalent to conversion predicate f_{cc}/4. The CHR rules for *datecvt*/4 can be obtained from (4.24) and (4.25) by replacing f_{cc} with *datecvt*, and replacing f_{IC} with *date_string*:

$$\begin{aligned} datecvt(T_1,C_1,T_2,C_2) \Leftrightarrow\ & groundTmp(T_1), nonground(T_2) \mid \\ & date\_string(M,C_1,T_1), date\_string(M,C_2,T_2). \end{aligned} \tag{4.26}$$

$$\begin{aligned} datecvt(T_1,C_1,T_2,C_2) \Leftrightarrow\ & nonground(T_1), groundTmp(T_2) \mid \\ & date\_string(M,C2,T_2), date\_string(M,C_1,T_1). \end{aligned} \tag{4.27}$$

When *datecvt* is added to the constraint store and the guard of one of the above rules is evaluated to true, the body of the rule will be applied, which in effect performs the actual conversion of a ground temporal entity from one context to another, i.e., the *datecvt* is executed during mediation if the guard is true. For example, when `datecvt("31-DEC-98","d-MMM-yy",V13,"MM/dd/yyyy")` is posted into the constraint store, rule (4.26) applies, which removes the *datecvt*/4 constraint and binds V13 to `"12/31/1998"`.

With these constraint handling rules, the constraints generated by the existing COIN can now be further processed. Let us illustrate these transformations using the constraints produced in the proof branch that generates the first half of MDQ1$_{so}$. For reader's convenience, we show these constraints again below:

```
{yhfrankfurt("IBM.f",V2,V3,V4,V5,V6,V7,V8,V9),
 datecvt(V2,"d-MMM-yy",V10,"MM/dd/yyyy"),
 "12/20/1998" =<V10,V10 =<"01/10/1999",
 datecvt("31-DEC-98","d-MMM-yy",V13,"MM/dd/yyyy"), V10=<V13, "12/20/1998"=<V13,      (4.16)
 datecvt(V12,"MM/ddyy",V10,"MM/dd/yyyy"),
 olsen("DEM","USD",R,V12), V11=V3*R }
```

Figure 4.9 illustrates how the constraint store in (4.16) is transformed using the CHR rules presented earlier. The constraint predicate(s) that triggers the CHR rule is underlined; the rule applied is indicated next to the arrow pointing to the state resulting from the application of the rule. Further explanation is given following the figure. We shall point that with ACLP, abduction and application of CHR rules are interleaved: as soon as a constraint is posted into the constraint store, rules applicable to the constraint will be applied immediately before constraint is put into the store by abduction. To simplify explication, in Figure 4.9 we show all constraints posted by abduction at once and explain the applicable CHR rules one at a time.

```
{yhfrankfurt("IBM.f",V2,V3,V4,V5,V6,V7,V8,V9), datecvt(V2,"d-MMM-yy",V10,"MM/dd/yyyy"),
 "12/20/1998" =<V10,V10 =<"01/10/1999",                                                  (4.16)
 datecvt("31-DEC-98","d-MMM-yy",V13,"MM/dd/yyyy"), V10=<V13, "12/20/1998"=<V13,
 datecvt(V12,"MM/dd/yy",V10,"MM/dd/yyyy"), olsen("DEM","USD",R,V12), V11=V3*R }
                                 |
                                 | (4.26) for datecvt/4
                                 v
{yhfrankfurt("IBM.f",V2,V3,V4,V5,V6,V7,V8,V9), datecvt(V2,"d-MMM-yy",V10,"MM/dd/yyyy"),
 "12/20/1998" =<V10, V10 =<"01/10/1999", V10=<"12/31/1998", "12/20/1998"=<"12/31/1998",
 datecvt(V12," MM/dd/yy ",V10,"MM/dd/yyyy"), olsen("DEM","USD",R,V12), V11=V3*R }
                                 |
                                 | (4.22), subsumption
                                 v
{yhfrankfurt("IBM.f",V2,V3,V4,V5,V6,V7,V8,V9), datecvt(V2,"d-MMM-yy",V10,"MM/dd/yyyy"),
 "12/20/1998" =<V10 ,V10=<"12/31/1998", "12/20/1998"=<"12/31/1998",
 datecvt(V12,"MM/dd/yy",V10,"MM/dd/yyyy"), olsen("DEM","USD",R,V12), V11=V3*R }
                                 |
                                 | (4.20), built-in
                                 v
{yhfrankfurt("IBM.f",V2,V3,V4,V5,V6,V7,V8,V9), datecvt(V2,"d-MMM-yy",V10,"MM/dd/yyyy"),
 "12/20/1998" =<V10 ,V10=<"12/31/1998",                                                  (4.28)
 datecvt(V12,"MM/dd/yy",V10,"MM/dd/yyyy"), olsen("DEM","USD",R,V12), V11=V3*R }
```

Figure 4.9 Transformation of Constraint Store with CHR Rules

After predicate `datecvt("31-DEC-98","d-MMM-yy",V13,"MM/dd/yyyy")` is abducted, CHR rule (4.26) for *datecvt* is applied. It uses the built-in predicate *date_string* to perform the necessary conversion. As a result, V13 is bound to `"12/31/1998"` and the predicate `datecvt("31-DEC-98","d-MMM-yy",V13,"MM/dd/yyyy")` is removed from the constraint store.

After constraint `V10=<V13` is abducted into the constraint store, as we explained earlier, the transitivity rule for ≤, which is a propagation rule, generates an extra constraint `"12/20/1998"=<V13`. Meanwhile, because V13 is now bound to ground value `"12/31/1998"`, subsumption rule (4.22) is applicable for constraints `V10 =<"01/10/1999"` and `V10=<V13`, with M being bound to `"12/31/1998"` and N being bound to `"01/10/1999"`. Via predicate $tr(<,M,N)$, the evaluation of $<(M,N)$, which is the prefix equivalent of $M<N$, is true in the integration context. According to the rule, `V10 =<"01/10/1999"` is removed from the constraint store.

Constraint `"12/20/1998"=<V13` is also removed when the built-in rule (4.20) is applied.

The constraint store now contains the following constraints:

```
{yhfrankfurt("IBM.f",V2,V3,V4,V5,V6,V7,V8,V9),
 datecvt(V2,"d-MMM-yy",V10,"MM/dd/yyyy"),
 "12/20/1998" =<V10,V10 =<"12/31/1998",                    (4.28)
 datecvt(V12,"d-MMM-yy",V10,"MM/dd/yy"),
 olsen("DEM","USD",R,V12), V11=V3*R }
```

At this point, the constraints cannot be further simplified, but the source capability limitation has not been considered (i.e., the source requires that V4 through V9 are bound to various date parts of the begin and end dates). To meet the source requirement, the ground primitive values in the constraints involving V10 need to be converted the source context, and the different date parts need to be extracted and equated to variables V4 – V9. This is accomplished using the following CHR rule:

$$\begin{aligned}&yhfrankfurt(T,Q,P,M_1,D_1,Y_1,M_2,D_2,Y_2), datecvt(Q,F_1,A,F_2) \setminus X \le A, A \le Y \Leftrightarrow \\ &\quad groundTmp(X), groundTmp(Y) \mid datecvt(U,F_1,X,F_2), datecvt(V,F_1,Y,F_2), getM(U,F_1,M_1), \\ &\quad getD(U,F_1,D_1), getY(U,F_1,Y_1), getM(V,F_1,M_2), getD(V,F_1,D_2), getY(V,F_1,Y_2).\end{aligned}$$

(4.29)

where the predicates starting with "get" are built-in predicates that bind the last argument according to the first two arguments. For example, getM("1-JAN-99", "d-MMM-yy", M) binds M to "JAN".

When this rule is applied to the constraints in (4.25), the two constraints involving V10 are removed and variables V4 – V9 are bound to appropriate values:

```
{yhfrankfurt("IBM.f",V2,V3,"DEC","20","98","DEC","31","98"),
 datecvt(V2,"d-MMM-yy",V10,"MM/dd/yyyy"),
 datecvt(V12,"d-MMM-yy",V10,"MM/dd/yy"),                          (4.30)
 olsen("DEM","USD",R,V12), V11=V3*R }
```

From these constraints, we can construct a mediated sub-query that is equivalent to the first sub-query in MDQ1 shown in Chapter 3, with the difference only in variable names.

4.6 Discussion on Future Extensions

In developing the temporal extensions, we made certain design choices and assumptions. For example, we chose to represent time-varying semantic using dynamic modifiers and commented on the alternative approach of using dynamic context association in Section 4.4.1. In this section, we discuss two other issues. The first is related to the assumption made in Section 4.4.2 about time-invariant semantics of temporal entities, and the second is related to the mediation-execution two-step process of query answering in relation to query optimization.

4.6.1 Dealing with Time-varying Semantics of Temporal Entities

We proposed to use temporal entities in data sources in the description of time-varying semantics of the other types of data in the sources. This approach requires that the semantics of temporal entities is static, i.e., the modifiers of temporal entity types should have static values. Otherwise, the interpretation of these temporal entities becomes indeterminate, which we explain below.

Proposition 4.1 (*Indeterminacy*) When temporal entities in the context to be described are used in dynamic modifier definition and the temporal entity type has at least one modifier that is also time dependent, the interpretation of the temporal entities become indeterminate because the reasoning process as specified in (4.5) does not terminate. ■

Explanation: We observe that when temporal entities are used in dynamic modifier specification, at least one *value* method on temporal entity is called in the definition; let us use the following predicate to refer to this method call:

$$T: temporalEntity \vdash \quad T[value@C \to T_v]$$

According to the definition of *value* method in (4.4), formula (4.5) is evaluated. It calls the modifier method of all modifiers of the temporal entity type. When at least one of the modifiers is also time dependent, its definition includes a *value* method call on temporal entity as shown above. It is evaluated using formula (4.5). Then the dynamic modifier is encountered again, and the above process continues in an infinite loop. ■

The indeterminacy can also be understood informally with the following example. Suppose a database recording hourly temperature in Boston; it uses local time without stating whether it is EST (Eastern Standard Time) or EDT (Eastern Daylight-saving Time). In the context description, we need to specify this implicit assumption. Informally, we may have the following to describe when it is EDT:

c_boston: EDT is used when T is in ["2005-4-3 2:00", "2005-10-30-2:00"]

The primitive values in the brackets are those in the source context. They can be replaced with semantic objects in context *c_boston*. Either way, there is no way to tell if "2005-4-3 2:00" is EDT or EST.

A similar problem exists for any non-temporal object when its interpretation depends on the relative values of the object. For example, suppose a database containing price information in terms of total cost. The context is such that when the base price is below $50, a $2 shipping charge is included; when the base price is greater than or equal to $50, there is no shipping charge. Then we cannot determine the base price for any price value in the range (50, 52) in this context.

The problem arises from the cyclic inter-dependency illustrated in Figure 4.10: the interpretation of objects of *semanticType1* depends on the modifiers they have; the definition of one of the modifiers *modifier* depends on objects of *semanticType1*.

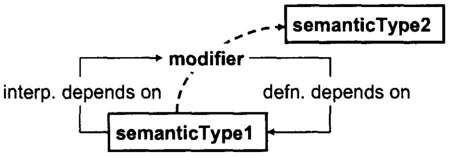


Figure 4.10 Interdependency of a type and its modifier

One way to avoid this circularity is to avoid using the objects of the type for which the modifier is defined; a type with static semantics in the integration context is needed. In the case of time-varying semantics of temporal entities, this means that when defining the dynamic modifiers of temporal entities, a special type of time whose semantics does not change should be used, i.e., an internal time model with static interpretation is needed when the meaning of temporal entities in sources and receivers also changes over time.

4.6.2 Extending Mediation-Execution for Query Optimization

Query answering in COIN involves two steps: (1) mediation by the mediator to generate instructions on all necessary data conversions; and (2) execution by the POE to carry out the instructions as efficiently as possible.

Traditionally, the mediator generates instructions for the purpose of semantic reconciliation, and does not consider particular needs from query optimization and execution point of view. In some cases, certain conversions are needed for the query optimization and query execution, but they are not generated by the mediator. As a result, the mediated query, which contains instruction for semantic reconciliation purposes only, may be difficult to optimize or even impossible to execute.

We illustrated this problem in Section 4.5.3 using the Yahoo historical quote example, where ground temporal values in the receiver context needed to be converted to the source context to meet certain variable binding requirements. We treated such requirements as capability constraints, represented using CHR rules. Since the mediator processes CHR rules, conversions required by the capability constraints are included in the mediated query.

While it may be possible to represent different requirements of POE using CHR rules, it is time consuming to develop and keep these rules up to date, especially when the requirements of POE evolve over time (e.g., query execution statistics accumulated over time may suggest moving the evaluation of an expression from one source to another, therefore requiring different conversions to be generated). Instead of changing the CHR rules manually every time the POE has new requirements, we contemplate a more general solution that allows the POE to request the mediator to generate the instructions on value conversions that are not in the original mediated query but are found necessary by the POE. We explain this using the following example:

Suppose a user query requests data from catalog *a* where price in *a* is less than price in catalog *b*, thus the query contains constraint *a.price<b.price*. The mediated query generated by the existing COIN would include instructions on how to convert price from context *a* to the receiver context as well as instructions on how to convert price from context *b* to the receiver context. If we denote these conversions with $f_{ar}(a.price)$ and $f_{br}(b.price)$, then the mediated query has constraint $f_{ar}(a.price) < f_{br}(b.price)$. The POE would generate a plan that converts all price data in *a* and *b* to the receiver context and applies the constraint to obtain the tuples that meet the criterion. A better plan would either convert data from *b* to *a* (with conversion $f_{ba}(b.price)$), or *a* to *b* (with conversion $f_{ab}(a.price)$), depending on other characteristics of the sources learned by the POE from accumulated query execution statistics, and then apply the constraint at the source. This requires that the POE is able to interact with the mediator to obtain the instructions on the conversions found necessary during query optimization.

4.7 Summary

In this Chapter, we provided the technical details of the existing COIN and the extensions developed for dealing with temporal semantic heterogeneity. Several contributions are made in the description of the existing COIN:

- characterization of *value* and *cvt* methods – the two methods play a critical role of lifting statements from one context to another. For the first time, we provide a clear definition for the *value* method and provide a procedural definition for composition conversion method *cvt*.
- modifier orthogonality – this notion is only briefly covered in previous COIN work. In this chapter, we discover and explain its dependency on the specification of component conversions.
- correspondence between F-logic constructs and FOL predicates – the two previous theses on COIN have different representation styles: the F-logic based representation in Goh (1997) and the FOL based presentation in Firat (2003). With F-Logic only, it is difficult to understand the implementation of COIN; with FOL only, it is difficult to

explain object-orientation that is so central to the COIN approach. The correspondence provided in this Chapter bridges this gap.

The temporal extensions build upon the existing COIN. In "testing" the limit of the existing COIN, we developed a number of important insights:

- constraints in modifier definitions –when ground primitive values in source contexts are used in the constraints in modifier definitions, the constraint store can have constraints in multiple contexts, making it difficult to detect inconsistencies and to semantically optimize the mediated query. To avoid this problem, semantic objects should be used and all constraints should be expressed in the receiver context.
- indeterminacy and the need for types in the integration context – when the semantics of temporal entities also changes over time in a context, semantic indeterminacy arises if temporal values in the context are used for modifier definitions. To avoid indeterminacy, type for time interpreted in the integration is needed. In general, when the semantics of a type is dependent on objects of the type, we need to introduce a special type whose interpretation in the integration context is static.
- query optimization and the need for conversion to source context – to simplify the mediation task, COIN uses a strategy that converts values from source contexts to receiver context. But from the query optimization point of view, sometimes it is better to convert certain values to source contexts. In limited cases, CHR rules can be used to generate the necessary conversions during mediation. We proposed a more general solution where the POE is able to request the mediator to generate conventions found useful by the POE.

We also pointed out the need for further research on dynamic context association and dynamic elevation. The insights from this work and the results from future research will further enahnce the capability and performance of COIN.

Appendix 4.1 Illustration of Query Mediation using *SLD+Abduction* Procedure

The SLD+Abduction procedure is first introduced by Cox and Pietrzykowski (1986); several extensions (e.g., Shanahan, 1989; Decker, 1996) also exist. Goh (1997) extended this procedure and applied it to query mediation. In fact, an example of using the procedure for query mediation is given in Chapter 5 of Goh (1997). Maybe because the example is quite involved, some readers found it difficult to follow. The example we gave in Section 4.5 is modest, but it is complicated with constraint handling using CHR rules.

The goal of this appendix is to use simple examples to illustrate how abduction is used for mediation. We will use the "wet grass" example to explain the difference between deduction and abduction. Then we use a simple mediation to illustrate how abduction is used for query mediation.

In the ensuring discussion, background knowledge about SLD is necessary. This can be found in any standard logic programming text, e.g., Lloyd (1987).

Roughly speaking, the SLD+Abduction procedure makes the following modification to the SLD procedure: when the chosen resolvant (i.e., a predicate or a negation of a predicate to be resolved) is an abducible predicate (which we call abducible for short), it does not attempt to resolve the abducible, instead, it puts the abducible into the constraint store. Each proof branch maintains its own constraint store.

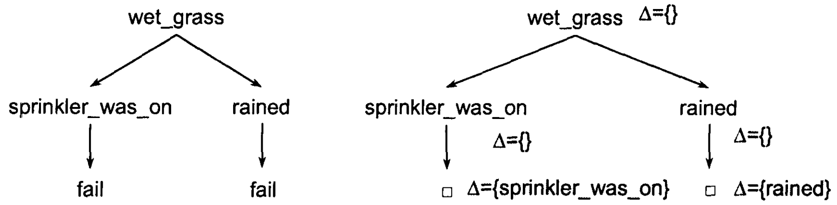
A4.1.1 The "Wet Grass" Example

Below is a logical theory *T* for "wet grass":

wet_grass ← sprinkler_was_on.

wet_grass ← rained.

Let us consider query "← wet_grass". In Figure 4.11 we show the SLD-trees of deductive proof as well as abductive proof. For abduction, the abducibles are A = {*sprinkler_was_on*, *rained*}.



(a) SLD-tree of deduction (b) SLD-tree of abduction

Figure 4.11 SLD-trees for Deduction and Abduction

The SLD-tree of deductive proof is given in Figure 4.11 (a). The deductive answer will be *false* because as seen in the SLD-tree, it cannot resolve either *sprinker_was_on* or *rained* (i.e., they are not in the logical theory).

Figure 4.11 (b) shows the SLD-tree of abduction. The constraint store Δ at each proof step is shown in a shaded box. When *sprinkler_was_on* is selected as resolvant, because it is an abducible, it is put into the constraint store of the proof branch. The other branch is similar. In the end, the constraint stores of all successful leaf nodes are collected to form the abductive answer, which is $\{sprinkler\_was\_on\}$ or $\{rained\}$.

A4.1.2 A Simple Mediation Example

We use a very simple example to illustrate how the basic ideas in the "wet grass" example are applied to query mediation.

Consider a data source containing one relation *r(Company, Profit)*, which records profits of different companies in 1's of USD. A receiver wants to see the profits in 1000's of USD. We call the context associated with the source c1 and the receiver context c2. When the following SQL is issued by the receiver:

```
Select Company, Profit from r;
```

the following mediated query is expected:

```
Select Company, Profit*1/1000 from r;
```

This scenario is illustrated in Figure 4.12, where we show the simple ontology, elevation from the source to the ontology, and context descriptions.

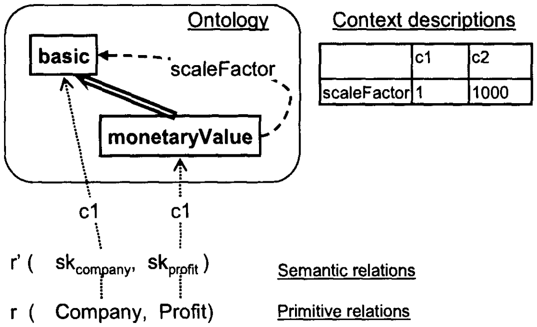


Figure 4. 12 Ontology, Contexts, and Source of Simple Mediation Example

Below are the F-logic rules for elevation (4.31), modifier specifications for contexts c1 and c2 (4.32 and 4.33), and component conversion for modifier *scaleFactor* (4.34).

$$r'(skolem(basic, Company, c_1, 1, r(Company, Profit))$$
$$skolem(moetaryValue, Profit, c_1, 2, r(Company, Profit))) \leftarrow r(Company, Profit).$$

(4.31)

$$O : monetaryValue, scaleFactor(c_1, O) : basic \vdash$$
$$O[scaleFactor @ c_1 \rightarrow scaleFactor(c_1, O)] \leftarrow scaleFactor(c_1, O)[value @ c_1 \rightarrow 1].$$

(4.32)

$$O : monetaryValue, scaleFactor(c_2, O) : basic \vdash$$
$$O[scaleFactor @ c_2 \rightarrow scaleFactor(c_2, O)] \leftarrow scaleFactor(c_2, O)[value @ c_2 \rightarrow 1000].$$

(4.33)

$$M : monetaryValue \vdash$$
$$M[cvt @ scaleFactor, C_r, M_s, M_r, U \rightarrow V] \leftarrow V = U * M_s / M_r.$$

(4.34)

As described in Section 4.2.3, these rules can be translated into Prolog rules straightforwardly. For illustration purpose, we will use these F-logic rules. Therefore, we will omit to show the corresponding Prolog rules. Modifier declaration is also omitted here.

Recall the SQL will be translated to well-formed COIN query in Datalog, which is shown below:

```
answer(V3,V4) :-   r'(V1',V2'), value(V1', c2, V3), value(V2', c2, V4).
```

The body of the query consists of the predicates for the abductive procedure to resolve. The SLD+Abduction procedure will resolve the predicates according to the order they appear. In Figure 4.13 below we show the SLD-tree of the abductive mediation of this query.

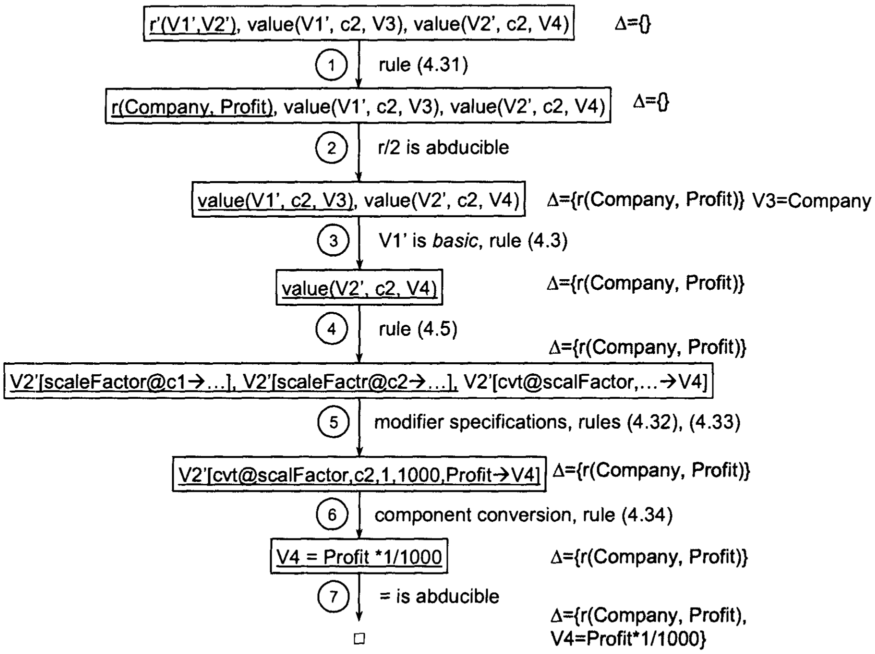

r'(V1',V2'), value(V1', c2, V3), value(V2', c2, V4)
Δ={}
1
rule (4.31)
r(Company, Profit), value(V1', c2, V3), value(V2', c2, V4)
Δ={}
2
r/2 is abducible
value(V1', c2, V3), value(V2', c2, V4)
Δ={r(Company, Profit)} V3=Company
3
V1' is basic, rule (4.3)
value(V2', c2, V4)
Δ={r(Company, Profit)}
4
rule (4.5)
Δ={r(Company, Profit)}
V2'[scaleFactor@c1→...], V2'[scaleFactr@c2→...], V2'[cvt@scalFactor,...→V4]
5
modifier specifications, rules (4.32), (4.33)
V2'[cvt@scalFactor,c2,1,1000,Profit→V4]
Δ={r(Company, Profit)}
6
component conversion, rule (4.34)
V4 = Profit *1/1000
Δ={r(Company, Profit)}
7
= is abducible
Δ={r(Company, Profit),
V4=Profit*1/1000}
□

Figure 4.13 SLD-tree for Query Mediation of the Simple Example

In Figure 4.13, for each step we underline the predicate to be resolved, show the constraint store, label the step with a number, and provide a brief explanation next to the number. We only show variable bindings useful for query construction.

In step 1, semantic relation r' is resolved to r/2 using the elevation rule (4.31).

In step 2, because r/2 is an abducible, it is removed and put into the constraint store.

In step 3, *value* predicate corresponds to the *value* method defined in Section 4.2.2. Because semantic object V1' is of type basic, it is resolved according to (4.3) and V3 is bound to "Company" attribute of r/2.

In step 4, similar to step 3, but composite conversion for semantic object V2' is introduced according to (4.5). There is only one modifier, which is *scaleFactor*. Here we only show the predicates for modifier specifications and the component conversion.

In step 5, to save space, we actually combine two steps that resolve the predicates for modifier specifications in contexts c1 and c2. Modifier values are bound to modifier variables in the next resolvant.

In step 6, the component conversion is resolved using the conversion rule for *scaleFactor* (4.34).

In step 7, because "=" is an abducible, it is removed and put into the constraint store. An empty resolvant indicates the derivation is successful.

At the end of this successful derivation, a mediated query can be constructed using the variable binding and the predicates in the constraint store:

```
    answer(Company,V4) :-      r(Company, Profit), V4 = Profit*1/1000.
```

We can then translate above mediated Datalog query to obtain the mediated SQL.

Appendix 4.2 Time Representation using Temporal System

The *temporal system* introduced by Nogueira et al. (2004) provides a flexible representation of temporal entities. Below we describe the system and illustrate its use as an internal time model. We adapt their definition for temporal system as a Cartesian product of non-empty subsets of integers:

Definition 4.7 *TS* is a temporal system $\Leftrightarrow \exists k_1,\cdots,k_n \in \mathrm{N}, \exists TU_1,\cdots,TU_n \subset \mathrm{Z}$: $TS = TU_1 \times \cdots TU_n$ and $|TU_i| = k_i, i = 1,\cdots n$. TU_i is called a temporal unit. A *time point* in *TS* is a tuple $<x_1, \ldots, x_n> \in TS$. ■

In the above definition, N is the set of natural numbers, and Z is the set of integers. The following three examples illustrate how various temporal systems can be defined.

Example 4.1 (Annual system) An annual system *AS* with only *year* in Gregorian calendar can be defined as *AS* = *Year*, where *Year* = {..., -1, 1,...}, and *–i* is interpreted as the year *i BC*, $i \in \mathrm{N}$. A tuple such as <1999> represents the *year* 1999.

Example 4.2 (24-hour clock) The 24-hour clock system $\mathcal{H}$ with a granularity of minute can be defined as $\mathcal{H}$ = *Hour* × *Minute*, where *Hour* = {0, ..., 23} and *Minute* = {0, ..., 59}. A tuple such as <12, 35> represents 12:35 in the afternoon.

Example 4.3 (Gregorian Date) A Gregorian date system $\mathcal{D}$ can be defined as $\mathcal{D}$ = *Year* × *Month* × *Day*, where Year is as specified in Example 4.1 *Month* = {1, ..., 12}, and *Day* = {1, ..., 31}. In addition, other constraints that ensure the validity of a tuple can be provided as part of the definition, e.g., number of days in a month or a leap year. A example tuple <1998, 12, 31> represents the day "December 31, 1998".

Example 4.4 (Date and time) A system *DT* with Gregorian date, UTC time, and a granularity of second can be defined as *DT* = *Year* × *Month* × *Day* × *Hour* × *Minute* × *Second*, where *Year*, *Month*, *Day*, *Hour* and *Minutes* are specified as in the previous examples, and *Second* = {0, ..., 60}. In addition, other constraints that ensure the validity of a tuple can be provided as part of the definition, e.g., number of days in a month, leap year, leap second. A tuple such as <1998, 12, 31, 23, 59, 60> represents the last (leap) second in the year 1998.

Nogueira et al. (2004) introduce two basic binary relations between time points: *equal* and *before.* They are defined on the premise that the temporal units in a temporal systems are ordered in such a way that the granularity of TU_i is greater than TU_{i+1}. Each of the two relations has a *weak* form and a *strong* form: the strong form requires the points are from the same temporal system, and the weak form only requires that the first temporal unit of each point is of the same granularity. We introduce the weak form, which is more general than the strong form.

Definition 4.8 (Weak *equal*) Let $\vec{x} =< x_1,\cdots x_m >$ and $\vec{y} =< y_1,\cdots,y_n >$ be two time points and *p* = *min*(*m*, *n*), $\forall i \in [1,p] : \vec{x} = \vec{y} \Leftrightarrow x_i = y_i$.

Definition 4.6 (Weak *before*) Let $\vec{x} =< x_1, \cdots x_m >$ and $\vec{y} =< y_1, \cdots, y_n >$ be two time points, $\exists j \leq \min(m, n) \forall i \in [1, j]$ $\vec{x} < \vec{y} \Leftrightarrow x_i = y_i \wedge x_j < y_j$.

Other (weak) relations can be defined using the two basic relations. An application specific time model consists of a temporal system or multiple systems with the same highest granularity and a set of binary temporal relations over time points in the temporal system(s). Such a model can be straightforwardly implemented in Prolog where a tuple is represented as a list.

"Never believe an observation until it is confirmed in theory."
– Francis Crick

Chapter 5

The Flexibility and Scalability of COIN

The problem of semantic interoperability is not new, and people have tried to achieve semantic interoperability in the past using various approaches. These approaches have sometimes been reasonably successful in limited applications, but have proven either very costly to use, hard to scale to larger applications, or both. COIN is an attempt to provide an approach that reduces the cost and complexity of achieving semantic interoperability.

In this chapter, we provide a systematic evaluation of the COIN approach in terms of its flexibility and scalability. We first briefly discuss our evaluation framework. Then we give an overview of the traditional and still commonly practiced approaches to dealing with semantic interoperability. In the last section, we characterize and compare the traditional approaches with the COIN approach. This systematic analysis allows us to formally show that the COIN approach is flexible and scalable.

5.1 Evaluation Framework

Although there has been extensive work to develop various semantic interoperability solutions, little has been done to systematically evaluate these solutions. Previous work on interoperability evaluation tends to be either ad-hoc or too narrowly focused. For example, in some of the surveys (Wasche, 2001; Kashyap and Sheth, 2000), we can only find informal evaluations using ad-hoc and inconsistent criteria. While El-Khatib et al. (2000) and Hammer et al. (2005) develop testing cases useful for consistent evaluation of different information integration approaches, their focus is on testing different approaches' capability of reconciling certain kinds of heterogeneity, not on evaluating the amount of effort required to implement the approaches.

Below, we present our preliminary *task-based* evaluation framework. It is motivated by Rosenthal et al. (2001), where eight tasks of data integration are identified. In our framework, instead of having the eight fine-grained tasks as in Rosenthal (2001), we group them into three major tasks for achieving semantic interoperability amongst disparate systems[36]:

[36] Recall that we use system, data source, and receiver application interchangeably.

- *Knowledge acquisition*: to acquire knowledge about all systems engaging in information exchange.
- *Implementation*: to encode the acquired knowledge, and implement necessary conversions as a set of instructions on how to reconcile semantic differences between systems.
- *Execution*: to fulfill a given information exchange task by determining and executing all necessary conversions to obtain data instances.

There are two kinds of knowledge. The first is about the *similarity* of the data elements in different systems so that correspondences amongst them can be established. For example, "price" in system *A* and "charge" in system *B* are similar and correspond to each other. The second is what we call *context knowledge* that describes the subtle *differences* amongst similar data elements. For example, "price" in system *A* may be in USD, not including taxes or shipping charges, while "charge" in system *B* may be in thousands of South Korean Won, including taxes and shipping charges.

The acquisition of both kinds of knowledge is a labor-intensive process. There has been extensive research that develops automatic schema matching algorithms (Rahm and Bernstein, 2001) to reduce the amount of human effort required for identifying correspondences, thus reducing the effort for acquiring *similarity* knowledge. Several commercial tools such as Cupid from Microsoft (Madhavan et al., 2001), Clio from IBM (Miller et al., 2001), and ProfileStage from IBM are also available for this purpose. Even though these algorithms and tools are helpful, substantial human involvement is still needed to verify the correspondences suggested by software. More importantly, these algorithms and tools are only good at identifying similarities; they cannot identify the differences amongst similar data elements, nor can they infer how the differences can be reconciled. Thus, knowledge acquisition will continue to be a labor-intensive process. A recent survey (Seligman et al., 2002) on time distribution among different tasks of data integration projects shows that nearly 30% of the time is spent on knowledge acquisition.

It is even more labor-intensive to encode the acquired knowledge and implement all necessary conversions. There are different approaches to implementing the conversion. With some of the traditional approaches, which we will discuss latter, this process can account for up to 70% of the costs of an integration project (Doughty, 2004).

The last task (i.e., execution) is carried out by software, which executes appropriate conversions to convert data instances. It is desirable that the algorithm of the software is efficient in terms of its time and space complexity (e.g., it is desirable that the algorithm always terminates in polynomial time with a correct answer).

This task breakdown allows us to separate two aspects to consider when evaluating semantic interoperability approaches. One concerns *human efforts* involved, the other concerns the *performance* of the software algorithm. Intuitively, the two aspects distinguish between "how hard humans have to work" and "how hard computers have to work" to achieve semantic interoperability. Apparently, tradeoffs can be made between "human efforts" and "computer efforts". For example, one alternative to achieving semantic interoperability is to develop a global standard and have all systems implement the standard, in which case all systems are semantically interoperable by design.

With this approach, most of the work is upfront human effort on developing and implementing the standard.

Different criteria can be developed for evaluating approaches to fulfilling different tasks. In this chapter, we will develop the criteria for evaluating various approaches to fulfilling the second task, i.e., *implementation*.

The knowledge acquisition task generally can be decoupled from the other two tasks. The efforts required for knowledge acquisition should not vary among interoperability approaches. However different methodologies can be adopted for knowledge acquisition. Future research should develop the criteria for evaluating the effectiveness of different methodologies for knowledge acquisition.

The criteria for evaluating algorithms are well established. Work has been done elsewhere to evaluate the time complexity of algorithms useful for achieving semantic interoperability. For example, Halevy (2001) and Lenzerini (2002) review the complexity results of query rewriting algorithms, used by data integration approaches that rely on the view mechanism of databases. Eiter et al. (1997) presents their complexity results of abductive reasoning, which we use in the COIN approach.

In the next section, we describe several traditional approaches to semantic interoperability. Following that, we present the criteria for evaluating different approaches to fulfilling the implementation task, and use the criteria to evaluate the traditional approaches as well as the COIN approach.

5.2 Traditional Approaches to Semantic Interoperability

We consider semantic interoperability as the capability of meaningfully exchanging information amongst a set of sources and receivers. This can be achieved in a number of ways, e.g., by developing and maintaining all necessary conversions for reconciling semantic differences, or by adopting certain standards that would eliminate semantic heterogeneity in the sources all together (assuming the standard can be enforced). In this section, we discuss three widely practiced approaches to achieving semantic interoperability.

5.2.1 Brute-force Data Conversions (*BF*)

The *BF* approach directly implements all necessary conversions in hand-coded programs. With N data sources and receivers, $N(N\text{-}1)$ such conversions need to be implemented. When N is large, these conversions become costly to implement and very difficult to maintain. This is a labor-intensive process; as mentioned earlier, nearly 70% of integration costs come from the implementation of these data conversion programs (Doughty, 2004)

This approach might appear sufficiently inefficient that one might be surprised at how common it is. The reason is that usually the conversion programs are written incrementally – each individual conversion program is produced in response to a specific need. Writing "only one conversion program" does not seem like a bad idea – but over time, this process continues toward the $N(N\text{-}1)$ conversion programs that must be maintained and updated.

A possible variation of the approach is to group sources that share the same set of semantic assumptions into one context. Much like the lifting axioms in McCarthy and Buvač (1997) and Guha (1995), conversion programs need to be implemented for each pair of the contexts. The number conversions is $n(n-1)$, with n being the number of contexts. The approach allows multiple sources in the same context to share the same conversion programs, thus it has the potential of reducing the number of conversion programs. However, it does require the overhead of establishing and maintaining the correspondences between the sources and the contexts[37]. We refer to the original approach and this variation as BF_S and BF_C, respectively. These approaches and two other standard-based approaches discussed below are illustrated schematically in Figure 5.1 and elaborated further in the remainder of the section.

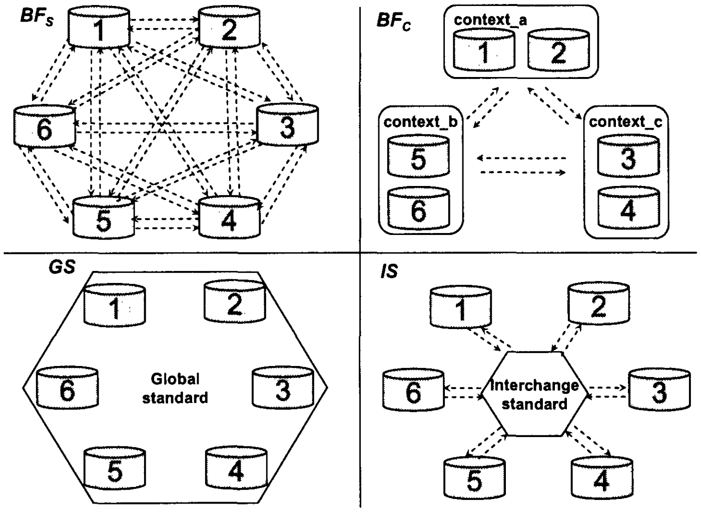


Figure 5.1 Traditional approaches to Semantic Interoperability

Our illustration in Figure 5.1 uses six data sources, some of which share the same context (i.e., they do not have semantic differences in data). A dotted arrow represents a program that implements all necessary conversions that convert data in the source for the receiving system to which the arrow points.

5.2.2 Global Data Standardization (*GS*)

In a collaborative environment, if the parties engaging in information exchange could develop and maintain a single data standard that defines a set of concepts and specifies the corresponding representation, all semantic differences would disappear and there would be no need for data conversion. Unfortunately, such standardization is usually infeasible in practice for several reasons.

Often there are legitimate needs for having different definitions for concepts, and storing and reporting data in different formats. For example, while the metric units are used in most parts of the

[37] COIN has similar requirement, but COIN provides a systematic method of describing contexts and associating sources with contexts. Other features in COIN significantly improve context and conversions sharing.

world, people in the U.S. still use and find it convenient to use English units[38]. Since most integration and information exchange efforts involve many existing systems, agreeing to a standard often means someone has to change his/her current implementation, which creates disincentives and makes the standard development and enforcement extremely difficult. This difficulty is exacerbated when the number of the data elements to be standardized is large. For example, in 1991, the Department of Defense (DoD) initiated a data administration program that attempted to standardize nearly one million data elements[39]; by the year 2000, it only managed to register 12,000 elements, most of which were infrequently reused. After a decade of costly effort, the DoD realized the infeasibility of having a single standard and switched to an alternative approach that allows different communities of interest to develop their own standards (Rosenthal, et al., 2004).

The latter approach by the DoD manifests the reality of standard development, i.e., there are often competing or otherwise co-existing standards. For example, there are multiple standards for airport codes and for country codes. Different systems can potentially choose different standards to implement. Thus, in most cases, we cannot hope that semantic differences will be standardized away; data conversion is inevitable.

5.2.3 Interchange Data Standardization (IS)

The data exchange systems sometimes can agree on the data to be exchanged, i.e., standardizing a set of concepts as well as their interchange formats. The underlying systems do not need to store the data according to the standard; it suffices as long as each data sender generates the data according to the standard. That is, this approach requires that each system have conversions between its local data and an interchange standard used for exchanging data with other systems. Thus each system still maintains its own autonomy. This is different from the global data standardization, where all systems must store data according to a global standard. With N systems exchanging information, the Interchange Standardization approach requires $2N$ conversions. The IS approach is a significant improvement over the brute-force approach that might need to implement conversions between every pair of systems.

Similar to BF_C approach, there could be an IS_C approach that groups systems that share the same context together to share conversion programs. With n unique contexts, the IS_C approach requires $2n$ conversions. When there are no systems sharing the same context, IS_C is the same as the original *IS* approach. In the ensuing discussion, we mainly focus on the original *IS* approach.

This approach has been used for business transactions, e.g., Electronic Data Interchange (EDI) and various Business-to-Business (B2B) trading standards. In the military setting, the U.S. Message Text Format (MTF) and its NATO equivalent, Allied Data Publication-3, have over 350 standard messages that support a wide range of military operations. This standard has been used for over 50

[38] Arguments for metric units can be found at http://www.metric4us.com/; the site also lists sites with arguments against metric unit. The NIST also has a program (http://ts.nist.gov/ts/htdocs/200/202/mpo_home.htm), promoting the adoption of metric unit in the U.S.

[39] Since it was necessary to accommodate existing systems with different contexts, there were data elements for *fuel-load-in-liters* and *fuel-load-in-gallons*, without explicit acknowledgement of the relationship between these elements.

years and currently an XML version is being developed (Miller et al., 2003). As a recent example, the DoD standardized exchange format of weather related data, which consists of about 1,000 attributes[40]. This standard has been successfully used by several systems that exchange weather data (Rosenthal, et al., 2004). Similarly, the XML-based Cursor-On-Target (COT) standard, which consists of 3 entities and 13 attributes, has been used successfully by over 40 systems to exchange targeting information (Rosenthal, et al., 2004).

Although this approach has certain advantages, e.g., local autonomy and a smaller number of conversions required, it also has several serious limitations. First, after all parties have had a clear understanding about the domain and decided what data elements should go into the standard (which is part of knowledge acquisition effort), they have reach an agreement on the data definition and data format. Reaching such an agreement can be a costly and time consuming process. It took the DoD five years to develop the interchange standard for weather data[41]. Furthermore, in many cases it is difficult to foresee what data needs to be exchanged or how the requirements might change over time, which makes it inappropriate to have a fixed standard. When the information of interest is not specified in the standard, ad-hoc conversions have to be implemented. Besides, any change to the interchange standard affects all systems and the existing conversion programs. And lastly, the approach can involve many unnecessary data conversions. For example, when the currency for price of the interchange standard is US dollars, and a source and receiver use South Korean Won, price in the source will be first converted to US dollars, then converted back to South Korean Won. Such unnecessary conversions can be avoided in the BF approach.

5.2.4 Summary of Traditional Approaches

Each of the three traditional approaches has certain drawbacks that make them inappropriate for integrating information from a large number of data sources. These weaknesses are summarized below:

- Brute-force data conversions (*BF*): this requires a large number of hand-written conversions that are difficult to maintain.
- Global Data Standardization (*GS*): it is costly and sometimes impossible to develop a global standard. In addition to legitimate reasons of having multiple standards, there are technological difficulties and organizational resistance to a single standard.
- Interchange Standardization (*IS*): the standard is static, only suitable for routine data sharing and it still requires a large number of hand-written conversions.

In addition, these approaches lack flexibility to adapt to changes because the data semantics is hard-coded in the conversions for *BF*, in the standard in *GS*, and in both the conversions and the standard in the case of *IS*.

[40] This is also known as the "communities of interests" approach, where organizations come together to develop standards for particulars domains in which they share common interests. These standards are equivalent to interchange standards when the organizations provide translations between the conventions in existing systems and the standards.

[41] Although now used by several DoD systems, it has not been adopted by all DoD legacy systems nor non-DoD systems (e.g., in private sector or foreign governments) that may need to interoperate with DoD systems.

The COIN approach overcomes these shortcomings by automating the code generation process. In addition, as discussed in Chapter 4, the concepts in the shared ontology in COIN are high level concepts that require minimal commitment; the parties engaging in information interchange still have the freedom to further refine the concepts using context descriptions. As we will see next, this approach allows for much greater flexibility and scalability.

In the preceding discussions, we have used the term *conversion* quite loosely. In the rest of the discussion, we differentiate four types of conversions:

(1) a *component conversion* is defined for a modifier between two modifier values in the COIN approach; it reconciles one aspect of semantic differences of a single data type (e.g., a conversion that only reconciles differences in currency of *profit*);

(2) a *compound conversion* reconciles all aspects of semantic differences of a single data type (e.g., a conversion that reconcile differences in all aspects of *profit*, e.g., definition, scale factor, as well as currency);

(3) a *composite conversion* combines multiple component conversions to reconcile the semantic differences of all data types involved in a specific user query, which may access multiple data sources (e.g., a conversion that reconciles differences in profit, sales, and number of employee, supposing these data types are requested in a user query); and

(4) a *comprehensive conversion* reconciles the semantic differences of all data types in two systems or between a system and an interchange standard (e.g., a conversion that reconciles the differences in all data types of two systems, which could have dozens or even hundreds of data types).

These conversions are listed in an increasing order in terms of the amount of effort required to develop and maintain them.

5.3 Flexibility and Scalability – Comparison of Different Approaches

In this section, we compare the traditional approaches with the COIN approach to show that the COIN approach has greater flexibility and scalability. When there are no sources that share the same context, then BF_C is the same as BF_S; therefore, in the rest of the analysis, we will not differentiate the two types of *BF* approach.

5.3.1 Flexibility Analysis

We use the term flexibility to collectively refer to two important capabilities of accommodating changes: *adaptability* is the capability of accommodating changes, such as semantic changes within a data source with minimal effort; *extensibility* is the capability of adding (or removing) data sources with minimal effort.

In this Thesis, we do not develop protocols for directly quantifying the amount of effort. Instead, we use the number of conversions that need to be developed and maintained as a proxy for this effort. Recall that we differentiate a comprehensive conversion from a component conversion, and a comprehensive conversion requires significantly more effort to develop and maintain than a component conversion.

Flexibility of Traditional Approaches

The Brute-Force (*BF*) data conversion approach has the least flexibility. With *N* sources, a change in any source would potentially affect 2(*N*-1) comprehensive conversions, i.e., *N*-1 comprehensive conversions converting from the changing source to the other sources and vice versa. Adding or removing a source has similar effects.

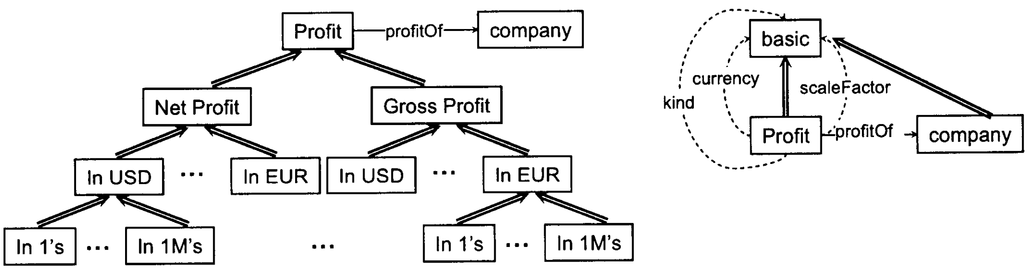
This problem is somewhat reduced with the Interchange Standardization (*IS*) approach. But it still requires re-programming to handle changes, which can be tedious and error-prone. Especially when the interchange standard is changed, then the 2*N* comprehensive conversions need to be updated to accommodate the change. Adding or removing a source requires adding or removing two comprehensive conversions.

The Global Data Standardization (*GS*) approach also lacks flexibility because any change requires agreement by all participants and this is a difficult and extremely time consuming process. In addition, all systems need to implement the changes, which sometimes cause disruption in operations. Only a source that implements the standard can be added to exchange information with existing sources.

Flexibility of the COIN Approach

The COIN approach requires a shared ontology, which is expected to be easier to develop and agree on because it is not as detailed as the agreements needed in the GS and IS approaches. More importantly, the COIN approach does not require an agreement on detailed definitions of the concepts in the shared ontology and their corresponding representations in sources and receivers; disagreements are allowed and are captured as context descriptions by assigning values to modifiers in the ontology.

This is an important distinction. As noted earlier, the knowledge acquisition task is generally the same for all these approaches, and COIN *does* require that the detailed definitions of the concepts used in the sources and receivers be captured. However, COIN captures these detailed definitions in the (localized) context descriptions. This both eliminates the need for global agreements on these detailed definitions, and provides a modular way of capturing the detailed knowledge required to define the conversions between these detailed definitions. This characteristic of the COIN approach to ontology is illustrated in Figure 5.2, using a simple ontology of company profit.



(a) a fully specified ontology of *profit* (b) a COIN ontology of *profit*

Figure 5.2 Fully Specified Ontology v. COIN Ontology.

The fully specified ontology in Figure 5.2 contains possible variations of the concept *profit*, organized in a multi-level and multi-branch hierarchy. Each leaf node represents a most specific *profit* concept. For example, the leftmost node at the bottom represents a *profit* that is a "*net profit* in 1's of USD". In contrast, the COIN ontology contains only concepts in higher levels, further refinements of these concepts do not appear in the ontology; rather, they are specified using modifiers.

Compared with the fully specified approach, the COIN approach has several advantages. First, a COIN ontology is usually much simpler, thus easier to manage. Second, it facilitates consensus development, because it is relatively easier to agree on a small set of high level concepts than to agree on every piece of detail of a large set of fine-grained concepts. And more importantly, a COIN ontology is much more adaptable to changes. For example, when a new concept "*net profit* in billions of South Korean Won" is needed, the fully specified ontology needs to be updated with insertions of new nodes. The update requires the approval of all parties who agreed on the initial ontology. In contrast, the COIN approach can accommodate this new concept by adding new context descriptions without changing the ontology.

Regardless of the approach taken for ontology development, conversions need to be specified to convert data between variations of the same higher level concept. The approach taken, however, may affect the provision of the conversions. For example, the fully specified approach would likely invite a provision of pair-wise conversions amongst the leaf nodes, leading to a solution similar to the *BF* approach. As defined earlier, such conversions are *compound conversions*, often combining several component conversions, e.g., the conversion from the leftmost node to the rightmost node on the leaf level would reconcile the differences in definition (i.e., *net profit* and *gross profit*), currency (i.e., USD and EUR), and scale factor (i.e., 1 and 1M).

In contrast, conversions in the COIN approach are much finer-grained to allow for high reusability. They are *component conversions* defined for each modifier in the ontology. We will see in the next section that the number of component conversions is not directly dependent on the number of sources. All pair-wise c*omprehensive conversions* needed in *BF* and *IS* approaches are dynamically generated from these component conversions. In most cases, changes in existing source and the addition of new sources are accommodated by updating and adding declarative rules, which is much simpler than updating comprehensive conversions. Therefore, the COIN approach has greater flexibility.

Specifically, changes in existing sources and addition of new sources can be accommodated by one of the following responses in the COIN approach, listed roughly in an ascending order of effort involved:

(1) updating or adding modifier definitions;
(2) (1) + updating or adding component conversions;
(3) (2) + adding new modifiers, their definitions, and their component conversions;
(4) (3) + adding semantic types; and
(5) redesigning the ontology.

Responses (1)-(4) involve the updating of declarative rules or some programming of component conversions implemented as callable functions. For example, the concept "*net profit* in billions of South Korean Won" can be accommodated by response (1) if the existing component conversions are parameterized (we discuss parameterized component conversions in the next section). Only when the changes are such that a redesign of the ontology is needed will the COIN approach require significant effort, in which case the traditional approaches need to rewrite all comprehensive conversions or redesign a global standard.

5.3.2 Scalability Analysis

Scalability refers to the capability of achieving and maintaining semantic interoperability with the amount of effort not growing dramatically with the number of participating sources and receivers. As we know from earlier discussion, most of the effort is in creating and maintaining the conversions. Therefore, we use the number of conversions that need to be implemented and maintained over time as a measurement of the scalability.

According to our measurement, the *GS* approach is scalable because it does not need any conversion at all. But if we consider the standard development process as a semantic reconciliation process performed by human designers, significant human efforts are involved in the process although such efforts are difficult to quantify. In practice, it is often impossible to establish a global standard when there are a large number sources participating in information interchange. The rest of this analysis will focus on the other approaches.

We have informally discussed the scalability of the two other traditional approaches. We summarize them below, followed by a detailed analysis on the scalability of the COIN approach.

Proposition 5.1 (*Scalability of BF*) The number of comprehensive conversions needed for achieving semantic interoperability among *N* sources with the *BF* approach is $N(N\text{-}1)$, which is $O(N^2)$. ■

Explanation: Each source needs to perform data conversions with the other N-1 sources; there are N sources, thus a total of $N(N\text{-}1)$ comprehensive conversions need to be in place to ensure pair-wise information exchange, which is $O(N^2)$. ■

Proposition 5.2 (*Scalability of IS*) The number of comprehensive conversions needed for achieving semantic interoperability among *N* sources with the *IS* approach is $2N$, which is $O(N)$. ■

Explanation: For each source there is a conversion to the standard and another conversion from the standard to the source. There are *N* sources, so the total number of conversions is $2N = O(N)$. ■

Proposition 5.3 (*Scalability of COIN*) Suppose the semantic heterogeneity of N sources can be described using an ontology that has *M* modifiers with the i^{th} modifier having n_i unique values, $1 \le i \le M$. The number of component conversions needed for achieving semantic interoperability with the COIN approach is $\sum_{i=1}^{M} n_i(n_i - 1)$, which is $O(Mn_k^2)$, or $O(n_k^2)$ if *m* is fixed, where $n_k = \max\{n_i \mid 1 \le i \le M\}$. ■

Explanation: As seen in Chapter 4, component conversions in COIN are defined for each modifier, not between pair-wise sources. Thus the number of component conversions depends only on the variety of contexts, i.e., number of modifiers in the ontology and the number of distinct values of each modifier. In worst case, the number of component conversions to be specified is $\sum_{i=1}^{M} n_i(n_i - 1)$. This is because in worst case for each modifier, we need to write a conversion from a value to all the other values and vice versa, so the total number of conversions for the i^{th} modifier is $n_i(n_i\text{-}1)$. n_k is the maximum of the number of unique values of all modifiers. When both M and n_k approach infinity, $\sum_{i=1}^{M} n_i(n_i-1) = O(Mn_k^2)$; for any given ontology with finite number of modifiers, the number of component conversions is $O(n_k^2)$, i.e., $\forall M, \sum_{i=1}^{M} n_i(n_i-1) = O(n_k^2)$. ■

Recall from Chapter 4 that in a component conversion atom $t[cvt@m,t_c,t_{ms},t_{mr},t_{\perp 1} \rightarrow t_{\perp 2}]$, t_{ms} and t_{mr} are terms representing modifier m's values in the source context and the receiver context, respectively. A term can be a constant or variable. Certain component conversions use variables for the two terms; such component conversions can convert for all possible valid modifier values. For example, (4.13) and (4.14) for date format and currency conversions are such conversions. We call such component conversions *parameterized component conversions*. If the component conversions of a modifier are parameterizable, we only need to specify one parameterized component conversion for the modifier.

A collection of externally defined functions called by the component conversions can be seen as a conversion library. The COIN approach takes the advantage of this library by automating the selection, composition, and parameter instantiation of the functions in the library.

With the *BF* and *IS* approaches, a library with each function specializing a component conversion often does not exist. For example, if there is a need to convert *profit* data from *net profit* in billions of South Korea Won to *gross profit* in 1's of USD, the chances are that this will be implemented as one *compound conversion* using some program, which is rarely reused elsewhere. The program actually implements several component conversions, e.g., a parameterized conversion for reconciling currency differences, one for scale factor, and one or more for profit definition. These component conversions in the COIN approach are reused to compose many other composite conversions as needed. Even if such a library existed, the decisions on what functions should be called with what parameters are hard coded in the *BF* and *IS* approaches as oppose to being automatically generated in the COIN approach.

When parameterization is difficult, we can exploit certain relationships among component conversions. In cases where a set of component conversions essentially implement a set of inter-related equations, COIN can use symbolic equations solvers to generate conversions that are not explicitly specified (Firat et al., 2002; Firat, 2003). For example, suppose we have three definitions for price: (A) base price, (B) tax included price, and (C) tax and shipping & handling included price. This can be modeled by using a *price* concept in the ontology, together with a modifier for *price* having a distinct value for each of these definitions. With known equational relationships among the three price definitions[42], and only two conversions (1) from base_price to base_price+tax (i.e., A to B) and (2) from base_price+tax to base_price + tax + shipping & handling (i.e., B to C), the COIN mediator can generate the other four conversions automatically (A to C and the three inverses). Thus the number of conversion definitions for a modifier can be reduced from $n(n\text{-}1)$ to $n\text{-}1$, where n is the

[42] Such relationships can be found in business rules that specify, say, how to compute taxes from a tax-included price. The component conversions represent such business rules.

number of unique values of the modifier. The requirement is that the component conversions for the modifier are invertible, i.e.,

$t[cvt@m,t_c,t_{ms},t_{mr},t_{\perp 1} \rightarrow t_{\perp 2}] \leftrightarrow t[cvt@m,t_c,t_{mr},t_{ms},t_{\perp 2} \rightarrow t_{\perp 1}]$

The following propositions summarize above discussions:

Proposition 5.4 (*Scalability of COIN*, parameterization) When the component conversions for a modifier are parameterizable, only one parameterized component conversion needs to be specified for the modifier. If the component conversions of all M modifiers are parameterizable, the COIN approach only needs *M* parameterized component conversions.

Proposition 5.5 (*Scalability of COIN*, inversion) When all component conversions are invertible, the COIN approach needs $\sum_{i=1}^{M}(n_i - 1)$ component conversions.

To put the results in perspective, let us consider a scenario of 200 agencies in a coalition of countries sharing counterterrorism data. Suppose there are 50 types of data such as subject height, arrival airport, meeting location, etc. The data is not standardized across the agencies. Assume that each type of the data has different representations and semantic assumptions that can be represented by a modifier with 3 unique values in the COIN approach. Each agency may obtain data from all the other agencies to perform counterterrorism analysis. If we consider each agency as a source, this is a problem of enabling semantic interoperability amongst 200 sources. According to the scalability results:

- *BF* approach requires 39,800 comprehensive conversions
- *IS* approach requires 400 comprehensive conversions
- COIN approach requires 300 component conversions in the worst case, or 100 invertible component conversions, or 10 parameterized component conversions

Clearly, the COIN approach has much greater scalability than *BF* and *IS* approaches.

5.3.3 Implication of Temporal Extensions

Recall that when a source has time-varying semantics, similar data at different time periods may be interpreted differently. For example, in the Frankfurt stock price source, stock prices on or before "December 31, 1998" are in DEM, and those on or after "January 1, 1999" are in EUR.

Time-varying semantics may be handled in several ways. With the *BF* or *IS* approach, one possibility is to write code in the comprehensive conversions to determine appropriate reconciliations corresponding to different time periods. Although the flexibility and scalability results are the same as previously discussed, the comprehensive conversions become much more convoluted with the extra code that deals time-varying semantics, making the conversions more difficult to maintain.

Alternatively, a source with time-varying semantics can be partitioned *a priori* into multiple sources, each having static semantics. For example, the Frankfurt stock prices source can be partitioned into two (virtual) sources, with one having data on and before "December 31, 1998", and the other having data on or after "January 1, 1999". While this alternative eliminates the code to determine different reconciliations for different time periods, it increases the number of (virtual) sources, therefore, adversely impacts the scalability of *BS* and *IS* approaches.

The temporal extensions to the existing COIN approach overcome these problems. Time-varying semantics in a source is described declaratively using modifiers that can have different values in different time periods. Using these descriptions, the extended COIN reasoning mechanism can determine the time periods during which there is no time-varying semantics, and for each such period, a set of component conversions can be used to reconcile semantic difference. As a result, the component conversions can be provided the same way as in the cases where no source has time-varying semantics. In addition, the reasoning mechanism's capability of determining time periods of static semantics eliminates the need for *a priori* partitioning of data sources. Thus, the overall effect of the temporal extensions is that the flexibility and scalability results for COIN, as presented in the previous section, remain unchanged when sources have time-varying semantics.

"The exciting thing is serendipitous reuse of data: one person puts data up there for one thing, and another person uses it another way."
– Tim Berners-Lee

Chapter 6

Public Policy for Database Protection and Data Reuse

There is an ever increasing amount of electronically accessible data, especially on the Internet and the Web. To a certain extent, the Web has become the largest data repository consisting of sources with semantic heterogeneity. The accessibility of the Web and a variety of technologies, including COIN's semantic interoperability technology, allow someone to easily create new databases by systematically extracting and combining contents of other sources. In fact, we demonstrated in Zhu et al. (2002b) that using the Cameleon (Firat et al., 2000) data extraction technology and the COIN mediation technology, we can create a database to provide a global price comparison service. The extraction technology allows us to extract and reuse price data from other web sources; the COIN mediation technology subsequently reconciles semantic difference amongst disparate sources and diverse users.

While many technology-enabled data reuse activities create value for society, these activities may be against the interests (e.g., financial interests) of the source owners whose data has been reused. This conflict has infused debate about providing legal protection to non-copyrightable database contents[43] and regulating data reuse activities.

In formulating public policy on this issue, one should consider various stakeholders and different factors related to the value of data and the value created from data reuse. There can be many stakeholders, among which database creators, data reusers, and the consumers of the creator and/or reuser database products are the primary ones. One of the important factors to consider in policy formulation is the financial interests in database contents. For example, a creator who invested in creating a database is interested in recouping the investment using the revenues the database helps to generate. The revenues can be reduced when a reuser creates a competing database by extracting the contents from the creator's database. Thus creators would like to have certain means of protecting

[43] A database can contain copyrightable contents, e.g., a database containing MP3 songs. In this cause, the reuse of the contents is regulated by copyright law. Copyright laws in different jurisdictions may differ in the minimal requirements for database contents to copyright protection. In the U.S., data records about certain facts, e.g., phone number listings in white pages, are not copyrightable.

the contents in their databases. Without adequate protection, the incentives of creating database could diminish. There may be other reasons for having restrictions on data reuse. For example, a database creator may want to restrict reuses as a means of ensuring data quality because certain reuses can potentially introduce inaccuracies in data. Not all reuses are for financial purposes only, in which case, a reuser may view restrictions on reusing publicly accessible data as a violation of "freedom of speech" right, an essential element of human rights protected by international law. Besides, privacy concerns often arise when the data contains personal information. Furthermore, people from different cultures and jurisdictions often hold different views, and thus attach different values (not necessarily financial values), to these various factors.

While all factors involved are worthwhile for study, it is beyond the scope of this chapter to provide a comprehensive analysis on all of them. Rather, we focus on the financial interests in non-copyrightable database contents, and analyze the case where the database is publicly accessible and no enforceable contract exists to restrict data reuse. We mainly address the issue of finding a reasonable balance between incentive protection and value creation through data reuse, i.e., determining appropriate protection to database contents so that the creators still have sufficient incentives to create databases, and at the same time, value-added data reuse activities are accommodated. We achieve this objective by developing an economic model, using the model to identify various conditions, and determining policy choices under these various conditions.

6.1 Introduction – Legal Challenges to Data Reuse

As mentioned earlier, technologies such as web data extraction and context mediation have made it much easier to create new databases by reusing contents from other existing databases. New business practices consequently emerged to take advantage of these capabilities. For example, *Bidder's Edge* created a large online auction database by gathering bidding data of over five million items being auctioned on more than 100 online auction sites, including the largest online auction site *eBay*. Similarly, *mySimon* built an online comparison shopping database by extracting data from online vendors. *Priceman* provided an improved comparison shopping service by aggregating data from over a dozen comparison databases including mySimon. There are also account aggregators that gather data from multiple online accounts on behalf of a user and perform useful analyses, e.g., *MaxMiles* allows one to mange various rewards program accounts and *Yodlee* aggregates both financial and rewards program accounts. Common to these aggregated databases is that they add value by providing ease of use of existing data, either publicly available or accessible on behalf of users (e.g., through the use of their user IDs and passwords). Various types of data reuse and the business strategies for data reuse can be found in Madnick and Siegel (2002).

Unfortunately, these value added data reusers have faced serious legal challenges for the data they extracted. For example, EBay won a preliminary injunction against Bidder's Edge and the two firms later settled the case. mySimon sued Priceman and the latter ceased to operate for fear of legal consequences. There have been a few other cases[44]. The legal principles commonly used in the plaintiff claims include copyright infringement, trespass to chattels, misappropriation, violation of federal Computer Fraud and Abuse Act, false advertisement, and breach of contract[45]. Since none of the cases reached a definite conclusion, it is still a question whether it is legal to reuse publicly available or otherwise accessible factual data in value creating activities. Although the issue of reusing facts existed long before the Web became pervasive, the difficulty in applying the laws that predate the Web and the ease of data reuse in recent years has given the lawmakers a certain sense of urgency to resolve the issue by creating a new database protection law.

One of the purposes of such a law is to preserve the incentives of creating databases by providing legal protection to the investment in databases. This will inevitably run afoul of the societal interests in advancing knowledge by allowing reuse of facts in databases (Samuelson, 1996). To resolve this conflict, the new law has to strike the right balance between preserving the incentives of database creation and ensuring adequate access for value creating data reuse.

Debate in the past and discussions in existing literature (Samuelson, 1996; Richman and Samuelson, 1997; Sanks, 1998; Maurer, and Scotchmer, 1999; Reichman and Uhlir, 1999; O'Rourke, 2000; Lipton, 2003) have identified this major issue but fall short in finding this delicate balance. In this chapter, we develop an economic model to identify various conditions for setting a reasonable balance. Before delving into the model, we first briefly describe the landscape of legal protection for databases. After a formal presentation of the model, we relate it with legal proposals and discuss several useful insights developed from our analytic results.

6.2 Legal Protection for Database Contents

6.2.1 A Brief History of Database Legislation

Non-applicability of Copyright Law. The impetus for database protection started in 1991 after the Supreme Court in the U.S. decided the *Feist v. Rural*[46] case. In compiling its phone book covering the service area of *Rural Telephone Co.*, *Feist Publications* copied about 8,000 records of Rural's White Pages. In the appeal case, the Supreme Court decided that Feist did not infringe Rural's copyright in that white pages lack the minimal originality to warrant copyright protection. It is the original selection and arrangement of data, not the investment in creating the database or the contents in the database, that is protected by copyright in the U.S. Thus, under current case law, copyright law has not been found to restrict the reuse of the contents in a database concerned in this Thesis.

New Database Legislation. While the database creators in the U.S. were pushing for new database legislation, the European Union (EU) introduced the Database Directive[47] in 1996 to

[44] E.g., *HomeStore.com v. Bargain Network* (S.D. Cal, 2002), *TicketMaster v. Tickets.com* (C.D. Cal., 2000), *First Union v. Secure Commerce Services, In.* (W.D. N.C, 1999), etc. Numerous cases in Europe can be found at http://www.ivir.nl/files/database/index.html and in Hugenholtz (2001).

[45] A legal analysis of these claims can be found in court documents, e.g., the eBay case in 100 F. Supp. 2d 1058. ND Cal., May 24, 2000.

[46] 499 US 340, 1991.

[47] "Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases", a copy of the Directive can be found at http://europa.eu.int/ISPO/infosoc/legreg/docs/969ec.html.

provide legal protection for database contents. Under its reciprocity provision, databases from countries that do not offer similar protection to databases created by EU nationals are not protected by the Directive within the EU. This created a situation where U.S. database creators felt they were neither adequately protected at home, nor abroad. In response, the database industry pushed the Congress to create a new law to provide similar protection to database contents. Since then, the U.S. has attempted six proposals, all of which already failed to pass into law. The recent two bills are HR 3261 and HR 3872. Figure 6.1 briefly summarizes these legislative proposals.

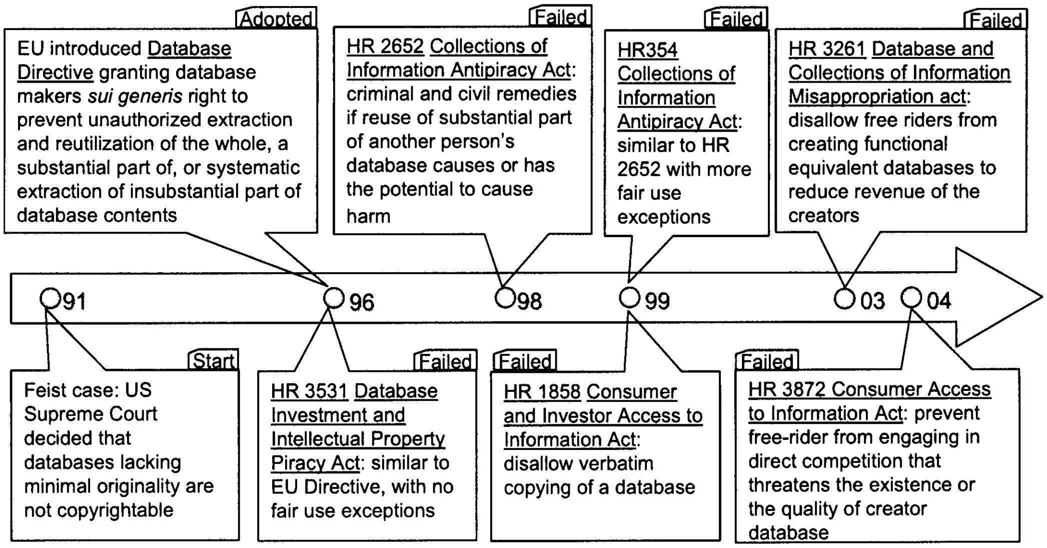

Adopted
EU introduced Database Directive granting database makers sui generis right to prevent unauthorized extraction and reutilization of the whole, a substantial part of, or systematic extraction of insubstantial part of database contents
Failed
HR 2652 Collections of Information Antipiracy Act: criminal and civil remedies if reuse of substantial part of another person's database causes or has the potential to cause harm
Failed
HR354 Collections of Information Antipiracy Act: similar to HR 2652 with more fair use exceptions
Failed
HR 3261 Database and Collections of Information Misappropriation act: disallow free riders from creating functional equivalent databases to reduce revenue of the creators
91
96
98
99
03
04
Start
Feist case: US Supreme Court decided that databases lacking minimal originality are not copyrightable
Failed
HR 3531 Database Investment and Intellectual Property Piracy Act: similar to EU Directive, with no fair use exceptions
Failed
HR 1858 Consumer and Investor Access to Information Act: disallow verbatim copying of a database
Failed
HR 3872 Consumer Access to Information Act: prevent free-rider from engaging in direct competition that threatens the existence or the quality of creator database

Figure 6.1 History of Database Protection Legislation.

The *sui generis*[48] right approach taken by the EU creates a new type of right in database contents; unauthorized extraction and reutilization of the data is an infringement of this right. Lawful users are restricted not to "perform acts which conflict with normal exploitation of the database or unreasonably prejudice the legitimate interests of the maker of the database". Here "the legitimate interests" can be broadly interpreted and may not be limited to commercial interests.

HR 3531 of 1996 closely followed this approach with even more stringent restrictions on data reuse. Although the Database Directive has been adopted by the EU, HR 3531 failed to pass in the U.S. One of the main concerns is the constitutionality of the scope and strength of the kind of protection in the EU Database Directive (Reichman and Samuelson, 1997; Colsten, 2001). Other issues in the EU Database Directive include the ambiguity about the minimal level of investment required to qualify for protection (Ruse, 2001; Hugenholtz, 2003), its lack of compulsory license provisions (Colsten, 2001), the potential of providing perpetual protection under its provision of automatic right renewal after substantial database update, the ambiguity in what constitutes a "substantial" update, and several other issues which we discuss in the next subsection.

All subsequent U.S. proposals took a *misappropriation* approach where the commercial value of databases is explicitly considered. HR 2562 of 1998 and its successor HR 354 of 1999 penalize the commercial reutilization of a substantial part of a database if the reutilization causes harm in the

[48] In Latin, meaning "of its own kind", "unique".

primary or any intended market of the database creator. The protection afforded by these proposals can be expansive when "intended market" is interpreted broadly by the creator. At the other end of the spectrum, HR 1858 of 1999 only prevents someone from duplicating a database and selling the duplicate in competition.

The proposal HR 3261 of 2003 has provisions that lie in between the extremes of previous proposals. It makes a data reuser liable for "making available in commerce" a substantial part of another person's database if "(1) the database was generated, gathered, or maintained through a substantial expenditure of financial resources or time; (2) the unauthorized making available in commerce occurs in a time sensitive manner and inflicts injury on the database or a product or service offering access to multiple databases; and (3) the ability of other parties to free ride on the efforts of the plaintiff would so reduce the incentive to produce the product or service that its existence or quality would be substantially threatened". The term ''inflicts an injury'' means "serving as a functional equivalent in the same market as the database in a manner that causes the displacement, or the disruption of the sources, of sales, licenses, advertising, or other revenue" (emphasis added by author).

The purpose of HR 3872 is to prevent misappropriation while ensuring adequate access to factual information. It disallows only the free-riding that endangers the existence or the quality of the creator database. Unlike in HR 3261, injury in the form of decreased revenue alone is not an offence. Another difference from HR 3261 is that it suggests the Federal Trade Commission be the enforcing authority.

These legislative initiatives demonstrate the substantial difficulties in formulating a database protection and data reuse policy that strikes the right balance, which is a prevailing issue in dealing with other kinds of intellectual property (Besen and Raskind, 1991). An economic understanding of the problem can help address other issues discussed in the next section.

6.2.2 Other Major Issues

Extensive legal discussions have raised many concerns about a new database law. These include but are not limited to the issues discussed below.

Data monopoly. There are situations where data can only come from a sole source due to economy of scale in database creation or impossibility of duplicating the event that generates the data set. For example, no one else but eBay itself can generate the bidding data of items auctioned on eBay. A law that prevents others from using the factual data from a sole source in effect legalizes a data monopoly. Downstream value creating reutilizations of the data will be endangered by a legal monopoly.

Cost distortion. Both the EU database directive and the latest U.S. proposals require substantial expenditure in creating the database for it to be qualified for protection. Database creators thus may over invest at an inefficient level to qualify (Samuelson, 1996).

Update distortion and eternal protection. This is an issue in EU law, which allows for automatic renewal of *sui generis* right once the database is substantially updated. Such a provision can induce socially inefficient updates and make possible eternal right through frequent updates (Koboldt, 1997).

Constitutionality. Although the Congress in the U.S. is empowered by the Constitution to regulate interstate commerce under the Commerce Clause[49] and the misappropriation approach often gives a database law a commercial guise, the restrictions of the Intellectual Property Clause[50] often apply to any grant of exclusive rights in intangibles that diminishes access to public domain and imposes significant costs on consumers (Heald, 2001). Most database contents are facts in the public domain; disallowing mere extraction for value creating activities runs afoul of the very purpose of the Intellectual Property Clause that is to "promote the progress of science and useful arts". Since little extra value for the society as whole is being created by simply duplicating a database in its entirety, preventing verbatim copying of a database is clearly constitutional. Extracting all contents of a database is very much like duplicating the database. Unlike in copyright law where there is a reasonably clear idea-expression dichotomy (i.e., copyright protects the expression, not the idea conveyed by the expression), extraction-duplication in data reuse is much like a continuum, not a dichotomy (Heald, 2001). Thus a constitutional database law needs to determine up to how much one is allowed to extract database contents.

International harmonization. Given the global reach of the Web and increasing international trade, it is desirable to have a harmonized data reuse policy across jurisdictions worldwide. The EU and the U.S. are diverging in their approaches to formulating data reuse policies. A World Intellectual Property Organization (WIPO) study (Tabuchi, 2002) also reveals different opinions from other countries and regions. Enforcement will be a big problem without international harmonization.

We believe the solution to these challenges hinges upon our capability of finding a reasonable balance between protection of incentives and promotion of value creation through data reuse. With this balance, value creation through data reuse is maximally allowed to the extent that the creators still have enough incentives to create the databases. Consensus can develop for international harmonization if we can determine the policy choices that maximize social welfare; a database policy so formulated should survive the scrutiny of constitutionality; other inefficiencies can be avoided or at least better understood. We will take on this challenge in the rest of the chapter by developing an economic model for database protection and data reuse policy.

6.2.3 Policy Instruments

When database creation requires substantial expenditure and competition from free riding reusers reduces the creator's revenue to a level that does not offset the cost, the creator would have no incentives to create the database and the market fails. Policy should intervene to restore the database market (assuming the database was worth creating). On the other hand, data reuse is often value creating; from a social welfare point of view, it is not necessary to intervene if the creator can remain profitable even though its revenue may decline because of competition. It is conceivable that there exist different conditions under which policy choices differ.

[49] Constitution 1.8.3, "To regulate Commerce with foreign Nations, and among the several States, and with the Indian Tribes".

[50] Constitution 1.8.8, "To promote the Progress of Science and useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries".

The recent U.S. proposal HR 3261 contains several useful aspects, underlined in the previous section, which are often considered in policy formulation. "Substantial expenditure" corresponds to the *fixed cost* in creating the database; "functional equivalent" measures the *substitutability* of the reuser database for the creator database, which is determined by the degree of *differentiation* of the two databases; "injury" or incentive reduction can be measured by *decrease of revenue*. "Time sensitive manner" is redundant with differentiation. It is common that information goods can be differentiated via temporal versioning (Shapiro and Varian, 1998). For example, real time stock quotes and 20-minute delayed stock quotes are two differentiated economic goods.

Policy instruments in most proposals on database protection are simply (1) the grant of legal protection when all criteria are met, and (2) the specification of penalties to violators. They focus on specifying what types of reuse constitute a violation and completely ignore the concern of the creator becoming a legal monopoly (except for HR 1858). Provisions on what the creator is supposed to do should not be ignored, e.g., under certain conditions the creator should be asked to license its data under reasonable terms. Such provisions are often found in other intellectual property laws, e.g., compulsory license provisions in patent laws in various jurisdictions. Thus, appropriate policy instruments should be a specification of conditions and the corresponding socially beneficial actions of the reuser as well as the creator.

6.3 Literature Review

There have been extensive legal studies on database protection policy[51] since 1996. Recently, Lipton (2003) suggests a database registration system similar to that for trademark to allow database creators to claim the markets within which their databases are protected from free riding. But social welfare analysis is not performed in this study to take account of the cost of maintaining such a system. After reviewing a number of data reuse cases in the EU and the U.S., Ruse (2001) suggests reusers negotiate licenses from database creators and conform to the licensing terms. The paper also criticizes the ambiguity in the Database Directive and recommends that the EU should consider the U.S. proposals that contain more broadly defined fair uses and provisions dealing with sole source databases. Colston (2001) provides a comparison of EU and U.S. approaches and suggests that the EU should reconsider the compulsory license provision that was in the early draft of the Database Directive, but removed from the final version. Hugenholtz (2003) introduces an emerging spin-off theory for databases that are created as a by-product of other business activities, in which case the cost of the business process should not be counted as cost of creating the database.

There has been little economics and information systems research that directly addresses the issues of database protection policy. We are aware of only one paper by Koboldt (1997), who studies various distortions of database update for *sui generis* right renewal under the EU Database Directive. From the social welfare point of view, the provision can induce inadequate update or excessive update of the database. He points out that the problem comes from the substantial change requirement for an update to renew the *sui generis* right. He shows that setting up an upper limit for

[51] See http://www.umuc.edu/distance/odell/cip/links_database.html for references to published legal reviews.

updating cost can eliminate the distortion of excessive update; no suggestion is made for eliminating the distortion of inadequate update.

Several possible economic theories can be applied in analyzing the issues. Cumulative innovation theory (Scotchmer, 1991) from the patent literature has been used to informally explain the importance of ensuring adequate access to data for knowledge and value creation (Maurer, 2001). The notion of the tragedy of anticommons (Heller, 1998) is also useful because information aggregators rely on the access to multiple information sources. As is shown in Buchanan and Yoon (2000), when there exist multiple rights to exclude, a valuable resource will be underutilized due to increased prices. Databases that hold factual information as a whole can be viewed as the "commons", thus, providing more than necessary protection to databases is analogous to anticommons and will lead to underutilization of protected databases.

6.4 A Model of Differentiated Data Reuse

As the legal discussions suggest, the reuser is sometimes a competitor of the creator in the database market[52]. Arguably, the intensity of competition depends on how differentiated the reuser database is from the creator database. The differentiation can be either horizontal or vertical[53] or both. Most aggregator databases are horizontally differentiated from the databases being extracted because they often have different features, over which the consumers have heterogeneous preferences. For example, while certain consumers value the extensive information about the auctioned items from eBay's database, other consumers value the searchability and ease of comparison at Bidder's Edge. Therefore the two databases are horizontally differentiated in product characteristics space.

As to the Priceman and mySimon databases, both provide price comparison, but the Priceman database has a wider coverage of online vendors, which may suggest a vertical differentiation between the two databases. But the reality can be more complicated, e.g., while mySimon is less comprehensive, it may be more reliable and responsive than PriceMan. Consumers often have different preferences over this set of feature. Thus the two databases are largely horizontally differentiated.

There may be cases where a reuser creates a database that is either superior or inferior in every feature relative to the creator database (to target a different market). However, we are interested in cases where the creator database is better in some features, whereas the reuser database is better in the other features. In such cases, the creator and reuser database are horizontally differentiated, with competing products located at different locations in the characteristics space. We will base our analysis on an extended spatial competition model, which was introduced by Hotelling (1929) and

[52] There are other reasons a creator does not want his data to be reused. For example, an online store may be afraid that a comparison aggregator can potentially have the effect of increasing price competition and lowering profit on sales of products. Our model focuses on "information goods" only, thus it does not capture such effect.

[53] Product characteristics are horizontally differentiated when optimal choice at equal prices depends on consumer tastes, e.g., different consumer tastes in color. Product characteristics are vertically differentiated when at same prices all consumers agree on the preference ordering of different mixes of these characteristics, e.g., at equal price, all prefer high quality to low quality. See Tirole (1988) for detail.

has been widely used in competitive product selection and marketing research (Salop, 1979; Schmalensee and Thisse, 1988).

6.4.1 Model Setup

We consider a duopoly case where there are two suppliers of database: (1) a database creator who creates one database product, and (2) a data reuser who produces a different database by reusing a portion of the contents from the creator's database. Both databases are for sale in the market. For example, the database creator could be a marketing firm who compiles a database of New England business directory that includes all business categories. A firm specializing in colleges in Greater Boston area may compile an entertainment guide by reusing a portion of the business directory. The two databases are different in terms of scope, organization, and purpose. In other words, they are differentiated in the product characteristics space. We can understand the databases in *eBay v. Bidder's Edge* case as well as in other data reuse cases similarly. Although most creator and reuser databases are free to individual consumers to view and search, a consumer still pays a price in an economic sense, e.g., time spent and certain private information revealed (e.g., search habit).

In a spatial competition model, we index database features onto a straight line of unit length, with the creator's database at point 0 and the reuser's database at point 1; the prices they set for their databases are p_0 and p_1, respectively. Consumers have heterogeneous preferences over the database features. For simplicity, we assume a unit mass of consumers uniformly distributed along the line $[0,1]$. We assume each database is worth a value v to a consumer with exact preference match. A customer at $x \in [0,1]$ consumes either none or exactly one database. When he does consume a database, he enjoys value v, pays a price, and also incurs a preference mismatch cost determined by the distance and a penalty rate t. This is summarized by the following utility function:

$$u_x = \begin{cases} 0, & \text{if buys none;} \\ v - p_0 - tx = u_{x,0}, & \text{if buys from firm 0;} \\ v - p_1 - t(1-x) = u_{x,1}, & \text{if buys from firm 1.} \end{cases}$$

We further assume that both the creator and the reuser have the same marginal cost, which is normalized to 0. The creator's investment in creating the database is modeled as a fixed cost F. The reuser incurs a fixed cost f, where $F >> f$, so we normalize f to 0. This assumption reflects the fact that the innovative reuser possesses complimentary skills to efficiently create the second database that the creator cannot preemptively develop. Firms simultaneously choose prices to maximize their profits; consumers make purchasing decisions that maximize utility u_x.

This setup reflects the uniqueness of the database and data reuse market. Many databases from which reusers extract contents are byproducts of business processes. The eBay database is the byproduct of its online auction business. Data in various accounts are generated by transactions of business activities. The cost in creating and maintaining these databases is not a decision variable to be optimized by calculating expected returns on the databases *per se*. MySimon itself is a reuser of vendor data; it is also a database creator when its database contents were extracted by Priceman. The reuse by Priceman is rather serendipitous in that mySimon made its investment decision without ever imagining its data could have been reused by another reuser. Thus, for the purpose of data reuse analysis, the cost of creating the original dataset is a sunk fixed cost instead of an investment in the sense in Research and Development literature. Similarly, the database features are often designed without ever thinking of various possible reusers. Therefore, the database locations in the feature space are not decision variables, either.

In this model, parameter t measures the degree of differentiation of the two databases with respect to consumer preferences; differentiation increases with t. When t is large, the two products are highly differentiated and the two firms can be two local monopolies. When t is small, the two products are close substitutes and fierce competition can lower profits to a level where the creator cannot recover its fixed cost. Our further analysis will be based on this intuition. For the purpose of analyzing if the creator is willing to allow reuse of its data, we also analyze the monopoly case where the creator is the only firm in the market.

In the rest of the chapter, unless otherwise noted, profit and social welfare are gross without counting the fixed cost or transaction cost. Utilitarian social welfare is used, which is the sum of firm profit and consumer surplus.

Lemma 6.1 In the duopoly case, the market is covered if $t \le v$, and is not fully covered otherwise. In the monopoly case, the market is covered by creator's database if $t \le v/2$, not fully covered otherwise. Best price, maximum profit, and social welfare vary with the differentiation parameter t in both cases as summarized in Table 6.1. ■

Table 6.1. Price, profit, and social welfare at different differentiation levels

	T	**Best price**	**Maximum profit**	**Social welfare**
Duopoly	*t≤2v/3*	$p_0^* = p_1^* = t$	$\pi_0^* = \pi_1^* = \pi^d = \frac{t}{2}$	$SW^d = v - \frac{t}{4}$
	2v/3<t≤v	$p_0^* = p_1^* = v - \frac{t}{2}$	$\pi_0^* = \pi_1^* = \pi^d = \frac{v}{2} - \frac{t}{4}$	$SW^d = v - \frac{t}{4}$
	v<t	$p_0^* = p_1^* = \frac{v}{2}$	$\pi_0^* = \pi_1^* = \pi^d = \frac{v^2}{4t}$	$SW^d = \frac{3v^2}{4t}$
Monopoly	*t≤v/2*	$p^m = v - t$	$\pi^m = v - t$	$SW^m = v - \frac{t}{2}$
	v/2<t	$p^m = \frac{v}{2}$	$\pi^m = \frac{v^2}{4t}$	$SW^m = \frac{3v^2}{8t}$

Proof. Duopoly, little differentiation (*t≤2v/3*). In the case of full market coverage, there exists a location $\tilde{x} \in [0, 1]$, such at $u_{\tilde{x},0} = u_{\tilde{x},1} \ge 0$. Then, the demand for database 0 is $\tilde{x}$ and the demand for database 1 is $(1-\tilde{x})$. Solving profit maximization for both firms with respect to p_0 and p_1, we obtain $p_0^* = p_1^* = t$ and $\pi_0^* = \pi_1^* = \pi^d = \frac{t}{2}$. Positive utility constraints at $\tilde{x}$ require *t≤2v/3*. By symmetry, the social welfare is $2\int_0^{0.5}(v-tx)dx = v - \frac{t}{4}$.

Duopoly, moderate differentiation (*2v/3<t≤v*). This is the case that requires careful examination of corner solutions. To see that $p_0^* = p_1^* = v - \frac{t}{2}$ is the equilibrium, we show that given $p_1 = v - \frac{t}{2}$, the profit maximizing price for the creator is also $v - \frac{t}{2}$, and vice versa. When $p_1 = v - \frac{t}{2}$, $u_{\frac{1}{2},1} = 0$. If the creator charges same price, then each firm takes up one half of the market and makes a gross profit of $\frac{v}{2} - \frac{t}{4}$. We only need to show that any deviation by the creator yields a lower profit. For any infinitesimal positive value $\delta \in R^+$, let us first suppose the creator wants to capture more than a

half of the market by choosing a lower price $p_0 = p_1 - \delta$. With $u_{\tilde{x},0} = u_{\tilde{x},1}$ we can find the creator's demand $\tilde{x} = {}^{(t+\delta)}\!/_{2t}$. Therefore, the creator's profit is $\pi_0 = p_0\tilde{x} = (p_1 - \delta)^{(t-\delta)}\!/_{2t}$. It is easily shown that $\frac{\partial \pi_0}{\partial \delta} = \frac{v - 3t/2}{2t} - \frac{\delta}{4t} < 0$ when ${}^{2v}\!/_3 < t$ because both terms are negative. Now let us suppose that the creator wants to deviates by charging a higher price $p_0 = p_1 + \delta$; as a result, it will cover less than a half of the market. We can derive $\frac{\partial \pi_0}{\partial \delta} = \frac{t-v}{t} - \frac{2\delta}{t} < 0$ because $t \leq v$.

Duoploy, high differentiation ($v<t$). Each firm's demand is up to the location of the marginal consumer whose utility of purchasing a database is 0. Take the creator, this marginal consumer is located at $\tilde{x} = {}^{(v-p_0)}\!/_t$. Maximizing profit yields $p_0^* = {}^v\!/_2$. Therefore, $\tilde{x} = {}^{(v-\frac{v}{2})}\!/_t = {}^v\!/_{2t} < {}^1\!/_2$, and $\pi_0^* = {}^{v^2}\!/_{4t}$. By symmetry we obtain the reuser's price and profit. Social welfare is $2\int_0^{v/2t}(v - tx)dx = {}^{3v^2}\!/_{4t}$.

Monopoly, moderate preference heterogeneity ($t \leq v/2$). Similar to moderately differentiated duopoly case, it is better for the monopoly to cover the entire market. Letting $u_{1,0} = 0$, we derive the price. Demand is 1. It is straightforward to derive social welfare.

Monopoly, high preference heterogeneity ($v/2<t$). Similar to highly differentiated duopoly case, it is better for the monopoly to cover a fraction of the market. Straightforward optimization yields the results. ■

We graph the result of Lemma 6.1 in Figure 6.2 to help make useful observations; values for both axes are the factors of the product valuation v.

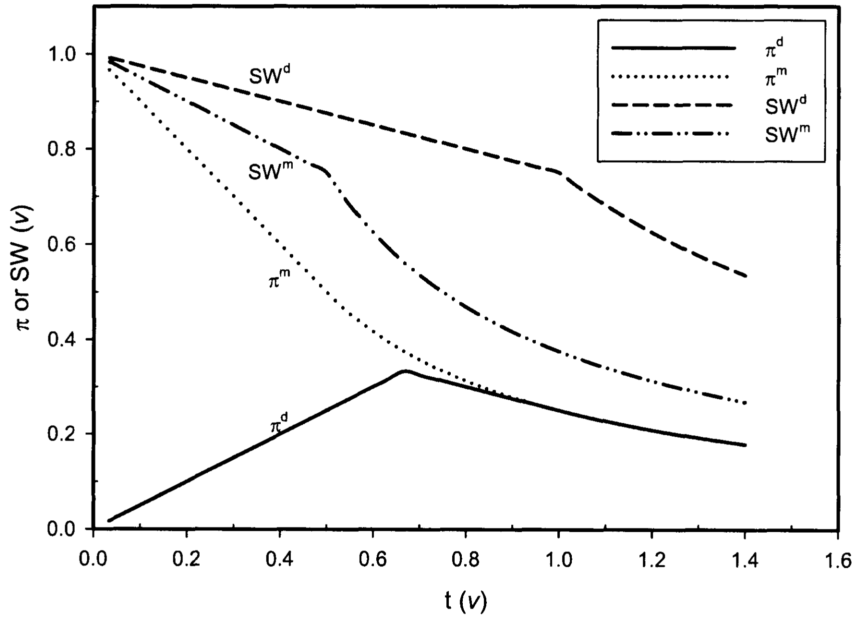


Figure 6.2. Change of profit and social welfare with differentiation factor t.

Corollary 6.1 *$\pi^m > \pi^d$ if $t < v$, and $\pi^m = \pi^d$ otherwise.* ■

Corollary 6.1 says that when the reuser's database is not sufficiently differentiated from the creator's, the creator makes less profit because of competition from the reuser's database. When the two databases are highly differentiated, the creator is not harmed by the reuser. The corollary also implies that if the creator is the sole data source and can fully control the access to the data, it will deny access if a reuser intends to free ride and make a database without sufficient differentiation.

Corollary 6.2 *$SW^d > SW^m$ for all $t>0$.* ■

Corollary 6.2 says that from the social welfare perspective, two differentiated databases are better than one database. The implicit assumption for this to hold is that the creator makes a positive net profit.

6.4.2 Necessity of Database Law

When the creator's profit is less than its fixed cost, i.e., $\pi^m < F$, the database will not be created to begin with. For market failure analysis, we focus on the case where the creator is self sustainable, i.e., $\pi^m \geq F$.

In the presence of a free riding reuser, the creator makes a duopoly profit π^d, which is smaller than monopoly profit π^m when $t \leq v$. Therefore, free riding will cause market failure if $\pi^d < F \leq \pi^m$ and $t \leq v$. That is, if F falls in the region above the π^d line and below the π^m line in Figure 6.2, free riding causes market failure, in which case an intervention is necessary. This is often the argument for having a new database law.

Without a database law, other means that database creators can use to protect their databases seem to be ineffective in some cases. For example, a creator can use certain non-price predation strategies, namely by raising rival's costs (Salop and Scheffman, 1987), to deter entry or at least to soften competition from the reuser. In the past, creators attempted cost raising strategies such as blocking the IP addresses used by reuser computers and frequently changing output format to make data extraction more difficult. This can be modeled by letting the creator choose a technology investment level T, with which the marginal cost of the reuser becomes $C_1(T)$. The cost of installing such anti-extraction technologies is often small enough to be negligible. When T is such that $0 \leq C_1(T) \leq \min\left\{\frac{3}{2}(v-t), 3t, 2v-3t\right\}$, the creator profit becomes $\pi^d_{0,T} = {(t+\frac{C_1}{3})^2}/{2t} > \pi^d = \frac{t}{2}$, i.e., the creator profit is higher than when the technology is not used. The reuser profit is $\pi^d_{1,T} = {(t+\frac{2C_1}{3})(t-\frac{C_1}{3})}/{2t}$, which could be greater or less than π^d, depending on the level of $C_1(T)$. The first two items in the constraint for $C_1(T)$ ensure that the reuser is not deterred; the third item ensures full coverage of the market. Obviously, if $C_1(T)$ is very high, the reuser will be deterred. However, anti-extraction techniques were not very effective in practice[54]; we suspect that $C_1(T)$ has been too small to have substantial effect. Therefore, we will assume no anti-extraction is in place in the rest of the analysis. Regardless of the effectiveness of anti-extraction techniques, they are socially wasteful investment because they merely help transfer consumer surplus and reuser profit to the creator. A database law

[54] eBay tried blocking the IP addresses used by Bidder's Edge, Bidder's Edge circumvented this obstacle by using a pool of IP addresses dynamically.

that grants the creator the right to license its data to reusers can reduce or eliminate this social inefficiency. When database creators are also reusers, the cost-raising problem may not arise at all[55].

There can be a need for a database law from the reuser's point of view. Database reusers often face legal challenges from database creators. For example, reusers often receive legal threat notices[56] and sometimes are sued by the creators. The uncertainty of various proposed database bills creates significant legal risks for the reusers, who are often small but innovative firms. As a result, some reusers have to exit the market, and certain value-added data reuses cannot occur. In this case, having a database law that clearly specifies the kinds of legal reuses will help to create and sustain a market of socially beneficial reuser databases.

6.4.3 Conditions and Choices of Data Reuse Policy

A socially beneficial data reuse policy can correct market failure by restricting certain free riding in data reuse; the legal certainties it provides also help eliminate or reduce wasteful cost-raising investment by incumbent database creators. This can be done either by requiring the reuser to pay the creator for the data or by disallowing data reuse all together. The creator can ask the reuser to pay a data reuse fee, r, which can be up to the reuser's profit π^d; asking a fee $r > \pi^d$ is equivalent to disallowing reuse because the reuser would make a negative profit. Negotiating the fee schedule r and administrating data reuse policy often incur some cost. To model this reality, let us suppose that when the creator asks for r, it actually gets αr, where $\alpha \in [0, 1]$ and it measures transaction efficiency. Thus, in the duopoly case with data reuse policy in place, ($\pi^d + \pi^d$) is the best the creator can get to offset its fixed cost F if it ever allows someone to reuse its data[57]. Before we develop the formal analysis, we describe the intuitions by plotting this upper bound condition along with profit curves in Figure 6.3.

[55] In the financial sector, many banks started offering account aggregation service shortly after account aggregators emerged. That is, banks as database creators, became data reusers, so they had incentives to lower data reuse cost. As a result, they initiated a standardization project to facilitate aggregation, see "FSTC to Prototype Next Generation Account Aggregation Framework" at http://www.fstc.org/press/020313.cfm. In this case, legal intervention is unnecessary.

[56] For instance, a few online travel agencies recently sent warning letters to data reusers that allow consumers to compare prices. See "Cheap-Tickets Sites Try New Tactics" by A. Johnson, Wall Street J., October 26, 2004.

[57] We assume there will be no collusive joint profit maximization. A Nash bargaining outcome will be 50/50 split of the reuser profit. For purpose of market failure correction, this outcome can be simulated by setting α to 0.5, although welfare analysis will be somewhat different.

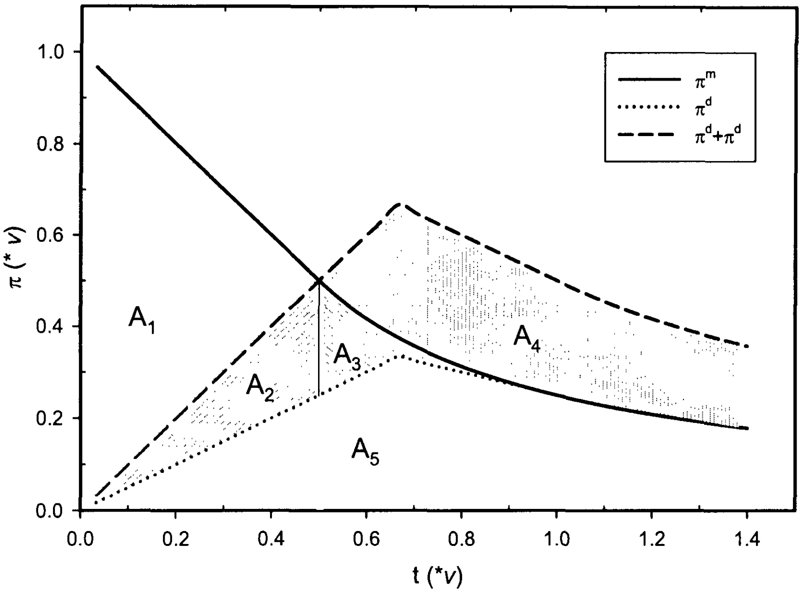


Figure 6.3 Change of Profits with Differentiation Factor *t*.

We mark five areas A_1 through A_5 in Figure 6.3. The upper-bound ($\pi^d + \pi^d$) curve will lower when α decreases, which enlarges A_1 and reduces A_2, A_3, and A_4. There are different implications when *t* (the X-axis) and the fixed cost *F* comparing to profits (the Y-axis) are such that *F* falls in one of the areas. If *F* falls in A_1 (i.e., *t* is between 0 and 0.5**v*, and *F* is below the solid line for monopoly profit and above the dashed line for sum of duopoly profits), the upper bound ($\pi^d + \pi^d$) curve is below π^m curve, meaning that even if the creator can reap all the profit made by the reuser, it still cannot cover all its fixed cost. In this case, the existence of a reuser causes an uncorrectable market failure, so it is better to let the creator be a lawful monopoly.

For $F \in A_2$, market failure can be corrected by asking the reuser to pay for the reuse of the data. But the creator prefers to be a monopoly. To maximize social welfare, the policy should insist that the creator license its data to the reuser. When $F \in A_3$, the creator can make more than it would as a monopoly, thus it is willing to license its data.

For the database to be created to begin with, we have assumed that a monopoly profit is greater than the required fixed cost ($\pi^m > F$). Area A_4 shows an interesting scenario. When $F \in A_4$, a monopolist creator cannot afford to create the database but the database and a variant of it still can be created so long as the creator and the reuser can share the fixed cost. In this case, the reuser can be considered as a database creator who incurs a fixed cost F_1, where $F_1<F$. When a third party reuses data from either or both jointly developed databases, the model can be used to analyze various conditions between this third party (i.e. the reuser) and the creator(s).

Finally, when $F \in A_5$, the cost of creating the database is low and free riding would not cause market failure. It actually enhances social welfare when $\alpha < 1$ because transferring *r* to the creator costs the society $(1-\alpha)r$.

Next, we will formalize the above intuitive explanations. To simplify analysis, we let $r = \pi^d$, i.e., we assume that the creator has the negotiation skills or legal power to ask the reuser to disgorge all profits from reusing the data. For notational simplicity, we let $\pi^{dl} = (1+\alpha)\pi^d$ and $SW^{dl} = SW^d - (1-\alpha)\pi^d$, which respectively denote the gross profit of creator and gross social welfare when the creator licenses its database to the reuser.

Theorem 6.0 (Minimal transaction efficiency) There exists a minimal transaction efficiency $\hat{\alpha}$, below which having a monopoly is welfare enhancing compared to having a duopoly with a fee paying reuser. $\hat{\alpha} = 0.5$ when $t \le v/2$; $\hat{\alpha} = \frac{3v^2+6t^2-8vt}{4t^2}$ when $v/2 < t \le 2v/3$; and $\hat{\alpha} = \max\{0, \frac{3v^2-4vt}{4vt-2t^2}\}$ when $t > 2v/3$. ■

Proof. With a fee paying reuser, the social welfare is $SW^{dl} = SW^d - (1-\alpha)\pi^d$. Licensing is socially beneficial only if $SW^{dl} \ge SW^m$. Using the results in Lemma 6.1, we can solve the inequality and obtain $\hat{\alpha}$. ■

This is a refinement to Corollary 6.2. When free riding causes market failure, data reuse policy must choose between asking the reuser to pay and disallowing data reuse all together. High transaction costs may out weigh the welfare gain from having a reuser database. When transaction efficiency is below this threshold, it is better that the creator not license data to the reuser; conversely, when transaction efficiency is above this threshold, the creator should license its database to the reuser, subject to the constraint that the creator can make a positive profit with licensing fee from the reuser.

Theorem 6.1 (A_1: Little differentiation, high cost) When $t \le v/2$ and $(1+\alpha)t/2 < F \le v - t = \pi^m$, legal protection to the creator's database should be granted. The existence of a reuser database causes a market failure even if the reuser pays a licensing fee; it is socially beneficial to let the creator be a monopoly in the market by disallowing the creation of the reuser database. ■

Proof. In the presence of a reuser database, the creator's profit is *t/2* (see Lemma 6.1) if the reuser is a free rider, or $(1+\alpha)t/2$ if the reuser pays a fee equal to its profit. In both cases, the creator cannot make a positive net profit, thus the database will not be created and social welfare is 0. Without the reuser database, the creator earns a monopoly profit $\pi^m = v - t$, which has been assumed to be greater than or equal to *F*; net social welfare is $SW^m - F = v - t/2 - F \ge t/2 > 0$. ■

Theorem 6.2 (A_2: Little differentiation, moderate cost) When $t \le v/2$ and $t/2 \le F < (1+\alpha)t/2$, legal protection to the creator's database should be granted. The creator is not willing to license its database to the reuser, but it is socially beneficial to require a compulsory license so long as $\alpha > \hat{\alpha} = 0.5$. If $\alpha \le 0.5$, it is better to let the creator be a monopoly. ■

Proof. This can be easily proved with Lemma 6.1 and Theorem 6.0. ■

Theorem 6.3 (A$_3$: Moderate differentiation, moderate cost) When $v/2 < t \le v$ and $\pi^d < F \le \min\{(1+\alpha)\pi^d, v^2/4t = \pi^m\}$, legal protection of the creator's database should be granted. The creator is willing to license its database if $\alpha \ge (\pi^m - \pi^d)/\pi^d = \tilde{\alpha}$. Within the range of differentiation, $\tilde{\alpha}$ can be less than or greater than $\hat{\alpha}$. If $\hat{\alpha} < \alpha < \tilde{\alpha}$, compulsory licensing is necessary; if $\tilde{\alpha} < \alpha < \hat{\alpha}$, licensing should be disallowed even though the creator prefers.

Proof. Legal protection is necessary because with free riding the creator cannot make enough profit to cover its fixed cost. When $\alpha \ge \tilde{\alpha}$, $\pi^d + \alpha\pi^d \ge \pi^m$, i.e., the creator is better off licensing its database to the reuser. When $\hat{\alpha} < \alpha < \tilde{\alpha}$, it is socially beneficial to license but the creator makes less than monopoly profit, therefore, compulsory licensing is required. When $\tilde{\alpha} < \alpha < \hat{\alpha}$ the creator prefers to license its database but it is socially wasteful, thus licensing should be disallowed. The values of $\hat{\alpha}$ and $\tilde{\alpha}$ are presented graphically in Figure 6.4. ■

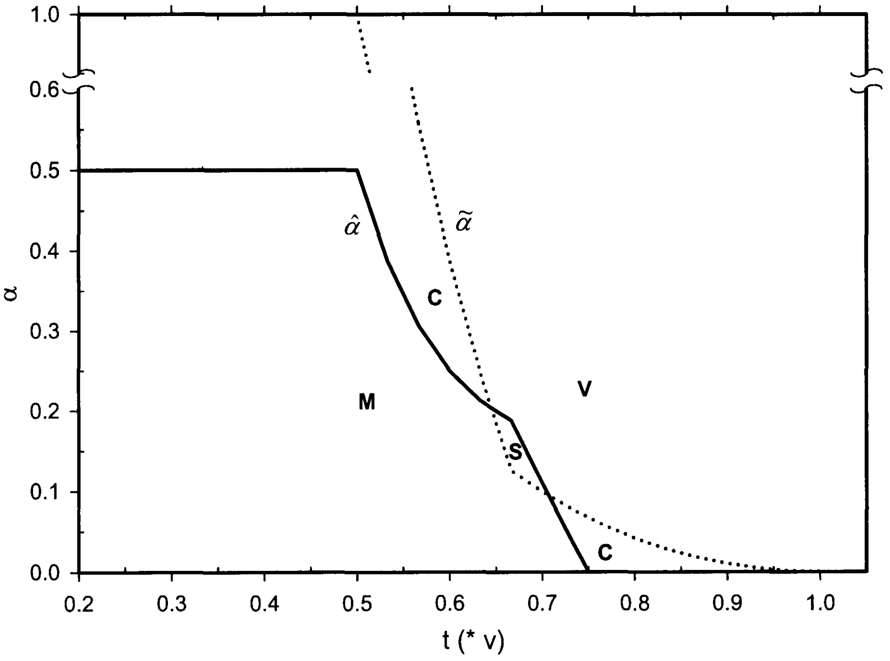


Figure 6.4 Change of minimal transaction efficiency with differentiation factor *t*.

In Figure 6.4, we see in most cases $\tilde{\alpha} > \hat{\alpha}$, meaning that generally transaction efficiency requirement is higher for voluntary licensing than for compulsory licensing. We label each region with a symbol denoting the socially beneficial policy choice, i.e., C indicates the necessity of compulsory licensing, V means voluntary licensing by the creator, M represents the case where a monopoly is better, and S is a special case where the creator wants to license but it is socially wasteful to license and the creator should be a monopoly.

Theorem 6.4 (A$_4$: Moderate to high differentiation, high cost) When $v/2 < t$ and $v^2/4t = \pi^m < F \le (1+\alpha)\pi^d$, the creator will not create the database, not because of the threat of free riding, but because of the high cost. The databases can only be jointly developed by the creator and the reuser with cost sharing agreement. ■

Proof. It is straightforward that the creator will not create the database because the development cost is greater than monopoly profit. If the cost is below joint profit discounted with transaction cost, the creator is willing to participate in joint development to make a positive profit. ■

Theorem 6.5 (A_5: Trivial databases) It is socially beneficial NOT to grant legal protection to databases when $F \le \pi^d \le v/3$ and $\alpha < 1$. ■

Proof. With low cost of creating the database, there is no market failure without legal protection. For $\alpha < 1$, $SW^d > SW^{dl} = SW^d - (1-\alpha)\pi^d$. Recall from corollary 6.2 that $SW^d > SW^m$. Social welfare without protection is SW^d, it is SW^{dl} or SW^m with protection. Therefore, no protection is socially beneficial. For all t, it is straightforward to show $\max(\pi^d) = v/3$ using Lemma 6.1. ■

Corollary 6.3 (A5: Highly differentiated databases) When $t > v$ and $\alpha < 1$, it is socially beneficial NOT to grant legal protection to databases. ■

Proof. This is a special case of Theorem 6.5. When $t > v$, $\pi^d = \pi^m$, thus free riding of the reuser has no impact to the creator. When $\alpha < 1$, it is socially better not to collect a license fee to avoid transaction cost. ■

Most enacted and proposed database protection bills grant legal protection only to databases that require substantial expenditure to create and maintain. Theorem 6.5 shows that such provisions are necessary for enhancing social welfare. We have been using the magnitudes of fixed cost to determine the socially beneficial policy choices. These magnitudes are not measured in absolute dollar amount, rather, they are relative to the market value of the database as explicitly shown in Theorem 6.5. Thus, we should not specify an absolute dollar amount threshold for a database to qualify for legal protection.

From Theorem 6.5, it seems desirable to set a fixed cost threshold $\hat{F}$ equal to duopoly profit, i.e., $\hat{F} = \pi^d \le v/3$. As we will see next, this can induce excessive investment when an efficient firm can create the database at a cost slightly lower than $\hat{F}$.

Theorem 6.6 (Over investment distortion) Suppose $\hat{F} = \pi^d$, when $\alpha > \hat{\alpha}$ and $t \le v$, a creator with $\underline{F} < F < \hat{F}$ has incentives to over invest to qualify for legal protection, where F is the cost when the creator produces the database efficiently. The value of $\underline{F}$ depends on α and t. When $t \le \frac{2v}{3+2\alpha}$ and $\underline{F} = \max\{^{(4+\alpha)t}/_2 - v, 0\}$, or when $\frac{v}{2} < t \le \frac{v}{\sqrt{2(1+2\alpha)}}$ and $\underline{F} = \max\{\frac{2(2+\alpha)t^2 - v^2}{4t}, 0\}$, the creator aggressively over-invest $\frac{(1+\alpha)t}{2}$ to become a monopoly; when $\frac{2v}{3+2\alpha} < t \le \frac{v}{2}$ or $\frac{v}{\sqrt{2(1+2\alpha)}} < t \le \frac{2v}{3}$, and $\underline{F} = {}^{(1-\alpha)t}/_2$, the creator only moderately over-invest $\pi^d = t/2$ to become eligible for licensing fee; when $\frac{2v}{3} < t \le v$,

$\alpha > \frac{(v-t)^2}{2vt-t^2}$, and $\underline{F} = (1-\alpha)\frac{2v-t}{4}$, the creator only over-invest $\pi^d = \frac{2v-t}{4}$ to qualify for receiving licensing fee. ■

Proof. When $F < \hat{F}$ (i.e., F is in A$_5$ area in Figure 6.3), the reuser can legally free ride, so the creator's net profit is $\pi^d - F$. The creator has incentive to over-invest to a level in A$_1$, A$_2$, or A$_3$ areas as long as it can make a higher net profit.

When $t \le \frac{v}{2}$, the creator has an incentive to over-invest to the minimal level of legal monopoly, π^{dl} , if the following conditions hold:

$$\begin{cases} \pi^m - \pi^{dl} \ge \pi^d - F & (1) \\ \pi^m - \pi^{dl} \ge \pi^{dl} - \hat{F} & (2) \end{cases}$$

where (1) is the condition under which being a lawful monopoly is better than having a free rider; (2) ensures that being a lawful monopoly is better than having a fee paying reuser. Solving (1) yields $F \ge \frac{(4+\alpha)t}{2} - v$, whose right hand side can be greater than or less than 0; therefore, we have $\underline{F} = \max\{{}^{(4+\alpha)t}\!/_2 - v, 0\}$. Solving (2) gives $t \le \frac{2v}{3+2\alpha}$. With the assumption of $\alpha > \hat{\alpha} = 0.5$, we know that $\frac{2v}{3+2\alpha} < \frac{v}{2}$. Similarly, the incentive compatible conditions for over-investing to $\hat{F}$ to only qualify for receiving licensing fee are:

$$\begin{cases} \pi^{dl} - \hat{F} \ge \pi^d - F & (3) \\ \pi^m - \pi^{dl} < \pi^{dl} - \hat{F} & (4) \end{cases}$$

Here, (3) ensures having a fee-paying reuser is better than having a free rider; (4) ensures having a fee paying reuser is better than being a monopoly. Solving (3) gives $F \ge (1-\alpha)\pi^d = \underline{F}$, where $\pi^d = \frac{t}{2}$; solving (4) yields $t > \frac{2v}{3+2\alpha}$.

When $\frac{v}{2} < t \le \frac{2v}{3}$, these constraints can be solved by plugging in appropriate profit functions.

When $t > \frac{2v}{3}$, the monopoly profit is only slightly higher than the duopoly profit; when α is not too small, $\pi^{dl} > \pi^m$, which gives $\alpha > \frac{(v-t)^2}{2vt-t^2}$. There is no incentive to become a lawful monopoly because the creator earns a bigger profit when there is a fee paying reuser. ■

Corollary 6.4 Over investment can also occur even if the creator already qualifies for protection but is subject to compulsory licensing as specified in Theorem 6.2. Specifically, the creator over invests at the level of ${}^{(1+\alpha)t}\!/_2$ when $(2+\alpha)t - v = \underline{F} < F < {}^{(1+\alpha)t}\!/_2$, $t \le {}^{v}\!/_2$, and $\alpha > \hat{\alpha} = 0.5$.

Proof. The creator over invests if $\pi^m - \pi^{dl} > \pi^{dl} - F$, which gives $F > \underline{F} = (2+\alpha)t - v$. The lower bound for α is necessary from Theorem 6.2. ■

Theorem 6.6 shows that when $F \in A_5$, the unprotected database creator wants to spend more at F', where $F' \in A_2$ or $F' \in A_3$, so that the database now becomes qualified and the creator can earn a bigger profit. The creator can more aggressively invest at F'' such that $F'' \in A_1$, in which case the creator becomes a legal monopoly. Corollary 6.4 shows that a creator with $F \in A_2$ wants to move to A_1 by spending more. These distortions benefit the creators but are socially wasteful.

6.5 Discussion

6.5.1 Summary of Findings

In this spatial competition model, we consider the creator, the reuser, and the society as a whole. Depending on the condition, the reuser can be a free-rider or a fee paying data reuser, or reuse is disallowed. As a unique feature of the model, we explicitly consider inefficiencies of policy administration and abstract it as the transaction efficiency parameter α . This is an improvement over previous policy research that often ignores this factor, which in effect assumes perfect efficiency of policy implementation and enforcement.

The model also allows us to clarify several important notions in policy design. The "substantial expenditure" requirement is not clearly defined in the EU Database Directive and the current U.S. proposals. We can see from this model that it should not be an absolute value; rather, it should be the fixed cost relative to the market value of the database product. The minimal cost for qualification also depends on the degree of differentiation of the reuser database. Another notion is the reduction of database creation incentives by the free riding of reusers. HR 3261 regards reduced revenue as an injury which in turn reduces the incentives of creating the database; any revenue reduction due to competition from the reuser is an offence. Thus the purpose of the proposal is about fairness to creators, not about social welfare maximization. In the model, the incentives of creating the database do not completely disappear as long as the creator makes a positive profit.

With this model, we are able to specify socially beneficial policy choices under various conditions determined by the magnitude of fixed cost of database creation (F), the degree of differentiation between the reuser database and the creator database (t), and the transaction efficiency (α). Roughly speaking, under the assumptions of this model, no protection should be given if the database can be created with trivial expenditure or the reuser database is highly differentiated. When legal protection is granted, it may take various forms, e.g., no reuse with the creator being a legal monopoly, reuse with compulsory license, and discouragement of voluntary licensing. Reuse should be disallowed if the reuser database is a close substitute of the creator database and the cost of creating the database is high. In other words, a legal monopoly is socially desirable in this case. In the other cases, the transaction efficiency plays an important role of determining if compulsory licensing is required, or if license is beneficial to the creator but wasteful to the society, thus voluntary licensing should be discouraged.

We also discover excessive investment distortions that come with this design of database policy. Creators with unqualified databases have incentives to over spend in database creation to become qualified; creators who are asked to license their databases may want to invest excessively to become a legal monopoly. These distortions occur only when the reuser database has little or moderate differentiation with the creator database. We are unable to find a mechanism to eliminate the distortions at this point. Thus the court is expected to scrutinize cases carefully to identify and penalize those who purposefully over spend in database creation.

In this model we assume that the fixed cost incurred by the reuser is negligible compared with that incurred by the database creator. Thus, the reuser database is a "free" product to the society and social welfare is generally higher when there are two databases. This result is similar to those from other intellectual property studies. For example, Yoon (2002) finds that depending on cost distribution no copyright protection can be socially beneficial. In the presence of demand network externalities, Takeyama (1994) finds that unauthorized reproduction of any intellectual property is Pareto improving, i.e., both consumers and the infringed producer, thus the society as a whole, benefit from unauthorized reproduction. We informally discussed the social welfare effect of investment that raises the reuser cost; similarly, the expenditure on monitoring data reuse is also wasteful. This is also true in copyright enforcement; see Chen and Png (2003) for their discussion on the adverse effect of anti-piracy investment.

6.5.2 Implementation and Implications

The model and the results provide helpful guidelines to specifying and implementing a socially beneficial database protection policy. This can be illustrated perhaps with critiques to the recent U.S. proposals, particularly HR 3261.

HR 3261 of 2003 is generally in line with the results here. It takes an appropriate approach with a focus on competition and the commercial value of databases. The scope of the proposal is confined by the term of "functional equivalent", which means that the proposal concerns reuse that produces a close substitute of the creator's database. Although it is a bit vague, it does intend to protect non-trivial databases only.

However, HR 3261 is obviously crude and lacks important compulsory licensing provisions. Our model has roughly three levels in both the degree of differentiation and the cost of database creation. This allows for fine tuning of policy choices. HR 3261 takes a more or less binary approach. It thus misses several opportunities of social welfare maximization. Without compulsory license, sole source creators will become a lawful monopoly under the proposal, which is harmful to society. These shortcomings will likely raise constitutionality concerns.

In addition, the three conditions in HR 3261 are essentially a paraphrasing of the misappropriation doctrine established in *INS v. AP*[58] of 1918 and more recently in *NBA v. Motorola*[59] of 1997. In the former appeal case, International New Service (INS) took factual stories from the Associated Press's (AP) bulletins and East Coast newspapers and wired them to its member newspapers on the West Coast. In the later case, Motorola transcribed NBA playoff scores from broadcast and sent them to its pager subscribers. Both INS and Motorola were found guilty under the misappropriation doctrine. The practical question is if it makes sense to codify a well established doctrine and make it into a statute to regulate data misappropriation only. With its substantial

[58] 248 US 215 (1918).

[59] 105 F.3d 841 (1997).

similarity to the misappropriation doctrine, HR 3261 would have little impact because without it any data reuse case can be decided using the doctrine.

With HR 3872 of 2004, injury alone is not an offense that triggers government intervention, which only comes in when the injury reaches the point where the creator would not create the database or maintain its quality. Our model clarifies this vague criterion.

HR 1858 prevents duplication of a database, which is an extreme case where t is nearly zero. With no differentiation, the reuser (now a duplicator) adds little value to the society, i.e., $SW^d \approx SW^m$, and market failure is highly likely with even a moderate creation fixed cost, thus database duplication should be disallowed. The proposal also clearly specifies a compulsory licensing requirement for sole source creators. Although HR 1858 would very likely pass constitutional scrutiny, it has certain drawbacks, e.g., its scope is deemed to be too narrow because it only covers one extreme case of data reuse.

Overall, the results from the economic model provide useful insights for formulating database protection policy. By revisiting the two undecided cases described in the introduction section, we can briefly discuss the implications of a policy suggested by the model.

eBay v. Bidder's Edge. In the eBay case, the computing resource is not the subject matter of such a policy, which concerns the data, not the resources that deliver the data. Thus trespass on the Internet is a different issue out of the scope of this discussion. According to the model, we need to at least examine the degree of differentiation of the database developed by the reuser Bidder's Edge. In terms of searching of bidding data, the reuser database has a much broader coverage; thus, there is competition from the reuser database. In terms of functionality, eBay's database allows one to buy and sell items; the reuser database does not provide any actual auction service. Thus the two databases exhibit significant differentiation. Searching alone does not, in general, reduce eBay's revenue from its auction service. In addition, searching and actual auction are two different markets. If we subscribe to the spin-off theory (Hugenholtz, 2003), the eBay database will not meet the cost criterion. Therefore, free reuse by Bidder's Edge should be allowed under our model.

mySimon v. Priceman. In mySimon case, the reuser database is a superset of the creator's. Both are in the searchable comparison shopping database market. Free riding by the reuser certainly reduces creator's revenue. If the reduction reaches a level that the creator cannot make a positive profit, which is likely in this case, then the reuser should be asked to pay a fee for using the data or it is penalized for violating of the database protection policy.

6.5.3 Concluding Remarks

We address a pressing issue in database legislation that needs to find the right balance between protecting incentives of database creation and preserving enough access to data for value creating activities. With an extended spatial competition model, we are able to identify a range of conditions and determine different policy choices under these conditions. From these results, we derive several guidelines that are useful to law makers when they consider economic factors in database policy formulation. A better understanding of the economic issues in database legislation provided by our analysis will be helpful in developing consensus towards international harmonization in database regulation.

There are also a number of limitations in our analysis. As discussed earlier, we focus on financial interests in database contents; other factors concerning societal values of data and data reuse are not considered. Our economic model considers the competition between the creator and reuser databases. The model does not capture other effects of data reuse, e.g., network effects of database products. In addition, the model also ignores factors that are specific to the kind of data being reused, e.g., privacy concerns when the reused data is about personal information, and increased price competition concerns when price data is reused.

In addition to relaxing the limitations identified above, we plan to look into a few other areas in future research. Out current analysis is based on a horizontal differentiation model; in the future, we plan to examine data reuse that is vertically differentiated, e.g., the reuser may produce a database of inferior or superior quality to target a different market. We also need to look at dynamic characteristics. As stressed in Landes and Posner (2003), intellectual property is also the input to intellectual property creation. With strong protection for database contents, the cost of database creation will likely rise. In addition, many online databases have characteristics of two-sided markets (Rocket and Tirole, 2003; Parker and Van Alstyne, 2005), e.g., they target both information seekers as well as advertisers. Therefore, the modeling techniques for two-sided markets and their interlinked network effects are worth exploring to derive new insights for policy formulation purposes. Nevertheless, the current model captures one of the major issues in database legislation and should be helpful to the formulation of a socially beneficial data reuse policy.

"The best way to predict the future is to invent it."
– Alan Kay

Chapter 7

Conclusion and Future Research

7.1 Conclusion

In this Thesis we have presented our interdisciplinary research on the effective use and reuse of information from disparate sources. We advance the technology to enable large scale semantic interoperability; we also analyze the data reuse policy issue that arises from systematic data reuse activities enabled by such technology.

We developed temporal extensions to the existing COIN framework to enable semantic interoperability when heterogeneity exists not only between sources and receivers, but also in different time periods within a data source. In describing the existing COIN framework, we identified and made concrete several important concepts that are missing or not well explained in previous COIN work. For example, for the first time we provided an operational definition for composite conversion and defined the *value* method based on composite conversion; the notion of modifier orthogonality was further clarified. For the temporal extensions, we developed and demonstrated a solution for representing and reasoning about time-varying semantics in the presence of heterogeneous temporal entities. Previously, no data conversions were executed in the mediation step. With time-varying semantics, conversions for ground temporal values need to be executed during mediation so that inconsistent temporal constraints can be detected for the purposes of semantic query optimization and of ensuring that an executable plan exists when a source has certain capability limitations.

For the first time, we performed a systematic evaluation on the flexibility and scalability of the COIN approach. Compared with several traditional approaches, the COIN approach has much greater flexibility and scalability.

For data reuse policy analysis, we developed an economic model that formalizes the policy instruments in one of the latest legislative proposals in the U.S. Our analysis indicates that depending on the cost level of database creation, the degree of differentiation of the reuser database, and the efficiency of policy administration, the socially beneficial policy choice can be protecting a legal monopoly, encouraging competition via compulsory licensing, discouraging voluntary licensing, or even allowing free riding. The results provide useful insights for designing and implementing a socially beneficial database protection policy.

7.2 Future Research

In the process of developing solutions for problems at hand, we identified a number of interesting issues worth exploring in the future.

The ontology and context descriptions are key components of the COIN framework. It is desirable to develop a systematic approach to the management and verification of large ontology and context definitions. For example, given an ontology and a set of context descriptions, it is useful to have a way of verifying if all necessary component conversions have been specified; or similarly, when modifier values are changed, identifying the component conversions that need to be updated.

The query answering mechanism is a two-step linear process: the mediator generates a mediated query that includes all conversion instructions; the query planner/optimizer/executioner (POE) takes that mediated query as input and generates an execution plan and executes it to obtain data instances. The conversion instructions primarily include conversions to the receiver context. For certain queries, an optimized plan needs instructions on converting to source contexts. We developed a solution in Chapter 4 for query optimization during the mediation step that generates conversions anticipated by the POE. Future research should investigate a more general approach, where the POE determines what data need to be converted to which context and then requests the mediator to generate the necessary conversion instructions. The POE and the mediator may need to communicate back and forth multiple times before an optimal plan is generated.

In Chapter 4, we chose to use dynamic modifiers to represent time-varying semantics. An alternative approach is to use dynamic context association. When a data field in the source is recycled, the same data field may map to different ontological concepts in different time periods. In this case, dynamic elevation (i.e., data field to ontology mapping) will be useful. Thus, future research should investigate dynamic context association as well as dynamic elevation.

Also in Chapter 4, in representing dynamic modifiers, we assumed that the semantics of temporal entities is not time-varying. When the assumption does not hold, we propose to use temporal entities with static semantics in the integration context to describe temporal contexts. The proposed solution needs further investigation.

The evaluation of the COIN approach in Chapter 5 is based on an analysis that uses the number of conversions as a proxy. In the future, it will be interesting to have empirical data on things such as how much time and effort it takes to implement a solution using COIN. Benchmarking comparison with other emerging solutions will be interesting as well.

For policy research, as mentioned in Chapter 6, we would like to investigate the double-sided market modeling technique because many data creators and data reusers do have two markets that have certain network effects.

As seen in Chapter 3, we have demonstrated the capability of the COIN technology using simple examples in financial and geo-political domains. In addition to further testing the technology with problems of the size comparable to real application scenarios, it is interesting to test the technology in other domains to identify new challenging issues. Possible domains include the integration of GIS systems, digital libraries, and healthcare systems.

Bibliography

Abdennadher, S., Frühwirth, T. and Meuss, H. (1999) "Confluence and Semantics of Constraint Simplification Rules", *Constraint Journal,* **4**(2), 133-165.

Abiteboul, S., Agrawal, R., Bernstein, P., Carey, M., Ceri, S., Croft, B., DeWitt, D., Franklin, M., Molina, H. G., Gawlick, D., Gray, J., Haas, L., Halevy, A., Hellerstein, J., Ioannidis, Y., Kersten, M., Pazzani, M., Lesk, M., Maier, D., Naughton, J., Schek, H., Sellis, T., Silberschatz, A., Stonebraker, M., Snodgrass, R., Ullman, J., Weikum, G., Widom, J. and Zdonik, S. (2005) "The Lowell database research self-assessment", *Communications of the ACM,* **48**(5), 111-118.

Alatovic, T. (2002) "Capabilities Aware Planner/Optimizer/Executioner", M.S. Thesis, Department of EECS, MIT.

Allen, J. F. (1983) "Maintaining Knowledge about Temporal Intervals", *Communications of the ACM,* **26**(11), 832-843.

Bacchus, F., Tenenberg, J. and Koomen, J. A. (1989) "A Non-Reified Temporal Logic", *First International Conference on Principles of Knowledge Representation and Reasoning*, 2-10.

Berbena, R.-P. and Jansenb, W. J. (2005) "Comovementnext term in international equity markets: A sectoral view", *Journal of International Money and Finance,* **24**(5), 832-857.

Berners-Lee, T., Hendler, J. and Lassila, O. (2001) "The Semantic Web", *Scientific American,* (May 2001), 34-43.

Bernstein, P. A., Melnik, S., Quix, C. and Petropoulos, M. (2004) "Industrial-Strength Schema Matching", *ACM SIGMOD Record,* **33**(4), 38-43.

Besen, S. and Raskind, L. (1991) "An Introduction to the Law and Economics of Intellectual Property", *Journal of Economic Perspectives,* **5**(1), 3-27.

Bettini, C., Wang, X. S. and Jajodia, S. (1998) "Temporal Semantic Assumptions and Their Use in Databases", *IEEE Transactions on Knowledge and Data Engineering,* **10**(2), 277-296.

Bettini, C., Jajodia, S. and Wang, X. S. (2000) *Time Granularities in Databases, Data Mining, and Temporal Reasoning*, Springer.

Bilke, A. and Naumann, F. (2005) "Schema Matching using Duplicates", *International Conference on Data Engineering (ICDE),* Japan.

Bratko, I. (2001) *Proglog Programming for Artificial Intelligence*, 3rd Ed., Addison-Wesley.

Bruijn, J. d., Martín-Recuerda, F., Manov, D. and Ehrig, M. (2005) "State-of-the-art survey on Ontology Merging and Aligning", SEKT (Semantically Enabled Knowledge Technology)

Bry, F. and Spranger, S. (2004) "Towards a Multi-Calendar Temporal Type System for (Semantic) Web Query Languages", *2nd International Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR'04),* St. Malo, France, 102-117.

Buchanan, J., M. and Yoon, Y. J. (2000) "SYMMETRIC TRAGEDIES: COMMONS AND ANTICOMMONS", *Journal of Law and Economics,,* **43**(1), 1-43.

Bunge, M. (1974a) *Semantics I: Sense and Reference*, D. Reidel Publishing Company, Boston.

Bunge, M. (1974b) *Semantic II: Interpretation and Truth*, D. Reidel Publishing Company, Boston.

Bunge, M. (1977) *Ontology I: The Furniture of the World*, D. Reidel Publishing Company, Boston.

Bunge, M. (1979) *Ontology II: A World of Systems*, D. Reidel Publishing Company, Boston.

Buvač, S. (1996) "Quantificational Logic of Context", *AAAI'96,* Menlo Park, California, 600-606.

Ceri, S., Gottlob, G. and Tanca, L. (1989) "What You Always Wanted to Know About Datalog (And Never Dared to Ask)", *IEEE Transactions on Knowledge and Data Engineering,* **1**(1), 146-166.

Colsten, C. (2001) "Sui Generis Database Right: Ripe for Review?" *The Journal of Information, Law and Technology,* **2001**(3).

Chen, Y., Png, I. (2003) Information Goods Pricing and Copyright Enforcement: Welfare Analysis. *Information Systems Research* **14**(1) 107-123.

Clark, K. (1978) "Negation as Failure", In Logic and Data Bases (Eds, Gallaire, H. and Minker, J.), Plenum, pp. 292-322.

Cox, P. T. and Pietrzykowski, T. (1986) "Causes of Events: Their Computation and Application", *8th International Conference on Automated Deduction,* Oxford, England, July 27 - August 1, 1986.

Das, A. K. and Musen, M. A. (2001) "A Formal Method to Resolve Temporal Mismatches in Clinical Databases", *AMIA Annual Symposium.*

Date, C. J., Darwen, H. and Lorentzos, N. A. (2003) *Temporal Data and the Relational Model: A Detailed Investigation into the Application of Interval Relation Theory to the Problem of Temporal Database Management*, Morgan Kaufmann Publishers.

Decker, H. (1996) "An extension of SLD by abduction and integrity maintenance for view updating in deductive databases", *Joint International Conference and Symposium on Logic Programming*, 157-169.

Denecker, M. and Kakas, A. C. (2002) "Abduction in Logic Programming", In *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part I*, Vol. LNCS 2407 (Eds, Kakas, A. C. and Sadri, F.), Springer Verlag, pp. 402-436.

Doughty, D. (2004) "The Achilles' Heel of Service-Oriented Architectures", *Business Integration Journal,* (September), 44-47.

Dyerson, C. (1994) "Temporal Indeterminacy", Department of Computer Science, University of Arizona.

Eiter, T., Gottlob, G. and Leone, N. (1997) "Abduction from Logic Programs: Semantic and Complexity", *Theoretical Computer Science,* **189,** 129-177.

El-Khatib, H. T., Williams, M. H., MacKinnon, L. M. and Marwick, D. H. (2000) "A framework and test-suite for assessing approaches to resolving heterogeneity in distributed databases", *Information & Software Technology,* **42**(7), 505-515.

Eshghi, K. and Kowalski, R. A. (1989) "Abduction Compared with Negation by Failure", *6th International Conference on Logic Programming,* Lisbon, 234-255.

Firat, A., Madnick, S. E. and Siegel, M. D. (2000) "The Cameleon Web Wrapper Engine", *Workshop on Technologies for E-Services (TES'00),* Cairo, Egypt.

Firat, A., Madnick, S. E. and Grosof, B. (2002) "Financial Information Integration in the Presence of Equational Ontological Conflicts", *12th Workshop on Information Technology and Systems (WITS),* Barcelona, Spain.

Firat, A. (2003) "Information Integration using Contextual Knowledge and Ontology Merging", PhD

Thesis, Sloan School of Management, MIT.

Frühwirth, T. (1996) "Temporal Annotated Constraint Logic Programming", *Journal of Symbolic Computation,* **22,** 555-583.

Frühwirth, T. (1998) "Theory and Practice of Constraint Handling Rules", *Journal of Logic Programming,* **37**(1-3), 95-138.

Frühwirth, T. and Abdennadher, S. (2003) *Essentials of Constraint Programming*, Springer Verlag.

Gallaher, M. P., O'Connor, A. C., Dettbarn, J. L. and Gilday, L. T. (2004) "Cost Analysis of Inadequate Interoperability in the U.S. Capital Facilities Industry", GCR 04-867, NIST

Galton, A. (1990) "A Critical Examination of Allen's Theory of Action and Time", *Artificial Intelligence,* **42**(2-3), 159-188.

Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J. and Widom., J. (1995) "Integrating and Accessing Heterogeneous Information Sources in TSIMMIS", *AAAI Symposium on Information Gathering,* Stanford, California, 61-64.

Goh, C. H. (1997) "Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Systems", Ph.D. Thesis, Sloan School of Management, MIT.

Goh, C. H., Bressan, S., Madnick, S. and Siegel, M. (1999) "Context Interchange: New Features and Formalisms for the Intelligent Integration of Information", *ACM TOIS,* **17**(3), 270-293.

Gruber, T. R. (1993) "A Translation Approach to Portable Ontology Specifications", *Knowledge Acquisition,* **5**(2), 199-220.

Guarino, N. (1997) "Understanding, building, and using ontologies", *International Journal of Human-Computer Studies,* **46**(2/3), 293-310.

Guha, R. V. (1995) "Contexts: a formalization and some applications", Stanford University

Guha, R. and McCarthy, J. (2003) "Varieties of Contexts", *CONTEXT 2003*, 164-177.

Guha, R., McCool, R. and Fikes, R. (2004) "Contexts for the Semantic Web", *ISWC'04 (LNCS 3298),* Japan, 32-46.

Halevy, A. Y. (2001) "Answering Queries using Views: A Survey", *VLDB Journal,* **10**(4), 270-294.

Halpern, J. Y. and Shoham, Y. (1986) "A Propositional Modal Logic of Time Intervals", *Symposium on Logic in Computer Science*, 279-292.

Hayes, P. J. (1996) "A Catalog of Temporal Theories", UIUC-BI-AI-96-01, University of Illinois

Heald, P. J. (2001) "The Extraction/Duplication Dichotomy: Constitutional Line Drawing in the Database Debate", *Ohio State Law Journal,* **62,** 933-944.

Heimonen, K. (2002) "Stock Market Integration: Evidence on Price Integration and Return Convergence", *Applied Financial Economics,* **12**(6), 415-429.

Heller, M. A. and Eisenberg, R. (1998) "Can Patents Deter Innovation? The Anticommons in Biomedical Research", *Science,* **280,** 698-701.

Heller, M. A. (1998) "The tragedy of the Anticommons: Property in the Transition from Marx to Markets", *Harvard Law Review,* **111,** 621-688.

Hernandez, M. A., Miller, R. J., Haas, L. M., Yan, L., Ho, C. T. H. and Tian, X. (2001) "Clio: A Semi-Automatic Tool For Schema Mapping", *ACM SIGMOD.*

Hobbs, J. R. (2002) "A DAML Ontology of Time", *LREC 2002 (Workshop on Annotation Standards* *for Temporal Information in Natural Language,* Las Palmas, Canary Islands, Spain, May 27, 2002.

Hobbs, J. R. and Pan, F. (2004) "An Ontology of Time for the Semantic Web", *ACM Transactions on Asian Language Processing (TALIP),* **3**(1), 66-85.

Hotelling, H. (1929) "Stability in Competition", *Economic Journal,* **39,** 41-57.

Hugenholtz, P. B. (2001) "The New Database Right: Early Case Law from Europe", *Nighth Annual Conference on International IP Law and Policy,* New York, April 19-20, 2001.

Hugenholtz, P. B. (2003) "Program Schedules, Event Data and Telephone Subscriber Listings under the Database Directive: The "Spin-Off" Doctrine in the Netherlands and elsewhere in Europe", *11th Annual Conference on International Law & Policy,* New York, April 24-25, 2003.

Imielinski, T. (1987) "Intelligent Query Answering in Rule Based Systems", *Journal of Logic Programming,* **4**(3), 229-257.

ISO (2000) "ISO 8601: Data elements and interchange formats - Information interchange - Representation of dates and times".

Jaffar, J. and Maher, M. J. (1994) "Constraint Logic Programming: A Survey", *Journal of Logic Programming,* **19/20,** 503-581.

Jensen, C. S. and Snodgrass, R. T. (1996) "Semantics of Time-Varying Information", *Information Systems,* **21**(4), 311-352.

Kakas, A. C., Kowalski, R. A. and Toni, F. (1993) "Abductive Logic Programming", *Journal of Logic and Computation,* **2**(6), 719-770.

Kakas, A. C. and Mancarella, P. (1993) "Constructive Abduction in Logic Programming", University of Pisa

Kakas, A. C., Michael, A. and Mourlas, C. (2000) "ACLP: Abductive Constraint Logic Programming", *Journal of Logic Programming,* **44**(1-3), 129-177.

Kamp, J. A. W. (1968) "Tense Logic and the Theory of Linear Order", Ph.D. Thesis, University of California (UCLA).

Kashyap, V. and Sheth, A. P. (1996) "Semantic and Schematic Similarities Between Database Objects: A Context-Based Approach", *VLDB Journal,* **5**(4), 276-304.

Kashyap, V. and Sheth, A. P. (2000) *Information Brokering Across Heterogeneous Digital Data : A Metadata-based Approach*, Springer.

Kiffer, M., Laussen, G. and Wu, J. (1995) "Logic Foundations of Object-Oriented and Frame-based Languages", *J. ACM,* **42**(4), 741-843.

Koboldt, C. (1997) "The EU-Directive on the legal protection of databases and the incentives to update: An economic analysis", *International Review of Law and Economics,* **17**(1), 127-138.

Koubarakis, M. (1994) "Foundations of Temporal Constraint Databases", Ph.D. Thesis, Department of Electrical and Computer Engineering, National Technical University of Athens.

Kowalski, R. A. and Sergot, M. J. (1986) "A Logic-based Calculus of Events", *New Generation of Computing,* **1**(4), 67-95.

Landes, W. M. and Posner, R. A. (2003) *The Economic Structure of Intellectual Property Law,* Belknap Press.

Lee, J. (1996) "Integration Information from Disparate Contexts: A Theory of Semantic Interoperability", PhD Thesis, Sloan School of Management, MIT.

Lenzerini, M. (2001) "Data Integration Is Harder than You Thought", *CoopIS*, 22-26.

Linn, A. (2000) "History of Database Protection: Legal Issues of Concern to the Scientific Community",

Lipton, J. (2003) "Private Rights and Public Policies: Reconceptualizing Property in Databases", *Berkeley Technology Law Journal,* **18**(Summer).

Lopez, I. F. V., Snodgrass, R. T. and Moon, B. (2005) "Spatiotemporal Aggregate Computation: A Survey", *IEEE Transactions on Knowledge and Data Engineering,* **17**(2), 271-286.

Lloyd, J. W. (1987) *Foundations of Logic Programming*, 2, Springer-Verlag.

Madhavan, J., Bernstein, P. A., Doan, A. and Alon Halevy (2005) "Corpus-Based Schema Matching", *21st International Conference on Data Engineering (ICDE'05),* Tokyo, Japan, 57-68.

Madhavan, J., Bernstein, P. A. and Rahm, E. (2001) "Generic Schema Matching with Cupid", *27th International Conference on Very Large Data Bases (VLDB)*, 49--58.

Madnick, S. E. and Siegel, M. D. (2002) "Seize the Opportunity: Exploiting Web Aggregation", *MISQ Executive,* **1**(1), 35-46.

Maiocchi, R. and Pernici, B. (1991) "Temporal Data Management Systems: A Comparative View", *IEEE Transactions on Knowledge and Data Engineering,* **3**(4), 504-524.

Marriott, K. and Stuckey, P. J. (1998) *Programming with constraints: an introduction*, MIT Press.

Maurer, S. M. and Scotchmer, S. (1999) "Database Protection: Is it Broken and Should We Fix it?" *Science,* **284**(May), 1129-1130.

Maurer, S. M. (2001) "Intellectual Property Law and Policy Issues in Interdisciplinary and Intersectoral Data Applications", *Data for Science and Society.*

McCarthy, J. and Hayes, P. J. (1969) "Some Philosophical Problems from the Standpoint of Artificial Intelligence", In *Machine Intelligence* (Eds, Meltzer, B. and Mitchie, D.), Edinburgh University Press, pp. 463-502.

McCarthy, J. (1987) "Generality in Artificial Intelligence", *Communications of the ACM,* **30**(12), 1030-1035.

McCarthy, J., Buvač, S. (1997) "Formalizing Context (Expanded Notes)", In *Computing natural language* (Eds, Aliseda, A., van Glabbeek, R. and Westerstahl, D.), Sanford University.

McDermott, D. V. (1982) "A Temporal Logic for Reasoning about Processes and Plans", *Cognitive Science,* **6**, 101-155.

Miller, R., Hernandez, M., Haas, L., Yan, L., Ho, C., Fagin, R. and Popa, L. (2001) "The Clio Project: Managing Heterogeneity", *SIGMOD Record,* **30**(1), 78-83.

Miller, R., Malloy, M. A. and Masek, E. (2003) "Transforming Tactical Messaging: Exploiting Web Information Standards for Interoperability", Inercom, 44(1), 50-51.

Nadel, M. S. (2000) "The Consumer Production Selection Process in an Internet Age: Obstacles to Maximum Effectiveness and Policy Options", *Harvard Journal of Law & Technology,* **14**(1).

Naiman, C. F. and Ouskel, A. M. (1995) "A classification of semantic conflicts in heterogeneous database systems", *Journal of Organizational Computing,* **5**(2), 167-193.

National Research Council (2002) *Making the Nation Safer: The Role of Science and Technology in Countering Terrorism*, The National Academies Press.

Nayak, P. (1994) "Representing Multiple Theories", *AAAI'94,* Menlo Park, California, 1154-1160.

Nogueira, V. B., Abreu, S. and David, G. (2004) "Towards Temporal Reasoning in Constraint Contextual Logic Programming", *MultiCPL'04 (3rd Internationl Workshop on Multiparadigm Constraint Programming Languages,* Saint-Malo, France, Sep. 6-10, 2004.

Özsoyoglu, G. and Snodgrass, R. T. (1995) "Temporal and Real-Time Databases: A Survey", *IEEE Transactions on Knowledge and Data Engineering,* **7**(4), 513-532.

O'Rourke, M. A. (2000) "Shaping Competition on the Internet: Who Owns Product and Pricing Information?" *Vanderbilt Law Review,* **53**(6), 1965-2006.

Parker, G. G. and Van Alstyne, M. (2005) "Two-Sided Network Effects: A Theory of Information Product Design", *Management Science,* (forthcoming).

Pierce, C. S. (1903) "Abduction and induction." In *Philosophical writings of Pierce* (Ed, Buchler, J.), (1955) Dover Publications, Inc, New York, pp. 150-156.

Rahm, E. and Bernstein, P. A. (2001) "A Survey of Approaches to Automatic Schema Matching", *VLDB Journal,* **10**(4), 334-350.

Ram, S. and Park, J. (2004) "Semantic Conflict Resolution Ontology (SCROL): An Ontology for Detecting and Resolving Data and Schema-Level Semantic Conflict", *IEEE Transactions on Knowledge and Data Engineering,* **16**(2), 189-202.

Reichgelt, H. (1989) "A Comparison of First order and Modal Logics of Time", In *Logic-Based Knowledge Representation* (Eds, Jackson, P., Reichgelt, H. and van Harmelen, F.), MIT Press, Cambridge, pp. 143-176.

Reichman, J. H. and Samuelson, P. (1997) "Intellectual Property Rights in Data?" *Vanderbilt Law Review,* **50,** 52-166.

Reichman, J. H. and Uhlir, P. F. (1999) "Database Protection at the Crossroads: Recent Developments and Their Impact on Science and Technology", *Berkeley Technology Law Journal,* **14**(Sping), 793-838.

Rochet, J.-C. and Tirole, J. (2003) "Platform Competition in Two-Sided Markets", *Journal of the European Economic Association,* **1**(4), 990-1029.

Roddick, J. F. (1995) "A survey of schema versioning issues for database systems", *Information and Software Technology,* **37**(7), 383-393.

Rosenthal, A., Seligman, L., Renner, S. and Manola, F. (2001) "Data Integration Needs an Industrial Revolution", *International Workshop on Foundations of Models for Information Integration (FMII-2001),* Viterbo, Italy, September.

Rosenthal, A., Seligman, L. and Renner, S. (2004) "From Semantic Integration to Semantics Management: Case Studies and a Way Forward", *ACM SIGMOD Record,* **33**(4), 44-50.

Ruse, H. G. (2001) "Electronic Aents and the Legal Protection of Non-creative Databases", *International Journal of Law and Information Technology,* **3**(3), 295-326.

Salop, S. C. (1979) "Monopolistic Competition with Outside Goods", *The Bell Journal of Economics,* **10**(1), 141-156.

Salop, S.C., Scheffman, D.T. (1987) "Cost-Raising Strategies". *J. Industrial Economics* **36**(1) 19-34

Samuelson, P. (1996) "Legal Protection for Database Contents", *Communications of the ACM,* **39**(12), 17-23.

Sanks, T. M. (1998) "Database Protection: National and International Attempts to Provide Legal Protection for Databases", *Florida State University Law Review,* **25,** 991-1016.

Schmalensee, R. and Thisse, J. F. (1988) "Perceptual Maps and the Optimal Location of New Products: An Integrative Essay", *International Journal of Research in Marketing,* **5,** 225-249.

Scotchmer, S. (1991) "Standing on the Shoulders of Giants: Cumulative Research and the Patent Law", *Journal of Economic Perspectives,* **5**(1), 29-41.

Seligman, L., Rosenthal, A., Lehner, P. and Smith, A. (2002) "Data Integration: Where Does the Time Go?" *IEEE Bulletin of the Technical Committee on Data Engineering,* **25**(3), 3-10.

Shahar, Y. (1994) "A Knowledge-Based Method for Temporal Abstraction of Clinical Data", Ph.D. Thesis, The Program in Medical Information Sciences, Stanford University.

Shanahan, M. (1989) "Prediction is deduction but explanation is abduction", *International Joint Conference on Artificial Intelligence*, 1055-1060.

Shapiro, C. and Varian, H. R. (1998) *Information Rules: A Strategic Guide to the Network Economy*, Harvard Business School Press.

Shoham, Y. (1987) "Temporal Logics in AI: Semantical and Ontological Considerations", *Artificial Intelligence,* **33**(1), 89-104.

Snodgrass, R. T. and Ahn, I. (1985) "A Taxonomy of Time in Databases", *ACM SIGMOD,* Austin, TX, 236-246.

Snodgrass, R. T. (Ed.) (1995) *The TSQL2 Temporal Query Language,* Kluwer Academic Publishers.

Soo, M. D. and Snodgrass, R. T. (1995) "Mixed Calendar Query Language Support for Temporal Constants", University of Arizona

Sterling, L. and Shapiro, E. (1994) *The Art of Prolog: Advanced Programming Techniques*, 2nd, The MIT Press.

Tabuchi, H. (2002) "International Protection of Non-Original Databases: Studies on the Economic Impact of the Intellectual Property Protection of Non-Original Databases", *CODATA 2002,* Montreal, Canada, Sept. 29 - Oct. 3, 2002.

Takeyama, L. N. (1994) "The Welfare Implications of Unauthorized Reproduction of Intellectual Property in the Presence of Demand Network Externalities", *The Journal of Industrial Economics,* **22**(2), 155-166.

Tirole, J. (1988) *The Theory of Industrial Organization*, The MIT Press, Cambridge, MA, USA.

Uschold, M. (2003) "Where are the semantics in the semantic web?" *AI Magazine,* **24**(3), 25-36.

Ventrone, V. and Heiler, S. (1991) "Semantic heterogeneity as a result of domain evolution", *SIGMOD Record,* **20**(4), 16-20.

Wache, H., Vögele, T., Visser, U., H. Stuckenschmidt, Schuster, G., Neumann, H. and Hübner, S. (2001) "Ontology-Based Integration of Information - A Survey of Existing Approaches", *IJCAI-01 Workshop: Ontologies and Information Sharing,* Seattle, WA, 108-117.

Wand, Y., Storey, V. C. and Weber, R. (1999) "An ontological analysis of the relationship construct in conceptual modeling", *ACM Transactions on Database Systems (TODS),* **24**(4), 494-528.

Wand, Y. and Weber, R. (1988) "An Ontological Analysis of Some Fundamental Information

Systems Concepts", *ICIS,* Minneapolis, Minnesota.

Yerneni, R. (2001) "Mediated Query Processing over Autonomous Data Sources", Ph.D. Thesis, Department of Computer Science, Stanford University.

Yoon, K. (2002) "The Optimal Level of Copyright Protection", *Information Economics and Policy,* **14,** 327-348.

Zhou, Q. and Fikes, R. (2000) "A Reusable Time Ontology", KSL-00-01, Stanford University

Zhu, H., Madnick, S. E. and Siegel, M. D. (2002a) "The Interplay of Web Aggregation and Regulations", *3rd International Conference on Law and Technology,* Cambridge, MA, USA, November 6-7.

Zhu, H., Madnick, S. and Siegel, M. (2002b) "Global Comparison Aggregation Services", 1st Workshop on E-Business, Barcelona, Spain.

Appendix: Source Code of the COIN Mediator

The COIN mediator is implemented using Prolog and CHR. Many people in the past contributed to the implementation of the mediator. Goh (1997) and Firat (2003) provide brief overviews to the implementation, but they did not include the source code in their theses.

For the convenience of future researchers who want to study the implementation, below we list the source code of the ACLP procedure (described in Chapter 4). It comes in two files: (1) "adbudct.pl", which is the Prolog code for the "abductive reasoning" part of ACLP; and (2) "constraint.chr", which is the CHR code for the "constraint handling" part of ACLP. In addition to the rules for solving general constraints (e.g., transitivity rules for "<" constraint in domains of real number and time), the CHR code also contains certain application/source specific rules (e.g., key constraints and capability constraints in a data source).

There are two versions of "abduction.pl" files. We show the version used this Thesis. The other version is from Firat (2003), which includes significant amount of extra code for dealing with ontology and application merging. The CHR code listed here also includes the CHR rules developed by Firat (2003) for symbolic equation solving.

In addition to the code of the ACLP procedure, the implementation also includes code that parses a SQL query into a Datalog query, translates a naïve Datalog query to a well-formed COIN query, and constructs a mediated Datalog query from the constraint stores of the ACLP procedure. Code for these for these features is not included here.

abduct.pl

```
/*****************************************************************************
*                                                                           *
*              Abduction Engine                                             *
*                                                                           *
*              From Cheng Goh by Stephane Bressan 09/15/96                  *
*              Modified by Aykut FIRAT, Harry Zhu                           *
*                                                                           *
*****************************************************************************/

:- module_interface(abduct).

:- export abduct2all/4.
:- export get_conflict_list2/1.

:- begin_module(abduct2).
:- local reference(conflict_list).
:- local reference(module), reference(context).
:- local variable(conflict_list_var).

abducible_relation(RName, Arity):- query_import_relation(_, RName, Arity).

query_import_relation(ISName, RName, Arity) :-
              getval(module,Application),
              call(rule(relation(ISName, RName,i,List, _),_),Application), length(List,
Arity).

query_import_relation(ISName, RName, Arity) :-
              getval(module,Application),
              call(rule(relation(ISName, RName,ie,List, _),_),Application), length(List,
Arity).

compiled_clause(H:-B):-
              getval(module,Application),
              call(rule(H, B), Application).
```

```
abduct2all(Query, Context, Answers, Module):-
        setval(module,Module),setval(context, Context),
        findall(Answer, abduct(Query, Answer), Answers).

abduct(':-'(Head, Goal), ':-'(Head,LDG)):-
        abductively_prove(Goal, LDG).

abduct([Head| Goal], [Head|LDG]):-
        abductively_prove(Goal, LDG).

abductively_prove(Goal, DG) :-
        sub_abductively_prove(Goal), getval(module,A),post_constraint('?do'(A)),
        abducted_list(DG).

% some builtins are not abducted
% and will be evaluated directly.

builtin(Lit):-
        functor(Lit, P, Ar),builtin(P, Ar).
builtin(atomic,1).
builtin(number,1).
builtin(string,1).
builtin(=..,2).
builtin(cste,4).
builtin(ground,1).
builtin(var,1).
builtin(fail,0).
builtin(containObj,2). %% for temporal context
builtin(contains, 2).
builtin(begins, 2).
builtin(ends, 2).

%%%%%%
% constraints are abducted
constraint(~=, 2, neq).
constraint(is,          2,      eq).
constraint(==,          2,      seq).
constraint(<,  2,       lss).
constraint(>,  2,       grt).
constraint(=<,          2,      leq).
constraint(>=,          2,      geq).
constraint(<>,          2,      neq).
constraint(plus, 3, sum).
constraint(minus,3, sub).
constraint(fplus,3, fsum).
constraint(fminus,3, fsub).
constraint(multiply, 3, mul).
constraint(divide, 3, div).
constraint(fmultiply, 3, fmul).
constraint(fdivide, 3, fdiv).
constraint(datecvt, 4, datecvt).

% and so are all edb predicates

abducible(Lit, LLit) :-
        functor(Lit, R, A),
        constraint(R,A, RR),!,
        Lit =.. [R|Arg],
        LLit =.. [RR|Arg].

abducible(Lit, '?'(Lit)) :-
        functor(Lit, R, A),
        abducible_relation(R,A).

%
sub_abductively_prove(true):-!.         % success

sub_abductively_prove((H,T)) :-!,       % subgoals
        sub_abductively_prove(H),
        sub_abductively_prove(T).
sub_abductively_prove([]):-!.

sub_abductively_prove([H|R]) :-!,     % subgoals
        sub_abductively_prove(H),
        sub_abductively_prove(R).

sub_abductively_prove('='(X,Y)) :-!,  % equality
        X = Y.

sub_abductively_prove(not('='(X,Y))) :-!,    % inequality 2
        X~=Y.
sub_abductively_prove(not('is'(X,Y))) :-!,   % inequality 3
        X~=Y.

/* predicates that need to be called directly
all_modifiers, attr, cvt, modifier, value, context,
*/
sub_abductively_prove(all_modifiers(X,Y)) :-!,       % Prolog to value
        all_modifiers(X,Y).
sub_abductively_prove(attr(X,Y,Z)) :- !,     % Prolog to value
        attr(X,Y,Z).
sub_abductively_prove(cvt(X0,X1,X2,X3,X4,X5,X6,X7,X8)) :-  !,       % Prolog to value
        cvt(X0, X1,X2,X3,X4,X5,X6,X7,X8).

sub_abductively_prove(modifier(S,O,M,C,SM)) :- !,
        modifier(S,O,M,C,SM).

sub_abductively_prove(value(X,Y,Z)) :- !,    % Prolog to value
        value(X,Y,Z).

sub_abductively_prove(if(X,Y,Z)) :- !,       %%% HZ: for part-of problem.
        (sub_abductively_prove(X) ->
        sub_abductively_prove(Y);sub_abductively_prove(Z)).

sub_abductively_prove(context(X,Y)) :- !,    % Prolog to value
        context(X,Y).

%% other literals

sub_abductively_prove(Lit) :- % builtin
        builtin(Lit), !,
        call(Lit).

sub_abductively_prove(Lit) :- % abduction in constraint store
        abducible(Lit, L1), !,
        post_constraint(L1).

sub_abductively_prove(Lit) :- % resolution
        compiled_clause(Lit :- Body),
        sub_abductively_prove(Body).

sub_abductively_prove({Lit}) :-
        call((Lit)).

%% more predicates to be directly called. For temporal
sub_abductively_prove(containObj(X,Y)):-
         containObj(X,Y).
sub_abductively_prove(contains(X,Y)) :-
         contains(X,Y).
sub_abductively_prove(begins(X,Y)) :-
         begins(X,Y).
sub_abductively_prove(ends(X,Y)) :-
         ends(X,Y).
sub_abductively_prove(sourcePos(O,P)):- sourcePos(O,P).
sub_abductively_prove(arg(I,T,E)):-arg(I,T,E).

/* constraint store (in module mystore) */

post_constraint(Lit) :- call(Lit, mystore). %true.
```

```
abducted_list([C|L]):- in_store(C),
                abducted_list(L),!.
abducted_list([]).

in_store(C):- call(chr_get_constraint(CC), mystore), reformat(CC, C).


reformat(round(X,Y,Z), Z is round(X,Y)).
reformat('?'(Y), Y).
reformat(lss(X, Y), X < Y).
reformat(leq(X, Y), X =< Y).
reformat(geq(X, Y), X >= Y).
reformat(grt(X, Y), X > Y).
reformat(neq(X, Y), '<>'(X, Y)).
reformat(eq(X, Y), X is Y).
reformat(seq(X,Y), '=='(X, Y)).
reformat(sum(X,Y,Z), Z is X + Y).
reformat(sub(X,Y,Z), Z is X - Y).
reformat(fsum(X,Y,Z), Z is X + Y).
reformat(fsub(X,Y,Z), Z is X - Y).
reformat(mul(X,Y,Z), Z is X * Y).
reformat(div(X,Y,Z), Z is X / Y).
reformat(fmul(X,Y,Z), Z is X * Y).
reformat(fdiv(X,Y,Z), Z is X / Y).
reformat(tle(X,Y), X =< Y).
reformat(tls(X,Y), X < Y).
reformat(datecvt(A,B,C,D),L):- B \==D, L = datecvt(A,B,C,D).

/* builtin context axioms */

cste(S, skolem(S, V, C, 1, cste(V)), C, V).
cste(_).

context(skolem(_, _, C, _, _), C).

isa(skolem(S, _Vsrc, _Csrc, _Pos, _Tuple), S).
isa(cste(S,_,_,_), S).

all_modifiers(basic,[]).
all_modifiers(S,L) :-
        sub_abductively_prove(modifiers(S,L1)),
        sub_abductively_prove(is_a(S,S1)),
        all_modifiers(S1,L2), union(L2,L1,L). % ToDo: code for ordering modifers

attr(X,Y,Z) :- compiled_clause(attr(X,Y,Z):-H), sub_abductively_prove(H).
attr(skolem(T,V1,V2,V3,V4),M,L):- sub_abductively_prove(is_a(T,P)),
attr(skolem(P,V1,V2,V3,V4),M,L).

cvt(X0,X1,X2,X3,X4,X5,X6,X7,X8):-
        compiled_clause(cvt(X0,X1,X2,X3,X4,X5,X6,X7,X8):-H), sub_abductively_prove(H).
cvt(X0,X1,X2,X3,X4,X7,X8,X5,X6):-
        compiled_clause(cvt(commutative,X1,X2,X3,X4,X5,X6,X7,X8):-H),
sub_abductively_prove(H).
cvt(X0,X1,X2,X3,X4,X5,X6,X7,X8):-
        sub_abductively_prove(is_a(X1,P)), cvt(X0,P,X2,X3,X4,X5,X6,X7,X8).

sourceCtxt(skolem(_S, _Vsrc, Csrc, _Pos, _Tuple), Csrc).

sourceValue(skolem(_S, Vsrc, _Csrc, _Pos, _Tuple), Vsrc).
sourceValue(cste(_S,_O,_C,V),V).

sourcePos(skolem(_S,_Vsrc,_Csrc,Pos,_Tuple),Pos).

value(skolem(_S, Vsrc, _Csrc, _Pos, _Tuple), none, Vsrc):-!, post_constraint('?df'(none)).
%value(skolem(_S, Vsrc, Csrc, _Pos, _Tuple), Csrc, Vsrc):-!.
value(cste(S, O, C,V),Ctxt,Val):- !,cste(S, O, C, V), value(O, Ctxt, Val).

value(O, Ctgt, Vtgt):-
       isa(O, S),
       sourceValue(O, Vsrc),
       sub_abductively_prove(all_modifiers(S, L)),
        allcvts(S, O, Vsrc, L, Ctgt, Vtgt).

modifier(S,O,M,C,SM) :- compiled_clause(modifier(S,O,M,C,SM):-B), sub_abductively_prove(B).
modifier(S,O,M,C,SM) :- sub_abductively_prove(is_a(C,CP)), modifier(S,O,M,CP,SM).
modifier(S,O,M,C,SM) :- sub_abductively_prove(is_a(S,SP)), modifier(SP,O,M,C,SM).

%% buit-in temporal context axioms
containObj(Int, O) :- sub_abductively_prove((
        attr(O, tempAttribute, T), contains(Int,T))).
begins(T1,[T1,T2]).
ends(T2,[T1,T2]).
contains(Int, Point):-
        begins(T1,Int), ends(T2, Int),getval(context, C),
        sub_abductively_prove((value(T1, C, T1v), value(T2, C, T2v),value(Point,C,Pv),
                               Pv =< T2v, T1v =<Pv)).

%%%%%

allcvts(_S, _O, V, [], _Ctgt, V):- !.
allcvts(S, O, V, [M|L], Ctgt, NV):-
        precvt(S, O, V, M, Ctgt, IV),
        allcvts(S, O, IV, L, Ctgt, NV).

precvt(S, O, V, M, Ctgt , IV) :-
        sourceCtxt(O, Csrc),
        sub_abductively_prove((modifier(S, O,M,Csrc,M1),
                modifier(S, O,M,Ctgt,M2),
                value(M1, Ctgt, MV1),
                value(M2, Ctgt, MV2),'<>'(MV1,MV2))),
                ((var(MV1);var(MV2)) ->
                subprecvt(S, O, M, Ctgt, MV1, V, MV2, IV);
                subprecvt(S, O, M, Ctgt, MV1, V, MV2, IV)). %,!).%%

precvt(S, O, V, M, Ctgt , V):-
        sourceCtxt(O, Csrc),
        sub_abductively_prove((modifier(S, O,M,Csrc,M1),
                modifier(S, O,M,Ctgt,M2),
                value(M1,Ctgt,MV1),ground(MV1), value(M2,Ctgt,MV1)
                %)),(M==dateformat ->
 (subprecvt(S,O,M,Ctgt,MV1,V,MV1,V),post_constraint('?df'(MV1)))),!.
                 )),(M==dateformat -> (subprecvt(S,O,M,Ctgt,MV1,V,MV1,V))),!.
precvt(S, O, V, M, Ctgt , V):-
        sourceCtxt(O, Csrc),
        sub_abductively_prove((modifier(S, O,M,Csrc,M1),
                modifier(S, O,M,Ctgt,M2),
                value(M2,Ctgt,MV2),ground(MV2), value(M1,Ctgt,MV2)
                %)),!.
                )),(M==dateformat -> (subprecvt(S,O,M,Ctgt,MV2,V,MV2,V))),!.

precvt(S, O, V, M, Ctgt , V):-
        sourceCtxt(O, Csrc),
        sub_abductively_prove((modifier(S, O,M,Csrc,M1),
                modifier(S, O,M,Ctgt,M2),
                value(M2,Csrc,MV2),ground(MV2), value(M1,Csrc,MV2)
                )),!.
                %)),(M==dateformat -> (subprecvt(S,O,M,Ctgt,MV2,V,MV2,V))),!.

precvt(S, O, V, M, Ctgt , V):-
        sourceCtxt(O, Csrc),
        sub_abductively_prove((modifier(S, O,M,Csrc,M1),
                modifier(S, O,M,Ctgt,M2),
                value(M1,Csrc,MV1), ground(MV1), value(M2,Csrc,MV1)
                )),!.
                %)),(M==dateformat -> (subprecvt(S,O,M,Ctgt,MV1,V,MV1,V))),!.

%%%%%%%%%%%%%%

%precvt(S, O, V, M, C , V). %hz commented out

subprecvt(S, O, M, Ctgt, MV1, V, MV2, IV):-
```

```
         %sub_abductively_prove(('<>'(MV1,MV2),cvt(X0,S, O, M, Ctgt,  MV1, V, MV2, IV))),!,

        sub_abductively_prove(cvt(X0,S, O, M, Ctgt,  MV1, V, MV2, IV)),!,
        sourceCtxt(O, Csrc),
        get_tuple(O,T),
        getConvFunc(M,CF),
        post_constraint('$'([S,V,T,M,Csrc,MV1,Ctgt,MV2,CF])).

subprecvt(S, O, M, Ctgt, MV1, Vs, MV2, Vt):-
        getval(module,Application),
        call(path(MV1, MV2, P, _D,Application,S,M), dijkstra),
         serially_prove(P, S, O, M, Vs, Vt),!,%%hz add !
        sourceCtxt(O, Csrc),
        get_tuple(O,T),
        getConvFunc(M,CF),
        post_constraint('$'([S,Vs,T,M,Csrc,MV1,Ctgt,MV2,CF])).

serially_prove([X|[]], S, O, M, Vs, Vs).

serially_prove([MV1,MV2|Rest], S, O, M, Vs, Vt) :- subprecvt(S, O, M, Ctgt, MV1, Vs, MV2,
IV),
        serially_prove([MV2|Rest], S, O, M, IV, Vt).


get_conflict_list2([C|L]):- in_store2(C),
                get_conflict_list2(L),!.
get_conflict_list2([]).

in_store2(C):- call(chr_get_constraint(CC), mystore), reformat2(CC, C).

reformat2('$'(Y), (Y)).

getConvFunc(M, CF) :- compiled_clause(cvt(_,_, _, M, _, _, _, _, _):-CF),!.
getConvFunc(M,none):- writeln("none").
get_tuple(skolem(_S, Vsrc, _Csrc, _Pos, T),T).
```

contraint.chr

```
% Basic INEQUALITIES for abduction
% Based on MInmax from thom fruehwirth & Pascal Brisset ECRC
% compile with lib(chr)
%handler abduction.

option(already_in_store, on).
option(already_in_heads, off).
option(check_guard_bindings, off).

:- import(date_time).
:- local reference(dateformat).

operator(700, xfx, lss).
operator(700, xfx, grt).
operator(700, xfx, neq).
operator(700, xfx, geq).
operator(700, xfx, leq).
operator(700, xfx, eq).
operator(700, fx, '?').
operator(700, fx, '$').
operator(700, xfx, seq).


constraints (leq)/2, (lss)/2, (neq)/2, (eq)/2, (seq)/2,('$')/1, ('?')/1, (sum)/3, (sub)/3,
(bound)/1, (fsum)/3, (fsub)/3, (fdsub)/6, (fdsum)/6, (fdmul)/6, (fddiv)/6, (div)/3 , (mul)/3,
(fmul)/3, (fdiv)/3, (round)/3, (datecvt)/4, ('?df')/1, (ftemp)/4, geq/2,grt/2, '?do'/1.

X geq Y <=> Y leq X.
X grt Y <=> Y lss X.

'?df'(F) <=>setval(dateformat,F).
```

```
/* tle <==>leq */

reflexivity @  X leq X <=> true.
antisymmetry @ X leq Y, Y leq X <=> X=Y.
transitivity @ X leq Y, Y leq Z ==> X \== Y, Y \== Z, X \== Z | X leq Z.
top_axiom    @ X leq Y <=> Y==top | true.
bottom_axiom @ Y leq X <=> Y==bottom | true.

%% subsumption rules tighten the constraints
tle_subsumption @   X leq N \ X leq M <=> number(N), number(M), N<M | true.
tle_subsumption_2 @ M leq X \ N leq X <=> number(N), number(M), N<M | true.
tle_conflict @ M leq X,  X leq N <=> number(N), number(M), N<M |fail.

tle_subsumption @ X leq N \ X leq M <=> string(N), string(M), tr(tls2,N,M) | true.
tle_subsumption_2 @ M leq X \ N leq X <=> string(N), string(M), tr(tls2,N,M) | true.
tle_conflict @ M leq X,  X leq N <=> string(N), string(M), tr(tls2,N,M) | fail.

tle_builtin    @ X leq Y <=> number(X),number(Y) | X =< Y.
tle_builtin2   @X leq Y <=> string(X), string(Y) | tr(tle2, X,Y).

/* tls */
irreflexivity @  X lss X <=> fail.
top_axiom    @ X lss Y <=> Y==top | true.
bottom_axiom @ Y lss X <=> Y==bottom | true.

transitivity @ X lss Y, Y lss Z ==> X \== Y, Y \== Z | X lss Z.
transitivity @ X leq Y, Y lss Z ==> X \== Y, Y \== Z | X lss Z.
transitivity @ X lss Y, Y leq Z ==> X \== Y, Y \== Z | X lss Z.

subsumption  @ X lss Y \ X leq Y <=> true.

subsumption  @ X lss N \ X lss M <=> number(N), number(M) | N@<M.
subsumption  @ M lss X \ N lss X <=> number(N), number(M) | N@<M.
tls_conflict @ M lss X,  X lss N <=> number(N), number(M), N<M |fail.

tls_subsumption @ X lss N \ X lss M <=> string(N), string(M), tr(tls2,N,M) | true.
tls_subsumption_2 @ M lss X \ N lss X <=> string(N), string(M), tr(tls2,N,M) | true.
tls_conflict @ M lss X,  X lss N <=> string(N), string(M), tr(tls2,N,M) | fail.

subsumption  @ X leq N \ X lss M <=> number(N), number(M) | N@<M.
subsumption  @ M leq X \ N lss X <=> number(N), number(M) | N@<M.
subsumption  @ X lss N \ X leq M <=> number(N), number(M) | N@<M.
subsumption  @ M lss X \ N leq X <=> number(N), number(M) | N@<M.
subsumption  @ X leq N \ X lss M <=> string(N), string(M) | tr(tls2,N,M).
subsumption  @ M leq X \ N lss X <=> string(N), string(M) | tr(tls2,N,M).
subsumption  @ X lss N \ X leq M <=> string(N), string(M) | tr(tls2,N,M).
subsumption  @ M lss X \ N leq X <=> string(N), string(M) | tr(tls2,N,M).


datecvt(D1,F1,D2,F2) <=> ground(D1),ground(F1),ground(F2),nonground(D2),
   date_string(D,F1,D1),date_string(D,F2,D2), setval(dateformat, F2)|true.
datecvt(D1,F1,D2,F2) <=> ground(D2),ground(F1),ground(F2),nonground(D1),
   date_string(D,F2,D2),date_string(D,F1,D1), setval(dateformat, F2)|true.
datecvt(D1,F1,D2,F2) ==>string(F2) | setval(dateformat, F2).

datecvt(D1,F1,D2,F2), datecvt(D1,F1,D,F2) ==>D2=D.
datecvt(D1,F1,D2,F2), datecvt(D,F1,D2,F2) ==> D1=D.

tls2(A,B) :- A @< B.

teq2([H1|T1], [H2|T2]) :- H1 == H2, !, teq2(T1,T2).
teq2([],_) :-!.
teq2(_,[]).

tle2(X,Y) :- tls2(X,Y),!; teq2(X,Y).

tr(R,N,M) :-  getval(dateformat, F),string(F),
 date_string(DN, F, N), date_string(DM, F, M),
 DN  =.. [R1|D1], DM =.. [R2|D2],
 Re=..[R,D1,D2],Re.
```

```
tr(R,N,F1, M,F2) :- string(F1), string(F2),
 date_string(DN, F1, N), date_string(DM, F2, M),
 DN  =.. [R1|D1], DM =.. [R2|D2],
 Re=..[R,D1,D2],Re.


/*** end of temporal ***/


/* neq */

built_in     @ X neq Y <=> X ~= Y | true.
       % can be replaced by ground(X),ground(Y) | X \= Y.
irreflexivity@ X neq X <=> fail.

subsumption  @ X neq Y \ Y neq X <=> true.
subsumption  @ X lss Y \ X neq Y <=> true.
subsumption  @ X lss Y \ Y neq X <=> true.

simplification @ X neq Y, X leq Y <=> X lss Y.
simplification @ Y neq X, X leq Y <=> X lss Y.


/* eq */

equations     @ X eq Y <=> ground(X), var(Y) |Y is X.
equations     @ X eq Y <=> var(X), ground(Y)| X is Y.
equations     @ X eq Y <=> var(X), var(Y) | Y = X.
equations     @ X eq Y <=> ground(X), ground(Y)| ZY is Y, ZX is X, ZX = ZY.
/* new addition*/
equations        @ X eq Y, Z eq Y ==> X = Z.

/* string equations */

sequations     @ X seq Y <=> ground(X), ground(Y)| ZY = Y, ZX = X, ZX = ZY.

sequations     @ X seq X <=> true.

simplification @ X neq Y, X leq Y <=> X lss Y.
simplification @ Y neq X, X leq Y <=> X lss Y.

sum_ground  @ sum(X,Y,Z) <=> ground(X), ground(Y) | Z is X + Y, bound(Z).
sum_ground  @ sum(X,Y,Z) <=> ground(X), ground(Z) | Y is Z - X, bound(Y).
sum_ground  @ sum(X,Y,Z) <=> ground(Y), ground(Z) | X is Z - Y, bound(X).

sum_identity  @ sum(0,Y,Z) <=>  nonground(Z), nonground(Y) | Z = Y.
sum_identity  @ sum(X,0,Z) <=>  nonground(Z), nonground(X) | Z = X.

sum_equality    @ sum(X,Y,Z), sum(X,A,Z) ==> nonground(Y), nonground(A) |  Y = A.
sum_equality    @ sum(A,Y,Z), sum(X,Y,Z) ==> nonground(X), nonground(A) | X = A.
sum_equality    @ sum(X,Y,A), sum(X,Y,Z) ==> nonground(Z), nonground(A) | Z = A.

sum_equality    @ sum(X,Y,Z), sum(X,A,Z) ==> nonground(Y), ground(A) | Y is A.
sum_equality    @ sum(A,Y,Z), sum(X,Y,Z) ==> nonground(X), ground(A) | X is A.
sum_equality    @ sum(X,Y,A), sum(X,Y,Z) ==> nonground(Z), ground(A) | Z is A.

sum_binding  @ sum(X,Y,Z), bound(X) ==> fdsum(X,Y,Z,X,0,0).
sum_binding  @ sum(X,Y,Z), bound(Y) ==> fdsum(X,Y,Z,0,Y,0).
sum_binding  @ sum(X,Y,Z), bound(Z) ==> fdsum(X,Y,Z,0,0,Z).

sum_symbolic    @ fdsum(X,Y,Z,X,0,0), fdsum(X,Y,Z,0,Y,0) <=> fsum(X,Y,Z), bound(Z).
sum_symbolic    @ fdsum(X,Y,Z,X,0,0), fdsum(X,Y,Z,0,0,Z) <=> fsub(Z,X,Y), bound(Y).
sum_symbolic    @ fdsum(X,Y,Z,0,Y,0), fdsum(X,Y,Z,0,0,Z) <=> fsub(Z,Y,X), bound(X).

sum_elimination @ fsum(X,Y,Z), sum(X,Y,Z) <=> fsum(X,Y,Z).
sum_elimination @ sum(X,Y,Z), sum(X,Y,Z) <=> sum(X,Y,Z).
sum_elimination @ fsum(X,Y,Z), fsum(X,Y,Z) <=> fsum(X,Y,Z).

sum_transformation  @ sum(X,Y,Z), bound(Z)  <=>   sub(Z,Y,X), bound(Z).
```

```
/* Subtraction Axioms*/

sub_ground   @ sub(X,Y,Z) <=> ground(X), ground(Y)  | Z is X - Y, bound(Z).
sub_ground   @ sub(X,Y,Z) <=> ground(X), ground(Z)  | Y is Z + X, bound(Y).
sub_ground   @ sub(X,Y,Z) <=> ground(Y), ground(Z)  | X is Z + Y, bound(X).

sub_identity   @ sub(X,0,Z) <=> nonground(Z), nonground(X)  | Z = X.
sub_identity   @ sub(X,Y,0) <=> nonground(Y), nonground(X)  | X = Y.

sub_equality   @ sub(X,Y,Z), sub(X,A,Z) ==> nonground(A), nonground(Y)| A = Y.
sub_equality   @ sub(A,Y,Z), sub(X,Y,Z) ==> nonground(A), nonground(X)| A = X.
sub_equality   @ sub(X,Y,A), sub(X,Y,Z) ==> nonground(A), nonground(Z)| A = Z.

sub_equality   @ sub(X,Y,Z), sub(X,A,Z) ==> nonground(A), ground(Y)| A is Y.
sub_equality   @ sub(A,Y,Z), sub(X,Y,Z) ==> nonground(A), ground(X)| A is X.
sub_equality   @ sub(X,Y,A), sub(X,Y,Z) ==> nonground(A), ground(Z)| A is Z.


sub_binding @ sub(X,Y,Z), bound(X) ==> fdsub(X,Y,Z,X,0,0).
sub_binding @ sub(X,Y,Z), bound(Y) ==> fdsub(X,Y,Z,0,Y,0).
sub_binding @ sub(X,Y,Z), bound(Z) ==> fdsub(X,Y,Z,0,0,Z).

sub_elimination @ fsub(X,Y,Z), sub(X,Y,Z) <=> fsub(X,Y,Z).
sub_elimination @ sub(X,Y,Z), sub(X,Y,Z) <=> sub(X,Y,Z).
sub_elimination @ fsub(X,Y,Z), fsub(X,Y,Z) <=> fsub(X,Y,Z).

sub_transformation   @ sub(X,Y,Z), bound(Z) <=> sum(Y,Z,X), bound(Z).

sub_symbolic   @ fdsub(X,Y,Z,X,0,0), fdsub(X,Y,Z,0,Y,0) <=> fsub(X,Y,Z), bound(Z).
sub_symbolic   @ fdsub(X,Y,Z,X,0,0), fdsub(X,Y,Z,0,0,Z) <=> fsub(X,Z,Y), bound(Y).
sub_symbolic   @ fdsub(X,Y,Z,0,Y,0), fdsub(X,Y,Z,0,0,Z) <=> fsum(Z,Y,X), bound(X).


/* Multiplication Axioms */
mul_ground   @ mul(X,Y,Z) <=> ground(X), ground(Y)  | Z is X * Y, bound(Z).
mul_ground   @ mul(X,Y,Z) <=> ground(X), ground(Z), X~=0 | Y is Z / X, bound(Y).
mul_ground   @ mul(X,Y,Z) <=> ground(Y), ground(Z), Y ~=0  | X is Z / Y, bound(X).

mul_ground   @ mul(X,Y,Z) <=> ground(X), ground(Y)  | Z is X * Y, bound(Z).
mul_ground   @ mul(0,Y,Z) <=> nonground(Z)  | Z is 0, bound(Z).
mul_ground   @ mul(X,0,Z) <=> nonground(Z)  | Z is 0, bound(Z).

mul_identity @ mul(1,Y,Z) <=> nonground(Z), nonground(Y)  | Z=Y.
mul_identity @ mul(X,1,Z) <=> nonground(Z), nonground(X)  | Z=X.

mul_equality   @ mul(X,Y,Z), mul(X,A,Z) ==> nonground(A), nonground(Y)| A = Y.
mul_equality   @ mul(A,Y,Z), mul(X,Y,Z) ==> nonground(A), nonground(X)|  A = X.
mul_equality   @ mul(X,Y,A), mul(X,Y,Z) ==> nonground(A), nonground(Z)|  A = Z.

mul_equality   @ mul(X,Y,Z), mul(X,A,Z) ==> nonground(A), ground(Y)| A is Y.
mul_equality   @ mul(A,Y,Z), mul(X,Y,Z) ==> nonground(A), ground(X)|  A is X.
mul_equality   @ mul(X,Y,A), mul(X,Y,Z) ==> nonground(A), ground(Z)|  A is Z.

mul_binding   @ mul(X,Y,Z), bound(X) ==> fdmul(X,Y,Z,X,0,0).
mul_binding   @ mul(X,Y,Z), bound(Y) ==> fdmul(X,Y,Z,0,Y,0).
mul_binding   @ mul(X,Y,Z), bound(Z) ==> fdmul(X,Y,Z,0,0,Z).

mul_symbolic   @ fdmul(X,Y,Z,X,0,0), fdmul(X,Y,Z,0,Y,0) <=> fmul(X,Y,Z), bound(Z).
mul_symbolic   @ fdmul(X,Y,Z,X,0,0), fdmul(X,Y,Z,0,0,Z) <=> fdiv(Z,X,Y), bound(Y).
mul_symbolic   @ fdmul(X,Y,Z,0,Y,0), fdmul(X,Y,Z,0,0,Z) <=> fdiv(Z,Y,X), bound(X).

mul_transformation   @ mul(X,Y,Z), bound(Z)   <=>   div(Z,Y,X), bound(Z).

mul_elimination @ mul(X,Y,Z), mul(X,Y,Z) <=> mul(X,Y,Z).
mul_elimination @ fmul(X,Y,Z), fmul(X,Y,Z) <=> fmul(X,Y,Z).
mul_elimination @ fmul(X,Y,Z), mul(X,Y,Z) <=> fmul(X,Y,Z).

/* Division Axioms*/
div_ground   @ div(X,Y,Z) <=> ground(X), ground(Y), Y ~=0 | Z is X / Y.
div_ground   @ div(X,Y,Z) <=> ground(X), ground(Z)  | Y is Z * X.
div_ground   @ div(X,Y,Z) <=> ground(Y), ground(Z)  | X is Z * Y.
div_ground  @ div(0,Y,Z) <=> nonground(Z) | Z is 0.
div_ground  @ div(X,0,Z) <=> true.

div_identity @ div(X,1,Z) <=> nonground(Z) | Z=X.
div_identity @ div(X,Y,1) <=> nonground(Y) | Y=X.

div_identity @ div(X,1.0,Z) <=> nonground(Z) | Z=X.
div_identity @ div(X,Y,1.0) <=> nonground(Y) | Y=X.

div_equality  @ div(X,Y,Z), div(X,A,Z) ==> nonground(A), nonground(Y)|  A = Y.
div_equality  @ div(A,Y,Z), div(X,Y,Z) ==>  nonground(A), nonground(X)| A = X.
div_equality  @ div(X,Y,A), div(X,Y,Z) ==> nonground(A), nonground(Z)| A = Z.

div_equality  @ div(X,Y,Z), div(X,A,Z) ==> nonground(A), ground(Y)|  A is Y.
div_equality  @ div(A,Y,Z), div(X,Y,Z) ==>  nonground(A), ground(X)| A is X.
div_equality  @ div(X,Y,A), div(X,Y,Z) ==> nonground(A), ground(Z)| A is Z.

div_binding @ div(X,Y,Z), bound(X) ==> fddiv(X,Y,Z,X,0,0).
div_binding @ div(X,Y,Z), bound(Y) ==> fddiv(X,Y,Z,0,Y,0).
div_binding @ div(X,Y,Z), bound(Z) ==> fddiv(X,Y,Z,0,0,Z).

div_symbolic  @ fddiv(X,Y,Z,X,0,0), fddiv(X,Y,Z,0,Y,0) <=> fdiv(X,Y,Z), bound(Z).
div_symbolic  @ fddiv(X,Y,Z,X,0,0), fddiv(X,Y,Z,0,0,Z) <=> fdiv(X,Z,Y), bound(Y).
div_symbolic  @ fddiv(X,Y,Z,0,Y,0), fddiv(X,Y,Z,0,0,Z) <=> fmul(Z,Y,X), bound(X).

div_transformation  @ div(X,Y,Z), bound(Z) <=> mul(Y,Z,X), bound(Z).
div_elimination @ div(X,Y,Z), div(X,Y,Z) <=> div(X,Y,Z).
div_elimination @ fdiv(X,Y,Z), div(X,Y,Z) <=> fdiv(X,Y,Z).
div_elimination @ fdiv(X,Y,Z), fdiv(X,Y,Z) <=> fdiv(X,Y,Z).

mul_sub_to_div_sum @ mul(X,A,Y), sub(B,Y,X) <=> div(B,N,X), sum(1,A,N).
mul_sum_to_div_sum @ mul(X,A,Y), sum(B,Y,X) <=> div(B,N,X), sub(1,A,N).
div_sub_to_mul_sum_div @ div(X,A,Y), sub(B,Y,X) <=> mul(A,B,N1), sum(1,A,N2), div(N1,N2,X).
div_sum_to_mul_sub_div @ div(X,A,Y), sum(B,Y,X) <=> mul(A,B,N1), sub(1,A,N2), div(N1,N2,X).

%% integrity constraints for gAggregate
hzshopper_ic1 @ '?'hzshopper(A,B1), '?'hzshopper(A,B2)
        ==> B1=B2.

hzg @ '?'hzshopper(A,B) ==> bound(A), bound(B).


'?''YHFrankfurt'(Q,P,T,M1,D1,Y1,M2,D2,Y2),'?''YHFrankfurt'(Q,P2,T,M1,D1,Y1,M2,D2,Y2) ==>
P=P2.

y1 @ '?''YHFrankfurt'(Q,P,T,M1,D1,Y1,M2,D2,Y2) , '?do'(application501)  ==>
   ftemp(['YHFrankfurt',Q,P,T,M1,D1,Y1,M2,D2,Y2],0,0,0).
y2 @ datecvt(Q,F1,A,F2) \ ftemp(['YHFrankfurt',Q,P,T,M1,D1,Y1,M2,D2,Y2],0,0,0)
<=>string(F2)|
     ftemp(['YHFrankfurt',Q,P,T,M1,D1,Y1,M2,D2,Y2],[Q,F1,A,F2],1,0).
y3 @ ftemp(['YHFrankfurt',Q,P,T,M1,D1,Y1,M2,D2,Y2],[Q,F1,A,F2],1,0) \A leq B <=> string(B) |
   datecvt(C,F1,B,F2), get_month(C,F1,M2),get_day(C,F1,D2),get_year(C,F1,Y2).
  %'?''YHFrankfurt'(Q,P,T,M1,D1,Y1,M2,D2,Y2).

y4 @ ftemp(['YHFrankfurt',Q,P,T,M1,D1,Y1,M2,D2,Y2],[Q,F1,A,F2],1,0) \B leq A <=> string(B) |
   datecvt(C,F1,B,F2), get_month(C,F1,M1),get_day(C,F1,D1),get_year(C,F1,Y1).

y5 @ ftemp(['YHFrankfurt',Q,P,T,M1,D1,Y1,M2,D2,Y2],[Q,F1,A,F2],1,0)  <=> A==B, string(B) |
   chr_get_constraint('?''YHFrankfurt'(Q,P,T,M1,D1,Y1,M2,D2,Y2)),
   datecvt(C,F1,B,F2), get_month(C,F1,M3),get_day(C,F1,D3),get_year(C,F1,Y3),
   '?''YHFrankfurt'(Q3,P,T,M3,D3,Y3,M3,D3,Y3).


'?''YHParis'(Q,P,T,M1,D1,Y1,M2,D2,Y2),
   '?''YHParis'(Q,P2,T,M1,D1,Y1,M2,D2,Y2) ==> P=P2.

y1 @ '?''YHParis'(Q,P,T,M1,D1,Y1,M2,D2,Y2) , '?do'(application501)  ==>
   ftemp(['YHParis',Q,P,T,M1,D1,Y1,M2,D2,Y2],0,0,0).
y2 @ datecvt(Q,F1,A,F2) \ ftemp(['YHParis',Q,P,T,M1,D1,Y1,M2,D2,Y2],0,0,0) <=>
     ftemp(['YHParis',Q,P,T,M1,D1,Y1,M2,D2,Y2],[Q,F1,A,F2],1,0).
y3 @ ftemp(['YHParis',Q,P,T,M1,D1,Y1,M2,D2,Y2],[Q,F1,A,F2],1,0) \A leq B <=> string(B) |
   datecvt(C,F1,B,F2), get_month(C,F1,M2),get_day(C,F1,D2),get_year(C,F1,Y2).

y4 @ ftemp(['YHParis',Q,P,T,M1,D1,Y1,M2,D2,Y2],[Q,F1,A,F2],1,0) \B leq A <=> string(B) |
   datecvt(C,F1,B,F2), get_month(C,F1,M1),get_day(C,F1,D1),get_year(C,F1,Y1).

'?''YHParis'(Q,P,T,M1,D1,Y1,M2,D2,Y2),
   '?''YHParis'(Q,P2,T,M1,D1,Y1,M2,D2,Y2) ==> P=P2.

y1 @ '?''YHNYSE'(Q,P,T,M1,D1,Y1,M2,D2,Y2) , '?do'(application501)  ==>
   ftemp(['YHNYSE',Q,P,T,M1,D1,Y1,M2,D2,Y2],0,0,0).
y2 @ datecvt(Q,F1,A,F2) \ ftemp(['YHNYSE',Q,P,T,M1,D1,Y1,M2,D2,Y2],0,0,0) <=>
     ftemp(['YHNYSE',Q,P,T,M1,D1,Y1,M2,D2,Y2],[Q,F1,A,F2],1,0).
y3 @ ftemp(['YHNYSE',Q,P,T,M1,D1,Y1,M2,D2,Y2],[Q,F1,A,F2],1,0) \A leq B <=> string(B) |
   datecvt(C,F1,B,F2), get_month(C,F1,M2),get_day(C,F1,D2),get_year(C,F1,Y2).

y4 @ ftemp(['YHNYSE',Q,P,T,M1,D1,Y1,M2,D2,Y2],[Q,F1,A,F2],1,0) \B leq A <=> string(B) |
   datecvt(C,F1,B,F2), get_month(C,F1,M1),get_day(C,F1,D1),get_year(C,F1,Y1).

join_ @ ftemp([R1,Q1,P1,T1,M11,D11,Y11,M12,D12,Y12],[Q1,F1,A,Fr],1,0),
        ftemp([R2,Q2,P2,T2,M21,D21,Y21,M22,D22,Y22],[Q2,F2,B,Fr],1,0) ==>
   R1 \== R2,A==B, atom_string(R1,R1S), atom_string(R2,R2S),
  joinable(R1S, R2S)|
   M11=M21,D11=D21,Y11=Y21,M12=M22,D12=D22,Y12=Y22.

joinable(R1, R2) :-
member(R1, ["YHNYSE", "YHFrankfurt", "YHParis"]),
member(R2,  ["YHNYSE", "YHFrankfurt", "YHParis"]).
```