SEMANTIC INFORMATION PREPROCESSING FOR NATURAL LANGUAGE INTERFACES TO DATABASES

by

Milan Mosny

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
November 1996

ISBN 0-612-17022-5

SIMON FRASER UNIVERSITY

PARTIAL COPYRIGHT LICENSE

Title of Thesis/Project/Extended Essay

Semantic Information Preprocessing for Natural Language Interfaces to Databases.

Author:

(signature)

Milan Mosny

(name)

November 19, 1996

(date)

APPROVAL

Name: Milan Mosny

Degree: Master of Science

Title of Thesis: Semantic Information Preprocessing for Natural Language Interfaces to Databases

Examining Committee: Dr. Ze-Nian Li
Chair

Dr. Fred Popowich
Senior Supervisor

Dr. Wo-Shun Luk
Supervisor

Dr. Veronica Dahl
Examiner

Date Approved: 14 November 1996

Abstract

A natural language interface to a database (NLID) needs both syntactic information about the structure of language and semantic information about what words and phrases mean with respect to the database. A semantic part of the NLID can implicitly or explicitly provide constraints on the input language. A parser can use these constraints to resolve ambiguities and to decrease overall response time. Our approach is to extract these constraints from the semantic description of the database domain and incorporate them semi-automatically or automatically into information directly accessible to the parser.

The advantage of this approach is a greater degree of system modularity, which usually reduces complexity, reduces the number of possible errors in the system and makes it possible to develop different parts of the system concurrently by different persons and thus reducing development time. Also domain independent syntactic information can be reused from domain to domain, then customized according to the semantic information from a specific domain.

To implement the idea, Abductive Equivalential Translation (AET) was chosen to describe the database related semantics. AET provides a formalism which describes how a "literal" logical form of an input sentence consisting of lexical predicates can be translated to a logical form consisting of predicates meaningful to the database engine. The information used in the translation process is a Linguistic Domain Theory (LDT) based on logic.

We shall constrain the expressive power of LDT to suit tractability and efficiency requirements and introduce Restricted Linguistic Domain Theory (RLDT). The main step for incorporation of semantic constraints into the syntax formalism is then extensive preprocessing of the semantic information described by an RLDT into Normalized Linguistic Domain Theory (NLDT). The system uses NLDT to produce selectional restrictions that follow from the semantic description of a domain by RLDT. Selectional restrictions in general

state which words can be immediately combined with which other words.

Once NLDT is constructed, it can be used as a main source of information for semantic processing of a sentence. Thanks to the soundness and completeness of the normalization process, the designer of the interface has possibility to express the semantic knowledge in more declarative terms.

KEYWORDS: Natural language processing, natural language interfaces, databases, logic programming, equivalences

Acknowledgements

I would like to thank Fred Popowich, Zuzke Repaskej, Dan Fass, Gary Hall, and my family.

Contents

List of Figures

Chapter 1

Introduction

Natural Language Interfaces to Databases (NLIDs) are systems that allow a user to access information stored in the database using natural language (e.g. English). This kind of access has several advantages over alternative methods as noted by [20].

- it provides an immediate vocabulary for talking about contents of the database
- it provides a means of accessing information in the database independently of its structure and encodings
- it shields the user from the formal access language of the underlying system
- it is available with minimum of training to both novice and occasional user

The NLIDs can be used to query and *update* a database. In this thesis, however, we shall concentrate solely on the task of querying the database.

In this chapter, we shall introduce NLIDs, describe architectures that the NLIDs use, and point out some problems that such systems have to deal with. Then we shall state what the main goal of this thesis is, namely, to develop preprocessing algorithms. We shall also briefly outline how the preprocessing algorithms can address some of the problems associated with NLIDs.

1.1 Common Principles

A number of different NLIDs with different underlying architectures were developed in the past, but certain common features can be observed.

Most of the NLID systems work with an assumption that natural language presented at the input of the system is at least to some degree compositional, that is the meaning of the whole can be represented as a function of the meaning of the parts.

Given an input sentence in natural language, the system creates some kind of structure for the sentence that describes how the parts of the language are combined together. The structure reflects the system's view of natural language. Some systems work with the phrase structure of the sentence, other systems use simple template matching, yet other systems produce a logical representation of the sentence. To create the structure, no information about the database schema - neither names of the tables and columns, nor any information about the meaning of the data stored in the database - is needed. What is needed is information about what kind of words and phrases can occur in human language that the system happens to process, how they can be combined together and how to assign a structure to the particular combination of the words and phrases. This information can contain morphological constraints, syntactic constraints and vocabulary. We shall call this information syntactic knowledge.

According to the structure of the sentence, a database query is constructed. During this process, information about the given domain, information about the structure of the database and domain independent semantic constraints are used. We shall call this information semantic knowledge. The semantic knowledge in general describes, how the database query can be constructed. It describes how the parts of the input sentence structure can be mapped into the concepts known to the database and how the mapped parts can be combined together to create the result. As an example consider a sentence

$$\textit{Who works in the department whose manager is Tony.} \tag{1.1}$$

The system can produce the following logical formula that can be considered as a structure of the sentence reflecting words and phrases which modify or complement other words or phrases

$$\exists X, E, D, M.(person(X) \wedge work(E, X) \wedge in(E, D)$$
$$\wedge department(D) \wedge possesive(D, M) \wedge manager(M) \wedge tony(M)) \tag{1.2}$$

Such a representation can be later translated into the database query using domain independent knowledge about mapping the quantifiers and logical conjunction into the database query, domain dependent knowledge about disambiguating certain predicates according to

their context (e.g. $possesive(D, M)$ in the context of manager and department into the relation *manages*) and database dependent knowledge about relating disambiguated predicates to tables and columns of the database.

1.2 Problems with Ambiguity

Often more than one structure for a given sentence can be produced. As an example consider a sentence from [12]

$$Lawyers\ give\ poor\ free\ legal\ advice. \tag{1.3}$$

where the word "poor" can refer either to "poor people that receive free legal advice" or to the quality of "free legal advice". Unfortunately, there are some domains where far more than two readings of the input sentence can be generated. Consider the phrase from [6]

$$trouble\ call\ activity\ total \tag{1.4}$$

that can be interpreted as

$$\begin{aligned} &(trouble(call\ activity))total\ \ OR \\ &((trouble\ call)activity)total\ \ OR \\ &(trouble\ call)(activity\ total)\cdots \end{aligned} \tag{1.5}$$

or even a more "hairy" phrase from the same source

$$the\ may\ 1992\ eastern\ divisions\ sub\ owned\ eqp\ fault\ count\ total\ stats \tag{1.6}$$

with many more possibilities. The processing of such phrases can lead to a combinatorial explosion resulting in long processing time and/or numerous inappropriate interpretations.

1.3 Systems Architectures

Based on the interaction between syntactic and semantic knowledge, we describe three types of NLIDs - tightly coupled systems, loosely coupled systems, and systems with hybrid architectures.

Tightly Coupled Systems

In tightly coupled systems, it is difficult to separate syntactic and semantic knowledge. The schema of such systems is depicted in fig. 1.1.

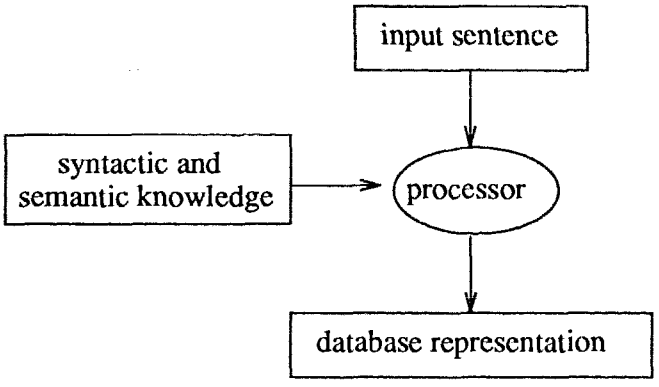

Figure 1.1: Architecture of a tightly coupled system

One of the advantages of this architecture is that syntactic and semantic knowledge are used together during the processing of the query. The architecture allows the use of more constraints to disambiguate the input sentence during initial processing than other architectures. The amount of constraints affects the number of produced representations, therefore performance is better.

On the other hand the lesser degree of modularity increases the difficulty of maintaining the system and it is difficult to reuse some parts of the interface while porting to another domain. Examples of such systems are [6], [27] and [10].

Loosely Coupled Systems

In loosely coupled systems, the clear distinction between syntactic and semantic knowledge can be observed. The schema of a loosely coupled system is depicted on fig. 1.2. According to [9], LUNAR can be considered an example of a loosely coupled system.

The clear advantage of this kind of architecture is its modularity. It is possible to reuse syntactic knowledge while porting the interface to another domain. It is also possible for different people to work on the syntactic information and semantic information independently. A higher degree of modularity also decreases the difficulty of maintaining the system.

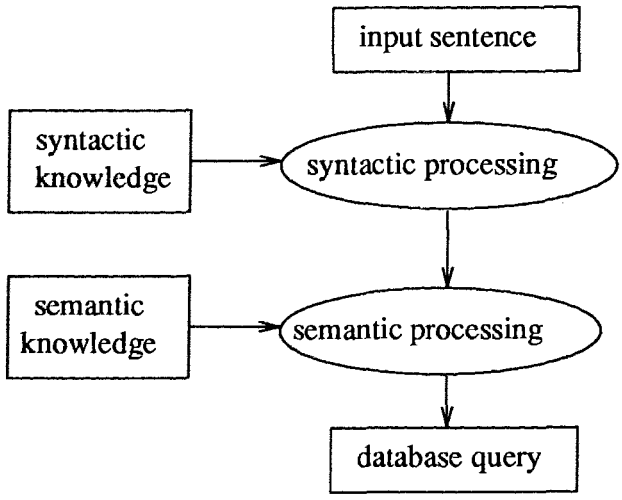

Figure 1.2: Architecture of loosely coupled system

The disadvantage of this architecture is its performance. In some domains it is difficult to use this kind of architectures, because of combinatorial explosion during syntactic processing (semantic knowledge is not available at that time to constrain analysis).

Systems with Hybrid Architecture

The hybrid architecture combines the modularity of loosely coupled systems with the performance of tightly coupled systems. The simplified architecture of one such system, CLARE [22], is shown in fig. 1.3.

The only disadvantage is the need to supply special semantic constraints (e.g. selectional restrictions) that are usually implicitly contained in the semantic knowledge. The extraction process needs human supervision and can introduce some errors. The extraction process can also be time consuming and special semantic constraints have to be updated anytime the semantic knowledge of the system changes.

1.4 What is this Thesis about

In this thesis, we shall explore a variation of the hybrid architectures. Our main goal is to create the basics of a system that can *automatically* derive reasonable semantic constraints that can be used for disambiguation during the syntactic processing out of a truly declarative

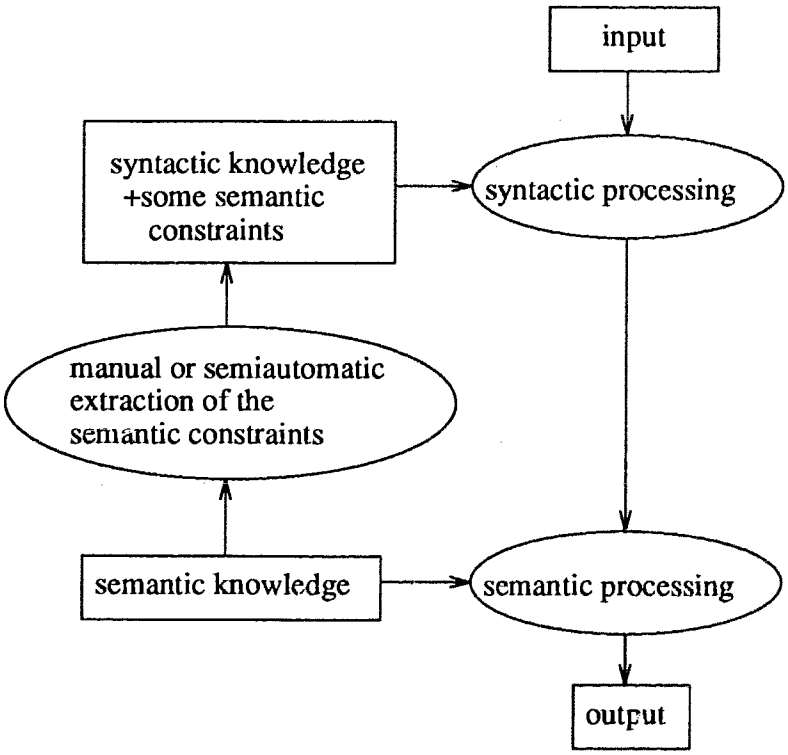

Figure 1.3: Architecture of system with hybrid architecture

description of the semantic knowledge.

We shall also show how the semantic knowledge expressed in declarative terms can be "compiled" into a data-structure called a *search graph* that provides the basis for semantic processing with reasonable efficiency and degree of declarativeness.

We believe that by solving the main goal we shall also address a trade-off between performance and the degree of modularity and portability that can be observed in the architectures described in previous section.

1.4.1 System's Architecture

The architecture of our system is depicted in fig. 1.4. We shall call the part of the system that contains syntactic knowledge (together with selectional restrictions) and syntactic processor *the syntactic part* and the part that contains semantic knowledge in suitable form and semantic processor *the semantic part.*

Essential to the system is the description of syntactic and semantic knowledge. Unification based grammars were chosen for describing syntactic knowledge, while a modified version of the formalism used in Abductive Equivalential Translation (AET) [23] was chosen to describe the database related semantics.

1.4.2 Syntactic Knowledge and Syntactic Processor

A unification based grammar is able to produce a "literal" logical representation of an English sentence. In such "literal" logical representations, predicates of atomic formulas approximately correspond to the content words of the input sentence. We shall call such predicates that occur in the "literal" logical representations of sentences *lexical predicates* and formulas whose atomic subformulas contain only lexical predicates *lexical formulas.* As an example consider a question

$$\textit{Is there a girl that loves chocolates from Germany?} \tag{1.7}$$

The "literal" logical representation of the sentence can be a lexical formula

$$\begin{aligned} \exists G, E, C, Ge.(&girl(G) \wedge love(E, G, C) \wedge chocolate(C) \\ &\wedge from(C, Ge) \wedge country(Ge, germany)) \end{aligned} \tag{1.8}$$

with lexical predicates *girl*, *love*, *chocolate*, *from*, and *country*.

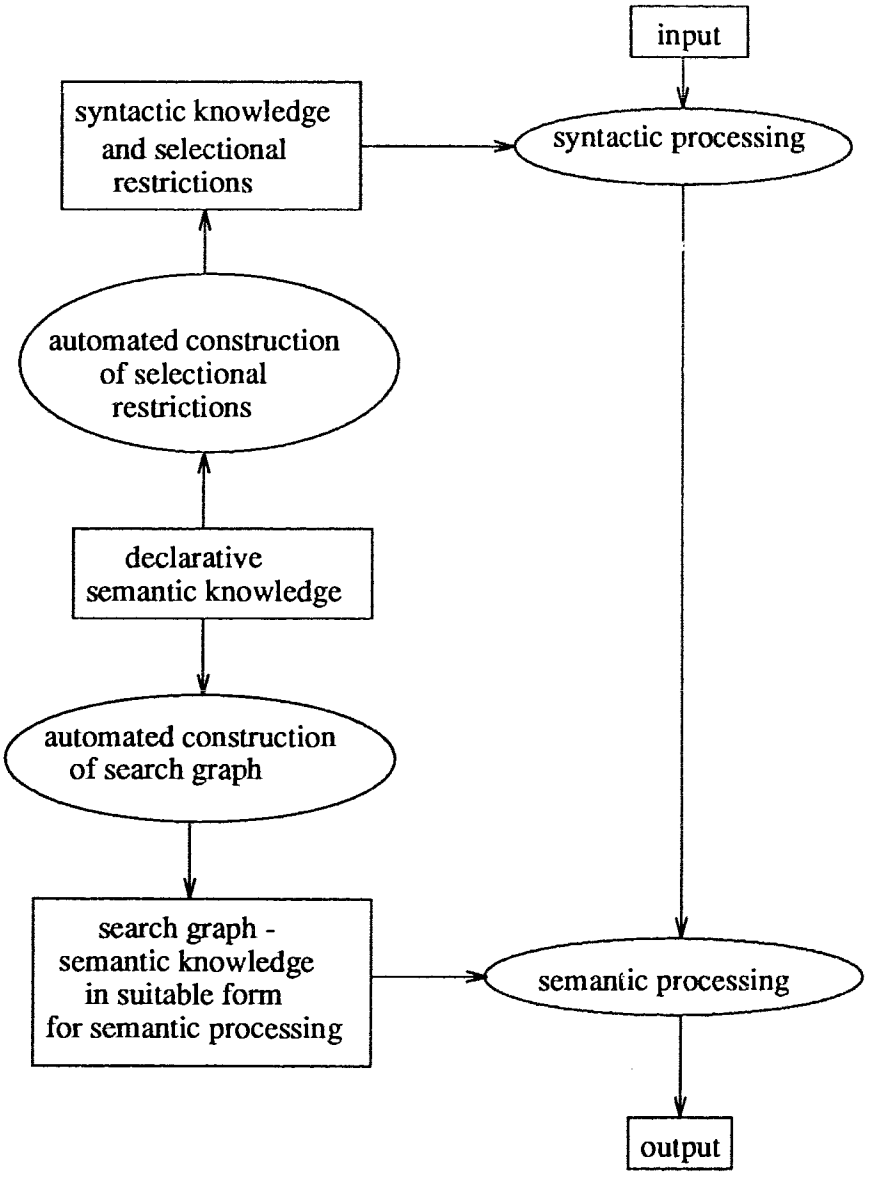

Figure 1.4: System's architecture.

1.4.3 Declarative Semantic Knowledge and Restricted Linguistic Domain Theories

AET provides a formalism that describes how lexical formulas that correspond to the input sentences can be translated into logical formulas that contain predicates understandable by the database (*database formulas*). The main source of semantic information for AET process is linguistic domain theory (LDT). LDT contains a set of axioms characterizing the relationship between lexical atomic formulas and database atomic formulas. The axioms describe in which context a particular atomic formula can be translated into another formula. Lexical atomic formulas can be translated through a number of intermediate formulas into the database formulas.

For example, LDT can contain axioms that can translate atomic formulas

- $girl(G)$ into $db\_person(G, w)$ in any context
- $chocolate(C)$ into $db\_product(C, chocolate)$ in any context
- $country(Ge, Name)$ into to $db\_country(Ge, Name)$ in any context
- $love(E, G, C)$ into $db\_likes(E, G, C)$ if the word love occurs in the context of *db_person* and *db_product*
- preposition $from(C, Ge)$ into $db\_location(C, Ge)$ if the word from occurs in the context of *db_location* and *db_product*

In this thesis we restrict the possible form of axioms used in original LDTs to suit the requirements of the preprocessing algorithms. For the same reason, we shall also impose a requirement of explicit finite dictionary. The dictionary relates atomic formulas that can be presented at the input of the semantic processor and atomic formulas that can be produced by the semantic processor. The LDT that uses constrained form of axioms and contains finite dictionary is called *restricted LDT (RLDT).*

RLDT, similarly to LDT, expresses a relationship between logic formulas that represent the natural language utterance and logic formulas that represent the database side of the NLID. Because RLDT technically works on slightly different kinds of logic formulas than original LDT, we shall call logic formulas that are expected at the input of a system that uses RLDT *input logic formulas* and formulas that are produced by such system *output logic formulas.*

The constraints imposed on the RLDT restrict expressive power of the theory and require more explicit information to be provided (e.g. the dictionary). On the other hand, implementation of the system compensates for the lack of expressiveness in RLDT as opposed to LDT by providing a set of simple and useful tools.

1.4.4 Automated Construction of Selectional Restrictions

The system is able to produce selectional restrictions that follow from the semantic description of a domain by RLDT. Selectional restrictions in general state which words can be immediately combined with which other words. The advantages of selectional restrictions is the possibility of very efficient implementation and their ability for seamless integration with the parser.

The process of constructing the selectional restrictions involves extensive preprocessing of the RLDTs. The preprocessing produces *normalized LDTs (NLDTs)*. The NLDTs contain all possible rules of translation for each input atomic formula. Each rule contains an input atomic formula, a pattern of possible contexts in which the input atomic formula can be translated and an output atomic formula representing the outcome of translation. The normalization process is sound, (i.e. the rules are logical consequences of the original RLDT) and complete (i.e. all the interesting rules of the given form are derived).

Using the patterns of contexts in the rules, the system can derive how the words can be combined together in the given domain. The constraints are then automatically incorporated into the unification based grammar.

Basic ideas for constructing of the selectional restrictions are also briefly introduced in our work [17].

1.4.5 Search Graph, Automatic Construction of Search Graph and Semantic Processing

Once an NLDT is constructed, it can be used as a main source of information for semantic processing of a sentence. Because the NLDT contains all possible rules for translation of input atomic formulas and the rules are represented in a straightforward fashion (patterns of possible context of input atomic formula in a sentence) the translation process is simplified. That is, each input atomic formula with its context in the input sentence is simply matched against the rules of the NLDT. If the matching rule is found, the input formula is substituted

with its output counterpart from the rule.

Thanks to the soundness and completeness of the normalization process, the designer of the interface has a possibility to express the semantic knowledge in more declarative terms.

Because rules of NLDTs correspond to natural language phrases, some special features of NLDTs can be observed. These features allow us to construct heuristics and a data structure for more efficient rule search. The data-structure is similar to the trie structures [1] used for pattern matching. We shall call the data structure a search graph.

The search graph construction and semantic processing algorithm is discussed in chapter 4.

1.5 Structure of this Thesis

AET is explained in more detail in the chapter 2 of the thesis. Section 2.1 provides an introduction to the main concepts and ideas of the AET. Section 2.2 compares AET with other approaches, namely Horn clause approaches, noninferential equivalential translation and interpretation as abduction. Section 2.3 introduces the AET from the formal point of view and section 2.4 describes LDTs in more detail.

Chapter 3 talks about restricted LDTs and the normalization process. The actual restrictions and modifications imposed on the LDTs (to suit the requirements of the preprocessing algorithms) together with the formal definition of RLDTs are described in section 3.2. Section 3.3 discusses and defines the NLDTs. Section 3.4 introduces the normalization algorithms and section 3.5 points to some issues of their implementation.

Chapter 4 describes the search graphs and the algorithm that uses them for translation of input formulas into output formulas. Chapter 5 explains in detail construction of the selectional restrictions that are based on NLDTs as well.

Chapter 6 summarizes the advantages and disadvantages of the proposed system and suggests some future directions of research.

Chapter 2

Abductive Equivalential Translation

2.1 Introduction

In this chapter, we shall provide a short overview of Abductive Equivalential Translation (AET) and Linguistic Domain Theories (LDTs) [23], the theory upon which our work is based. In his Ph.D. thesis, Rayner introduces AET and LDTs as a way of relating the semantics of natural language to the semantics of a database.

He supposes the existence of a language engine that provides a mapping from a natural language utterance into a *lexical logical formula.* The lexical logical formula is considered to be "literal" in the sense that there is a close correspondence between component words of a natural language expression and component symbols in its logical representation. We shall call predicates that occur in lexical logical formulas *lexical predicates.*

On the database side, a similar correspondence is established. Database objects, functions and relations expressible in a database query language correspond to quantifiers, predicates, functions and constants in *database logical formulas.* We shall call predicates that occur in the database logical formulas *database predicates.* An example of lexical formulas and database formulas was given in subsection 1.4.1.

The task of finding the relationship between database queries and natural language utterances is then simplified to finding the relationship between lexical logical formulas and database logical formulas. The idea is that given a lexical logical formula, the system will

find an equivalent database logical formula w.r.t. some background logical theory. Formally, a database logical formula F_{db} is called the translation of a lexical logical formula F_{ling} in theory Γ iff

$$\Gamma \Rightarrow (F_{ling} \equiv F_{db}) \tag{2.1}$$

The term *abductive*, as found in the name AET, is usually associated with an abductive reasoning process, where the main goal is to find explanations or conditions with the lowest cost under which the given fact can be satisfied in the theory [11]. During such reasoning processes, the theorem prover that tries to prove the goal can use not only the axioms of the given theory, but also unproven assumptions (with given costs) that are reasonable to assume. This idea is implemented in AET, but from a slightly different perspective. The difference is that only the lexical logical formula F_{ling} of the whole goal $(F_{ling} \equiv F_{db})$ is known at the beginning of the process. The system given theory gamma and lexical logical formula F_{ling} tries to find simultaneously database logical formula F_{db} and the cheapest set of assumptions A, such that

$$\Gamma \cup A \Rightarrow (F_{ling} \equiv F_{db}) \tag{2.2}$$

2.2 AET vs. Other Theories

In this section we shall describe very briefly other approaches to NLID semantic interpretation, namely the *Horn clause approach* [15], [19]; *noninferential equivalential translation* [4], [25]; and *interpretation as abduction* [11]. Then we compare these other approaches to the AET. Our comparison is based on that in [23].

Horn clause approaches

The main idea of this approach is to view a database as a Prolog database consisting of atomic formulas. The interpretation is achieved using theories consisting of Horn clauses. The theories allow one to define the interpretation of words and phrases in terms of database atomic formulas.

The reasoning engines built on top of Horn clause theories are well understood, allow reasonable expressive power and still provide efficient implementations. The problem is that the relationship imposed by Horn clause theories on input and output is implication rather

than equivalence. That means that the systems are not able to distinguish between "No" and "Don't know" answers. Although the Closed World Assumption can be invoked for this distinction, according to [23] it is difficult to write Horn clause theories for nontrivial domains.

Noninferential equivalential translation

Noninferential equivalential translation approaches do not use general inferential methods. Systems are usually based on inheritance hierarchies or simple domain models and context disambiguation is provided by some kind of type checking. These systems provide less expressive power than systems based on logical inferential methods. On the other hand, they provide equivalential translation.

Interpretation as abduction

Interpretation as abduction systems are Horn clause based systems with abductive reasoning. The advantage is that the system provides a simple way to handle defaults that is not present in two previous approaches. However, similarly to the Horn clause approach, the equivalence between input and output is not guaranteed.

We choose AET as a base for semantic processing in our system since it in a sense combines all three previously mentioned approaches. It provides nonmonotonicity, a reasonably efficient reasoning mechanism, the expressive power of logic based systems and still guarantees logical equivalence between input and output, if the translation can be found. Failure to find a translation is considered as a "Don't know" answer.

2.3 AET Formally

In this section, we shall present the AET translation algorithm. In our presentation, we shall show concepts and algorithms first by way of an example or by a very informal description of what we think could have been a motivation for a precise solution. After the informal introduction, we shall define the concepts or algorithms precisely.

We begin with an informal introduction of the *conjunctive context* (or *full conjunctive context*). Although the formal definition of this concept is not needed to define AET processes (and was not provided in Rayner's original work), the concept of conjunctive context is one of the most important ideas behind AET.

Then we describe the actual translation process (or algorithm) for finding a database logic formula F_{db} given a logic theory describing the semantics of a database and a lexical logical formula F_{ling}.

2.3.1 Conjunctive Context

The conjunctive context provides a reasonable "environment" or "context" for particular words or phrases in the input sentence. Speaking in logic terms, the conjunctive context defines a logical environment for instances of atomic formulas in the logical representations of the sentence. The translation process can then use this contextual information to disambiguate the meaning of ambiguous words (or atomic formulas that represent these words) within a sentence. The conjunctive context consists of two parts. The first part, called the *global conjunctive context*, contains world knowledge represented by logical axioms or data stored in the database. The second part is called *the local conjunctive context.* Under the assumption that logical formulas do not contain implications, the local conjunctive context of a given subformula contains formulas that are connected to it by the conjunctive $\wedge$.

Example 2.1 Consider, for example, the sentence

*Is there a student who takes cmpt*710 *or cmpt*720? (2.3)

whose logical representation could be

$$\begin{aligned} &\exists X, E, Y, Y_1.\mathit{student}(X) \wedge \\ &(\mathit{take}(E, X, Y) \wedge \mathit{unknown\_word}(Y, \mathit{cmpt}710) \vee \\ &\mathit{take}(E, X, Y_1) \wedge \mathit{unknown\_word}(Y_1, \mathit{cmpt}720)) \end{aligned} \tag{2.4}$$

The atomic formula $\mathit{student}(X)$ denotes that X refers to a student, atomic formula $\mathit{take}(E, X, Y)$ denotes that event E describes an action of "taking" referent Y by referent X. Atomic formula $\mathit{unknown\_word}(X, S)$ denotes that referent X can be described by word S, but the actual semantics of the word S is unknown.

The local conjunctive context for the atomic formula representing word "cmpt-710" does not contain only the fact that "it is taken" but also says that "the taking is done by a student". Using the logical notation and our informal definition of local conjunctive context, the local conjunctive context of the atomic formula $unknown(Y, cmpt710)$ would be a formula $student(X) \wedge take(E, X, Y)$.

The global conjunctive context is common for all atomic subformulas of the formula (2.4). It consists of data stored in the database and axioms describing relationships between the university world (described by lexical predicates like *student* or *take*) and the database world (represented by predicates that describe tables). ∎

2.3.2 Translation Process

The main goal of the translation process is to find a database logic formula F_{db} given a logic theory describing the semantics of a database and a lexical logical formula F_{ling}.

We shall show how these schemas work on a simplified version of sentence (2.3):

$$Is\ there\ a\ student\ that\ takes\ cmpt710? \tag{2.5}$$

and its logic representation

$$\begin{gathered} F_{ling} = \exists X, E, Y.student(X) \wedge \\ take(E, X, Y) \wedge unknown\_word(Y, cmpt710) \end{gathered} \tag{2.6}$$

Let us suppose that the global conjunctive context is represented by theory Γ. The theory Γ describes world knowledge which consists of the axiom

$$\begin{gathered} student(X) \wedge take(E, X, Y) \rightarrow \\ (unknown\_word(Y, S) \equiv db\_course(Y, S)) \end{gathered} \tag{2.7}$$

It says that if an object referenced by unknown word is taken by a student, then the object is actually a course. We shall call axioms similar to (2.7) *conditional equivalences*. The part

$$student(X) \wedge take(E, X, Y)$$

is called a ***condition***, the part

$$unknown\_word(Y, S)$$

is called *a left hand side (LHS)* of the conditional equivalence and the part

$$db\_course(Y, S)$$

is called *a right hand side (RHS)* of the conditional equivalence.

In this example, we shall use axiom (2.7) to show how the atomic formula

$$unknown\_word(Y, S)$$

can be translated (or disambiguated) in the lexical logical formula (2.6). In other words, we shall find formula F s.t. in the formula F, atomic formula $unknown\_word(Y, cmpt710)$ from the input sentence is translated into an atomic formula with a database predicate. F also has to be equivalent to formula (2.6) w.r.t. the theory Γ.

The translation process that uses the idea of conjunctive context relies on translation schemas (2.8) and (2.9)

$$\begin{aligned} & Context \wedge Q \rightarrow (P \equiv P') \\ \Rightarrow\ & Context \rightarrow (P \wedge Q \equiv P' \wedge Q) \end{aligned} \tag{2.8}$$

$$\begin{aligned} & Context \rightarrow (\theta(P) \equiv \theta(P')) \\ \Rightarrow\ & Context \rightarrow (\exists \vec{x}.P \equiv \exists \vec{x}.P') \end{aligned} \tag{2.9}$$

where θ substitutes $\vec{x}$ by unique constants w.r.t. P, P' and $Context$.

We shall proceed as follows. We shall recurse down the formula (2.7) using the schemas (2.8) and (2.9) until we reach the atomic formula with predicate *unknown_word*. During this recursive process the local conjunctive context will be collected. Then we use the theory Γ and the collected local conjunctive context to translate or disambiguate the unknown word.

Translation schema (2.9) says that it is possible to conclude equivalence between lexical logical formula (2.6) and formula F that we are looking for

$$\begin{aligned} & (\exists E, X, Y.student(X) \wedge take(E, X, Y) \\ & \wedge unknown\_word(Y, cmpt710)) \equiv F \end{aligned} \tag{2.10}$$

if we find formula $F1(e, x, y)$ s.t. (2.11) holds

$$\begin{aligned} & (student(x) \wedge take(e, x, y) \\ & \wedge unknown\_word(y, cmpt710)) \equiv F1(e, x, y) \end{aligned} \tag{2.11}$$

and set $F = \exists E, X, Y.(F1(E, X, Y))$. In (2.11), existentially quantified variables E, X,Y were substituted by constants e, x, y respectively and the existential quantifier was removed.

The translation schema (2.9) can be regarded then as a rule that allows us to recurse down the input formula through existential quantification. Translation schema (2.8), on the other hand, allows us to recurse through conjunction $\wedge$ while collecting the local conjunctive context.

According to translation schema (2.8) it is possible to conclude equivalence (2.11) if we find a formula $F2$ s.t. (2.12) holds

$$student(x) \rightarrow (take(e, x, y) \wedge unknown\_word(y, cmpt710) \equiv F2) \tag{2.12}$$

and set $F1(e, x, y) = student(x) \wedge F2$. Using translation schema (2.8) again, we can conclude (2.12) if the (2.13) is true

$$student(x) \wedge take(e, x, y) -> (unknown\_word(y, cmpt710) \equiv F3) \tag{2.13}$$

and we set $F2 = F3 \wedge take(e, x, y)$. That means, that is is possible to find the formula F, if it is possible to find a translation $F3$ of

$$unknown\_word(y, cmpt710)$$

using local conjunctive context

$$student(x) \wedge take(e, x, y)$$

Now it is time to introduce another translation schema (2.14).

$$\begin{aligned} & (Conds \rightarrow (P_1 \wedge P_2 \wedge P_3 ...) \equiv P') \wedge \\ & Context \rightarrow \theta(P_2 \wedge P_3 ... \wedge Conds) \\ \Rightarrow \quad & Context \rightarrow (\theta(P_1) \equiv \theta(P')) \end{aligned} \tag{2.14}$$

The formula $Context$ of the translation schema represents the local conjunctive context and formula P_1 is an atomic formula being translated. The first line of the translation schema is an axiom of the form of conditional equivalence from the global conjunctive context. The second line of the translation schema says that the conditions of the axiom from the global conjunctive context has to be implied by the local conjunctive context (w.r.t. to the global conjunctive context). The third line of the schema (2.14) says that under conditions

described by the first and second line of the schema it is possible to substitute LHS of the conditional equivalence by its RHS.

Let us put $Context = student(x) \wedge take(e, x, y)$. It is obvious that (2.15) holds

$$Context \rightarrow (student(X) \wedge take(E, X, Y)) \quad \{X \rightarrow x, Y \rightarrow y, E \rightarrow e\} \tag{2.15}$$

Using conditional equivalence (2.7) as a first line of the schema (2.14), and using (2.15) as the second line of the schema, we can conclude that (2.13) holds with

$$F3 = db\_course(y, cmpt710)$$

The rewritten formula is shown as

(2.16)

$$student(x) \wedge take(e, x, y) \rightarrow (unknown\_word(y, cmpt710) \equiv db\_course(y, cmpt710)) \tag{2.16}$$

The assignment $F3 = db\_course(y, cmpt710)$ yields

$$F = \exists E, X, Y.(student(X) \wedge take(E, X, Y) \wedge db\_course(Y, cmpt710)$$

that represents the final translation.

In addition to schemas (2.8), (2.9), and (2.14), Rayner in [23] also provides schemas for translating formulas that contain negations, disjunctions, implications and universal quantifiers.

$$\begin{aligned} & Context \rightarrow (P \equiv P') \\ \Rightarrow\ & Context \rightarrow (P \vee Q \equiv P' \vee Q) \end{aligned} \tag{2.17}$$

$$\begin{aligned} & Context \rightarrow (P \equiv P') \\ \Rightarrow\ & Context \rightarrow (\neg P \equiv \neg P') \end{aligned} \tag{2.18}$$

$$\begin{array}{rl} & Context \rightarrow (P \equiv P') \\ \Rightarrow & Context \rightarrow (P \rightarrow Q \equiv P' \rightarrow Q) \end{array} \tag{2.19}$$

$$\begin{array}{rl} & Context \wedge P \rightarrow (Q \equiv Q') \\ \Rightarrow & Context \rightarrow (P \rightarrow Q \equiv P \rightarrow Q') \end{array} \tag{2.20}$$

$$\begin{array}{rl} & Context \rightarrow (\theta(P) \equiv \theta(P')) \\ \Rightarrow & Context \rightarrow (\forall \vec{x}.P \equiv \forall \vec{x}.P') \end{array} \tag{2.21}$$

where θ substitutes $(\vec{x})$ by unique constants w.r.t. P, P' and $Context$.

Note, that equivalent but syntactically different formulas can yield different conjunctive contexts. (The idea of conjunctive context and translation that uses it is not "complete".) The simplest example is probably a pair of formulas $\neg a \vee b$ and $a \rightarrow b$. In the first case, the translation schemas provide the empty conjunctive context for atomic formula b whereas in the second case, the resulting conjunctive context contains atomic formula a.

Schemas for translating existential quantifiers and higher order formulas follow.

$$\begin{array}{rl} & \exists \vec{x}.P_1 \equiv P' \\ \Rightarrow & \theta(\exists \vec{x}.P_1) \equiv \theta(P') \end{array} \tag{2.22}$$

where θ is one-to-one on $\vec{x}$ (i.e. it does not map distinct variables into identical terms).

$$\begin{array}{rl} & Context \rightarrow (\theta(P) \equiv \theta(P')) \\ \Rightarrow & Context \rightarrow (count(N, \lambda X.P) \equiv count(N, \lambda X.P')) \end{array} \tag{2.23}$$

where θ replaces X with a unique constant. $count(N, \lambda X.P)$ holds if there are precisely N values of A such that $P(A)$ holds.

$$\begin{array}{rl} & Context \rightarrow (\theta(P) \equiv \theta(P')) \\ \Rightarrow & Context \rightarrow (sum(S, \lambda X.P) \equiv sum(S, \lambda X.P')) \end{array} \tag{2.24}$$

where θ replaces X with a unique constant. $sum(S, \lambda X.P)$ holds if all the objects A of which $P(A)$ holds are summable quantities, and S is their sum.

$$\begin{aligned} & Context \rightarrow (\theta(P) \equiv \theta(P')) \\ \Rightarrow \quad & Context \rightarrow (order(Selected, \lambda X.\lambda D.P(X,D), Ordering) \equiv \\ & order(Selected, \lambda X.\lambda D.P'(X,D), Ordering)) \end{aligned} \tag{2.25}$$

where θ replaces X and D with unique constants. Formula

$$order(Selected, \lambda X.\lambda D.P(X,D), Ordering)$$

holds if $Ordering$ is an ordering relation, $Dmax$ is the maximal D under the relation $Ordering$ such that $P(X,D)$ holds for some X, and $Selected$ is such an X.

The translation proceeds one predicate at a time. Each translation step substitutes an atomic formula with another formula until a formula consisting entirely of database predicates is reached. The translation step can be done only according to one of the conditional or existential equivalences of the theory.

The original definition of the algorithm (from [23]) follows

Recurse: Descend through F using the translation-schemas, until an atomic sub-formula A is reached. During this process, a conjunctive context E has been accumulated in which conditions will be proved, and some bound variables will have been replaced by unique constants. Constants resulting from bound variables are specially marked if i) they come from existentially bound variables that occur only in A, and ii) the only connectives between A and the binding existential quantifier are conjunctions. We will refer to these as *marked existential constants.*

Translate-universal: Find a rule $(H \wedge R \equiv B) \leftarrow C$ such that H unifies with A with m.g.u. θ. If it is then possible to prove $\theta(R \wedge C)$, replace A with $\theta(B)$. The leaves of the proof may include elements of the conjunctive context E, facts from the database, Horn clauses from the linguistic domain theory Γ, or Horn clause readings of conditional equivalences from Γ.

Translate-existential: Find a rule $\exists \vec{x}.H \equiv B$ such that H unifies with A with m.g.u. θ. If all the elements of $\theta(\vec{x})$ are distinct marked existential constants (in the sense defined immediately above in the **Recurse** step), then replace A with $\theta(B)$.

Simplify: if possible, apply simplifications to the resulting formula.

2.4 Linguistic Domain Theories

In this section, we shall talk a little bit more about the structure of linguistic domain theories (LDTs). These are the theories that contain all the domain knowledge needed by the AET translation algorithm to translate lexical logical formulas into their database equivalents. The LDTs contain a logical part (an example is a theory Γ from the previous section), assumption declarations that are used for abductive reasoning purposes and functional declarations used for simplification of results.

The kind of information contained in LDTs can be described from different points of view. We shall start with a technical description. Then we shall discuss the relationship of different kinds of predicates to stages of translation, and relationship of different kinds of axioms to their reusability potential.

2.4.1 Technical Description

In this description, P, P_1, P_n, R_1, R_l will stand for atomic formulas and $Cond$ will stand for a conjunction of atomic formulas.

Logical Part

The logical part of an LDT can be divided into a positive theory and a negative theory. Together, they form a logic theory Γ that is used for reasoning about the domain.

A positive theory contains Horn clauses of the form

$$P_1 \wedge \cdots \wedge P_n \rightarrow P \tag{2.26}$$

conditional equivalences of the form

$$Cond \rightarrow (P_1 \wedge \cdots P_n \equiv \exists X_1 \cdots X_m.(R_1 \wedge \cdots \wedge R_l)) \tag{2.27}$$

and existential equivalences of the form

$$\exists X_1 \cdots X_n.(P_1 \equiv P) \tag{2.28}$$

A negative theory contains rules of the form

$$P_1 \rightarrow \neg P \tag{2.29}$$

Assumption Declarations

LDT also contains assumption declarations of the form

```
assumable(Goal, Cost, Justification, Type, Conds)
```

The semantics of the assumption declaration is that goal *Goal* can be assumed at cost *Cost* in the context, where *Conds* hold. *Justification* is a tag presented to the user when the assumption is taken. *Type* is an atom specifying the type of the assumption. As an example consider an assumption declaration

```
assumable(
        car_is_company_car,
        0,
        all_cars_referred_to_are_company_cars,
        specialization,
        true)
```

and the conditional equivalence

$$car\_is\_company\_car \rightarrow (car(X) \equiv db\_company\_car(X)) \tag{2.30}$$

The atomic formula *car_is_company_car* is an assumption that is needed to translate lexical atomic formula $car(X)$ into its database counterpart. The assumption can be assumed at cost 0. Its type is "specialization", meaning that the assumption allows a user to use a word in its specialized sense (e.g the word "car" refers to company car). The specified condition is *true*, i.e. this assumption can be used in any context. When the assumption was taken the user is notified by the tag

$$all\_cars\_referred\_to\_are\_company\_cars$$

Functional Declarations

Besides the logic theory and assumption declarations, there are *functional declarations* that are used for simplification of the intermediate and final results of translation. The form of a functional declaration is

```
function(Template, FunctionalArgs -> RemainingArgs)
```

Such a declaration says that the relation represented by *Template* is a function from *FunctionalArgs* to *RemainingArgs*. The use of the functional declaration can be illustrated by the following example. Consider the sentence

$$Was\ John's\ transaction\ done\ on\ Saturday? \tag{2.31}$$

with an underlying database containing the predicate

$$db\_transaction(TransactionId, Payee, Date)$$

Let us also suppose that in a certain stage of translation the algorithm achieves intermediate representation

$$\begin{gathered}\exists TransationId, Date, Payee. \\ (db\_transaction(TransactionId, john, Date) \wedge \\ db\_transaction(TransactionId, Payee, saturday))\end{gathered} \tag{2.32}$$

Formula

$$db\_transaction(TransactionId, john, Date)$$

represents the construction "john's transaction" and the formula

$$db\_transaction(TransactionId, Payee, saturday)$$

represents the construction "transaction on Saturday". Using the functional declaration

```
function(db_transaction(TransId, Payee, Date),
        [TransId] -> [Payee, Date])
```

the formula (2.32) can be simplified to

$$\exists TransationId.db\_transaction(TransactionId, john, saturday)$$

2.4.2 Predicates and the Stage of Translation

The main idea of the AET translation process is to translate an input lexical logical formula containing the lexical predicates directly related to the actual words of an input sentence to the database formula containing the database predicates that can be easily evaluated by the database. During the translation process, the formula can also contain *intermediate predicates*, i.e. predicates that are neither database related nor lexical.

In this section, we shall talk about different kinds of predicates and when such predicates usually occur in the translation process. We start with predicates that occur at the end of translation process and then we proceed towards predicates that are closer to lexical predicates. Note, that the structuring of predicates is only approximate. It does not mean that LDT cannot contain any other types of predicates or that translation has to proceed according to the order described in this section.

Database Predicates

There are three kinds of predicates that can be considered as "final" or database predicates. The first kind of predicates are *table* predicates that directly correspond to relational tables and their arguments correspond to the attributes of the tables. An example of a table predicate is

$$db\_transaction(TransactionId, Payee, Date)$$

from the example in the previous section.

The second kind of database predicates are predicates related to arithmetic operations that can be performed on the database objects such as addition, subtraction, greater, smaller etc.

The third kind of predicates are dereferable predicates. These are the predicates that the system can cause to hold in the future. An example of the executable predicate is

$$execute\_in\_future(Action)$$

which is true if system performs the *Action* in the future.

Conceptual Predicates

Very close to the table predicates are conceptual predicates. These predicates almost correspond to the database tables. Their purpose is to "change" a database design to reflect better the requirements of natural language input. As an example consider sentences (2.33) and (2.34)

$$Who\ takes\ cmpt720\ at\ SFU? \tag{2.33}$$

$$Who\ takes\ cmpt720\ at\ UBC? \tag{2.34}$$

These sentences might be encountered when using a database that describes courses taken only at SFU without any attributes about the place where the courses are taken. In such cases, it is convenient to introduce a conceptual predicate that contains an attribute describing the place. The lexical logical formulas are first translated into formulas that use conceptual predicates and then the formulas are (or, in the case of (2.34) and the SFU database, are not) mapped into the database logical formulas. The conceptual predicate can also cover other sentences which contain information that is implied in the database e.g. (2.35) or (2.36)

$$\textit{Is John studying at SFU?} \tag{2.35}$$

$$\textit{What is the most popular course at SFU?} \tag{2.36}$$

As a second example consider a database that contains transaction IDs and check IDs. The fact that a transaction ID is the same as a check ID does not mean that the IDs represent the same things. Therefore the mapping from conceptual predicates into database predicates also takes care of mapping terms in conceptual predicates (e.g. $check\_id\#ID$), that represent real objects in the real world, into terms in database predicates, that represent database objects - strings, numbers, etc. As an example consider an axiom

$$conceptual\_check(check\_id\#ID, date\#Date) \equiv db\_check(ID, Date) \tag{2.37}$$

Mapping from conceptual predicates into database predicates should also reflect the condition imposed on records from the table. Suppose that we have a database that contains records from the last two years. While answering questions similar to

$$\textit{Show me all the students that took cmpt720.} \tag{2.38}$$

the mapping has to make an assumption that the facts talked about are no older than two years. Such an assumption should also be communicated to the end user.

Attribute Predicates

If we consider, for example, temporal prepositions, it is clear that they often take events or noun phrases as their arguments, but they usually impose conditions on the time when those events occur. The sentence

$$\textit{Who took more than ten courses during summer 95?} \tag{2.39}$$

is an example. An LDT introduces attribute predicates that associate objects with their characteristic attributes. An LDT contains four attribute predicates - *associated_time*, *associated_start_time*, *associated_end_time* and *associated_size*, that associate an object with a time or size of particular granularity (e.g. years or semesters etc.). An example of an axiom that translates a temporal preposition follows.

$$\begin{aligned} during(E1, E2) \equiv & \\ & \exists T1, T2, Gran1, Gran2. \\ & (associated\_time(E1, T1, Gran1) \wedge \\ & associated\_time(E2, T2, Gran2) \wedge \\ & time\_during([T1, Gran1], [T2, Gran2])) \end{aligned} \tag{2.40}$$

Lexical Predicates

The last important category of predicates is lexical predicates. They correspond approximately to the content words in the input sentence.

The whole translation process then proceeds from lexical predicates to the mixture of attribute and conceptual predicates. This mixture is translated to pure conceptual predicates that are finally replaced by database predicates. The schema of the process is depicted in fig. 2.1

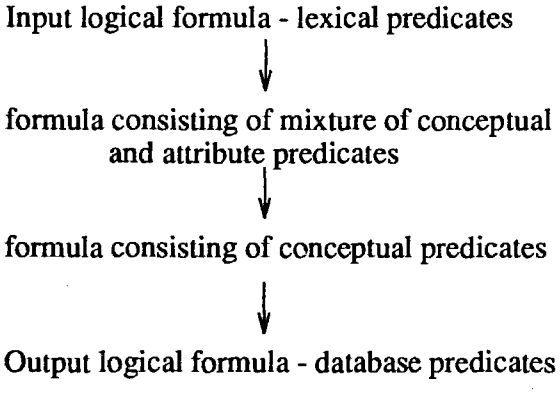

Figure 2.1: AET translation process - predicates at various stages of translation

2.4.3 Axioms

Axioms of the LDT can be divided into three main groups - general axioms, domain specific axioms, and database specific axioms.

General axioms take care of general words and phrases that usually occur in natural language interfacing (e.g. show me, tell me ...), temporal expressions (e.g. during ...) and expressions referring to the size of objects (e.g. larger, smaller ...). The axioms also express knowledge about translating interval predicates on different levels of granularity into expressions that involve predicates like "<" or ">", and expressions that express translation between different levels of granularity (e.g. from days into weeks). It is obvious that these axioms can be used in many different domains without any changes.

Domain specific axioms describe translation of domain dependent words and phrases into conceptual predicates. The axioms contain the core knowledge given a domain and they have to be rewritten when the domain changes. On the other hand, they are clearly separated from the structure of actual tables, so they can be applied to different database schemas under the condition that those schemas represent the same domain.

Database specific axioms describe the translation between conceptual predicates and the database predicates. These axioms are obviously the least general axioms and they have to be maintained with any change of the database schema.

2.5 Summary

In this chapter we provided a brief introduction to Rayner's Abductive Equivalential Translation upon which the rest of this thesis is based. The main concepts of AET are the AET process and the LDTs.

The AET process is able to translate a lexical logic formula whose predicates correspond to the words and phrases of an English sentence into a database logic formula understandable by a database. During the translation process, some assumptions can be taken (but do not have to be proven) to guarantee the equivalence between the lexical logic formula and the database logic formula. This allows for nonmonotonicity of the translation process which is considered a serious advantage.

The main source of information for the AET process is the LDT. The LDT is a logic theory that describes how the lexical logic formulas are related to the database logic formulas. Only certain kind of axioms are allowed in the LDTs. This restriction yields an

implementation with reasonable response time.

Chapter 3

Restricted Theories and Normalization

In this chapter we shall provide a general overview of the normalization of LDTs that can be considered as the main part of semantic preprocessing. Section 3.1 puts the normalization process into general context and also contains a small example of a normalized theory.

The normalization process, however, will not work with all LDTs, only a subset of them. If we want to normalize an LDT, we need to constrain its expressive power. We shall call such constrained LDTs *restricted* LDTs. Section 3.2 describes what restricted LDTs look like and also discusses the constraints and assumptions taken.

Section 3.3 defines the normalized LDT precisely. Section 3.4 contains detailed description of the formal algorithms and section 3.5 explains the main ideas behind the implementation of the system.

3.1 General Overview

3.1.1 General Method and Motivation

The semantic part of an NLID can be viewed as a binary relation. The relation is determined by a process the semantic processor is built on (e.g. AET) and semantic information the process uses (e.g. LDT). The relation is defined on a set of possible input formulas of some sort (e.g. lexical logic formulas) that represents a set of possible linguistic utterances presented at the input of the system and a set of output formulas (e.g. database logic formulas)

that represents a set of possible actions performed by the database access system. We can define the relation using a description of the condition that holds between its arguments - input and output formulas:

> The relation between an input formula and an output formula holds if and only if one of the interpretations of the input formula produced by the semantic part of the NLID is the output formula.

That means that only input formulas that can be understood by the underlying database access system after processing by the semantic part of the NLID can occur in the relation. By simple projection of the relation on its first argument we can obtain a unary relation that restricts its argument to certain values. The set of the values represents semantic restrictions on the set of possible input formulas, thus providing semantic restrictions on the input language.

By making the relation between input formulas and output formulas more explicit, the semantic constraints become more explicit as well. Because selectional restrictions can be viewed as a subset of semantic constraints on the input language, we believe that if the semantic constraints on the input language are made more explicit then these more explicit semantic constraints provide valuable information for derivation of the selectional restrictions. We also believe that by expressing the relationship between input and output formulas more explicitly, the translation process that translates input formula to output formulas becomes simpler and more efficient. In this thesis we shall demonstrate these two points on a system based on the AET translation algorithm.

3.1.2 Applying the Method - What does Normalization Look Like and What is it Good for

As was seen in the chapter 2, any LDT can be seen as a description of the relation between set of lexical logic formulas that mirror the actual natural language utterance and database logic formulas that represent the database side of the NLID.

Similarly a restricted LDT expresses the relationship between input logic formulas corresponding to the natural language utterances and output logic formulas that correspond to the database side of the NLID.

The main part of the preprocessing is the normalization of a restricted LDT. The purpose of the normalization is, as was suggested above, to make the relation between input logic

formulas and output logic formulas more explicit. The result of the normalization process is a *normalized* LDT.

A normalized LDT contains a set of quadruples called the *positive theory* and a set of axioms called the *negative theory.* Each quadruple of the positive theory describes a possible translation of an input atomic formula into an output atomic formula. The quadruple

$$(Condition, Assumptions, InputAF, OutputAF)$$

consists of an input atomic formula $InputAF$, output atomic formula $OutputAF$, conjunction of input atomic formulas $Condition$, and conjunction of assumptions $Assumptions$.

The negative theory is a set of axioms of a very restricted form that defines permissibility of assumptions.

Interpretation of the quadruples of the positive theory and the axioms of the negative theory is described as follows. For any input formula presented in the input of the system, an instance of the input atomic formula $InputAF$ can be translated into an instance of the output atomic formula $OutputAF$, if its local conjunctive context implies the condition $Condition$ w.r.t. the empty theory and does not contradict the assumptions $Assumptions$ w.r.t. the negative theory. (The term *empty theory* refers to a logic theory that contains only basic logic axioms and no "user defined" axioms. Proving implications of certain form in such theories can be done very efficiently.)

Due to the justification of conditions in the empty theory and justification of assumptions in the very constrained negative theory, the normalized LDT expresses the relation between input formulas and output formulas much more straightforwardly than the original LDT. The normalized theory simply eliminates the need for a reasoning process within an LDT. This elimination guarantees more declarative LDT because the designer of the system does not have to think "how" such reasoning can be done at runtime and how much time it will take. The normalized theory reflects only the results of a reasoning process.

The following example shows a very simple LDT and some quadruples that belong to the LDT in its normalized form. Although there is no need to do a lot of reasoning in the simple LDT to conclude the desired results, the example shows how such reasoning is eliminated in the normalized LDT.

Example 3.1 Suppose, that the LDT consists of the positive theory:

$$student(X) \equiv db\_student(X) \tag{3.1}$$

(word "student" refers to the student relation in the database)

$$professor(X) \equiv db\_professor(X) \tag{3.2}$$

(word "professor" refers to the professor relation in the database)

$$\begin{gathered} assumption\_person\_is\_a\_professor \rightarrow \\ (person(X) \equiv db\_professor(X)) \end{gathered} \tag{3.3}$$

(word "person" can refer to the professor relation in the database if it is safe to assume it)

$$\begin{gathered} assumption\_person\_is\_a\_student \rightarrow \\ (person(X) \equiv db\_student(X)) \end{gathered} \tag{3.4}$$

(word "person" can refer to the student relation in the database if it is safe to assume it)

$$course(X) \equiv db\_course(X) \tag{3.5}$$

(word "course" refers to the course relation in the database)

$$\begin{gathered} db\_student(X) \wedge db\_course(Y) \rightarrow \\ (take(E,X,Y) \equiv \\ db\_student\_takes\_course(E,X,Y)) \end{gathered} \tag{3.6}$$

$$\begin{gathered} db\_student\_takes\_course(E,X,Y) \rightarrow \\ take(E,X,Y) \end{gathered} \tag{3.7}$$

(word "take" refers to the relation describing student taking a course if it occurs in the right context)

$$\begin{gathered} db\_professor(X) \wedge db\_course(Y) \rightarrow \\ (teach(E,X,Y) \equiv \\ db\_professor\_teaches\_course(E,X,Y)) \end{gathered} \tag{3.8}$$

$$\begin{gathered} db\_professor\_teaches\_course(E,X,Y) \rightarrow \\ teach(E,X,Y) \end{gathered} \tag{3.9}$$

(word "teach" refers to the relation describing a professor teaching a course if it occurs in the right context) and the negative theory

$$\begin{gathered} assumption\_person\_is\_a\_student \rightarrow \\ \neg assumption\_person\_is\_a\_professor \end{gathered} \tag{3.10}$$

Here

$$student(X), professor(X), course(Y),$$
$$take(E,X,Y), teach(E,X,Y)$$

are input atomic formulas corresponding to words "student", "professor", "course", "take" and "teach";

$$db\_student(X), db\_professor(X), db\_course(Y),$$
$$db\_student\_takes\_course(E,X,Y),$$
$$db\_professor\_teaches\_course(E,X,Y)$$

are output predicates.

The following quadruples, and others, will belong to the normalized LDT. The quadruples are of the form

$$(Condition, Assumptions, InputAF, OutputAF)$$

$$(true, true, professor(X), db\_professor(X))$$

(word "professor" can always refer to $db\_professor$)

$$(professor(X) \wedge course(Y), true,$$
$$teach(E,X,Y), db\_professor\_teaches\_course(E,X,Y))$$

(word "teach" refers to the $db\_professor\_teaches\_course$, if it occurs in the context of words "professor" and "course")

$$(person(X) \wedge course(Y), assumption\_person\_is\_a\_professor,$$
$$teach(E,X,Y), db\_professor\_teaches\_course(E,X,Y))$$

(also words "person" and "course" can make the word "teach" refer to the

$$db\_professor\_teaches\_course$$

if the word "person" is not translated as $db\_student$ somewhere else in the sentence)

Let us compare the translation of the word "teach" in the original LDT and in the normalized LDT. In the original LDT, in order to translate the word "teach" in the context of words "professor" and "course", a reasoning process

has to be started. The reasoning process proves that the context consisting of words "professor" and "course" implies the condition

$$db\_professor(X) \wedge db\_course(Y)$$

Only then can the conditional equivalence for translation of the word "teach" be applied. On the other hand, in the normalized LDT, all that is needed is a simple lookup for a right quadruple. At this point we are not trying to argue efficiency of the processes. What we would like to demonstrate by this example is how straightforwardly a normalized LDT can express the relationship between natural language utterances on one hand and the database on the other. However, in order for this normalization process to work, it requires some restrictions on the theory.

■

3.2 Restricted LDT

In this section we shall provide and discuss basic assumptions and constraints we impose on the LDT. These assumptions guarantee that the LDT can be normalized in a sense suggested in the section 3.1. We shall call an LDT that satisfies all the criteria a *restricted LDT.*

Before discussing assumptions of the restricted LDT itself, we shall describe the input language and the output language of the "translator", whose knowledge is encoded in the restricted LDT.

3.2.1 Input Language

Assumption 3.1 (Input Language Syntax) *We suppose that the syntactic part of an NLID will produce an input logical formula, that represents a meaning of the natural language utterance. The input logical formula is a formula of extended First Order Logic (FOL). This means that formula can contain atomic formulas, conjunctions* $\wedge$, $\vee$, $\rightarrow$, $\neg$ *and quantifiers* $\exists$ *and* $\forall$. *In addition to the usual rules of formula construction of FOL, there are three higher order operators: count, sum and order. Their syntax and interpretation was defined in subsection 2.3.2.*

Some parsers (e.g. [2]) produce output consisting of the atomic formulas and constants that approximately correspond to the content words of the natural language or to the relationships between them. For example, the noun "book" can be expressed as an atomic formula $book(X)$. The atomic formula $book(X)$ can be read as "X denotes a book". The verb "give" can be represented by the formula $give(E, X, Y, Z)$ that can be read as "event E describes giving thing Y to person Z by the person X". This correspondence is fully determined by the parser, grammar and the lexicon.

For parsers that produce different representations, we need only assume that the resulting formula is a formula of the extended FOL and the semantic description "knows" what kind of input it can expect. There are no other requirements on the form of the output of the parser. One word can be represented as a complex logical formula or two or more words can be represented by one atomic formula. For example the verb "give" can be represented as a formula

$$giver(E, X) \wedge givee(E, Y) \wedge given(E, Z)$$

with the same interpretation as in the previous example or the phrase "John gives Mary a book" could be represented as an atomic formula

$$john\_gives\_mary\_a\_book(E)$$

that is true, if E denotes an event of John giving a book to Mary.

We do not expect that the meaning of the natural language utterance is resolved to the finest detail. It is supposed that the output of the parser can still contain certain ambiguities that can be represented by unresolved predicates. The semantic part of the interface is able to resolve such ambiguities (cf. chapter 6), which can include for example genitive or possessive relations or proper nouns representing unknown entities.

In the rest of the thesis we shall call formulas of the input language of the restricted LDT *input formulas.* Similarly we shall call atomic formulas of the input language *input atomic formulas* and we shall refer to predicates of the input language as *input predicates.*

3.2.2 Output Language

Assumption 3.2 (Output Language Syntax) *We suppose that the output language has the same syntax as the input language. All formulas of the output language are formulas of extended First Order Logic. This means that formulas can contain atomic formulas,*

conjunctions $\wedge$, $\vee$, $\rightarrow$, $\neg$ *and quantifiers* $\exists$ *and* $\forall$. *In addition to the usual rules of formula construction of FOL, there are three higher order operators: count, sum and order. Their syntax and interpretation was defined in subsection 2.3.2.*

Assumption 3.3 (Output Language Semantics) *The output logical formulas containing output predicates of the restricted LDT are not "true" database logical formulas that contain database predicates understandable by the database as they were described in the chapter 2. Instead, we suppose that the output of a process based on the restricted LDT is fed into another process based on the very simple* post-processing LDT *(PPLDT). The process based on PPLDT then produces the database logical formulas.*

Figure 3.1 compares the architecture of the original AET with the architecture of the system based on restricted LDTs with the terminology used for different parts of the system and different stages of translation.

We suppose that the PPLDT contains universal equivalences of the form

$$(\forall)(P \equiv A)$$

and existential equivalences of the form

$$(\forall)(\exists X_1 ... X_n P \equiv A)$$

where P is an output atomic formula and A is an arbitrary formula that contains the database predicates. The PPLDT also contains functional declarations that serve for simplification of the results. (c.f. chapter 2)

The fact that the equivalences do not contain any conditions that have to be proven guarantees that the translation from output logical formula into the formula that contains only database predicates can be performed very efficiently. The only thing the AET algorithm (cf. section 2.3.2) has to do is a simple substitution of output atomic formulas into database formulas according to the axioms of the PPLDT.

3.2.3 Assumptions about the Restricted LDT

In this section we shall present the assumptions made about the restricted LDT. We shall also discuss the assumptions from the point of view of translation power. We shall compare systems based on the restricted LDTs together with the PPLDT to the system based on the original LDT for AET as well.

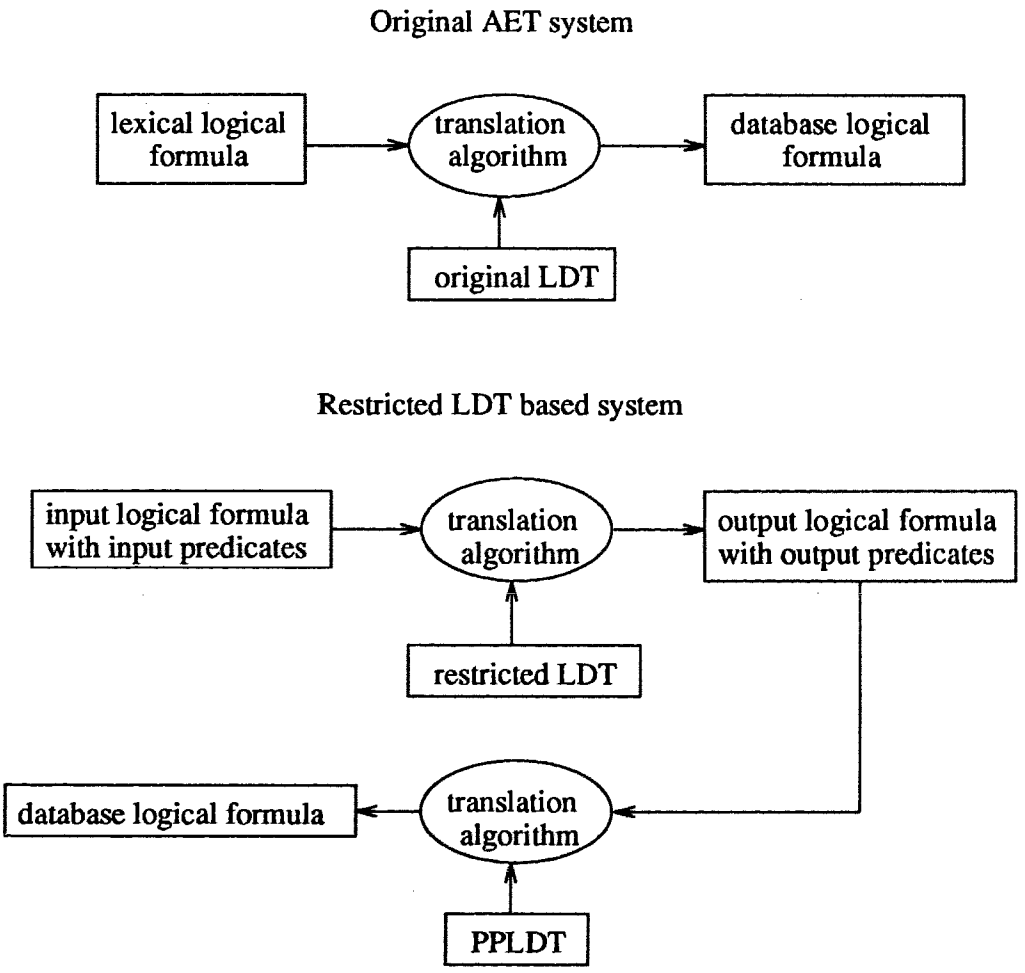

Figure 3.1: Translation process using AET with LDT vs. RLDT

Assumption 3.4 (RLDT Predicates) *A restricted LDT declares a set of input predicates, a set of output predicates and a set of assumption predicates. The sets of the predicates are mutually disjoint.*

(Note, that intermediate predicates that are neither assumptions nor output predicates nor input predicates are allowed.) We do not see any limitations that this assumption posits on the expressive power of the restricted LDT.

Assumption 3.5 (Finite Dictionary) *Any input formula that can be translated into the output formula can be also obtained by (one by one) substitution of the input atomic formulas with the output atomic formulas. Moreover, there is a finite dictionary that contains pairs of input and output atomic formulas provided by the designer of the system. The dictionary covers all possible cases of translation.*

Put informally, we simply assume that each atomic formula in the input formula has only a finite number of meanings w.r.t. to the database. We believe that the assumption is in most cases plausible. For example, there are probably no more than 20 meanings of the atomic formula representing a word "student" in the university database. There are however some cases where the assumption can not be fulfilled without an additional level of translation. This additional level of translation can be performed by an "intelligent" database engine that can provide some derived relations. An example of such a case is a database containing the relation *parent*. The word "ancestor" in this case can be translated in theoretically an infinite number of ways. An intelligent database engine can provide the additional relation *ancestor* derived from the relation *parent*. Examples of such an "intelligent" database engines are deductive databases [8] or Prolog.

Assumption 3.4 together with the assumption 3.5 about the output language is a superset of constraints implicitly made in the [23]. Original AET assumes that any lexical formula that can be translated into a database formula can be also obtained by (one-by-one) substitution of the lexical atomic formulas (in some cases together with the existential quantifiers associated with them) with the formulas (not necessary atomic) that contain only database predicates. One atomic formula can be translated in the infinite number of ways. Our assumption differs, because we require that there exists a finite dictionary which together with the output LDT covers all possible cases.

Assumption 3.6 (Formulas) *A restricted LDT contains two logic theories: one positive and one negative. The positive logic theory can contain only formulas of the following form:*

Conditional Equivalences:

$$(\forall)P_1 \cdots P_n \rightarrow (R_1 \cdots R_m \equiv Q_1 \cdots Q_l) \tag{3.11}$$

where $P_1 \cdots P_n$, $Q_1 \cdots Q_l$ and $R_1 \cdots R_m$ are atomic formulas, $Q_1 \cdots Q_l$ and $R_1 \cdots R_m$ cannot be assumptions

Horn Clauses:

$$(\forall)P_1 \cdots P_n \rightarrow Q \tag{3.12}$$

where $P_1 \cdots P_n$ and Q are atomic formulas, Q cannot be an assumption

The negative logic theory contains axioms of the form

Negative Horn Clauses:

$$(\forall)P_1 \cdots P_n \rightarrow \neg Q \tag{3.13}$$

where $P_1 \cdots P_n$ are atomic formulas whose predicates were not declared as assumptions and Q is an assumption atomic formula

Exclusive Declarations:

$$(\forall)(X_1 \cdots X_n \neq Y_1 \cdots Y_n) \rightarrow (P \rightarrow \neg Q) \tag{3.14}$$

where $\neq$ means not equal and P and Q are atomic formulas. Moreover, we restrict any other occurrences of predicates of atomic formulas P and Q to the condition part of conditional equivalences or Horn clauses of the positive theory.

The examples of conditional equivalences and Horn clauses were demonstrated in the chapter 2. Note, that a formula of the form (3.11) is equivalent to two clauses (3.15) and (3.16)

$$(\forall)P_1 \cdots P_n \wedge Q_1 \cdots Q_l \rightarrow R_1 \cdots R_m \tag{3.15}$$

$$(\forall)P_1 \cdots P_n \wedge R_1 \cdots R_m \rightarrow Q_1 \cdots Q_l \tag{3.16}$$

and these are equivalent to Horn clauses

$$(\forall)P_1 \cdots P_n \wedge Q_1 \cdots Q_l \rightarrow R_1$$
$$\vdots$$
$$(\forall)P_1 \cdots P_n \wedge Q_1 \cdots Q_l \rightarrow R_i$$
$$\vdots$$
$$(\forall)P_1 \cdots P_n \wedge Q_1 \cdots Q_l \rightarrow R_m$$

$$(\forall)P_1 \cdots P_n \wedge R_1 \cdots R_m \rightarrow Q_1$$
$$\vdots$$
$$(\forall)P_1 \cdots P_n \wedge R_1 \cdots R_m \rightarrow Q_i$$
$$\vdots$$
$$(\forall)P_1 \cdots P_n \wedge R_1 \cdots R_m \rightarrow Q_l$$

Therefore, it is also possible to simply assume that the positive logic theory consists only of the Horn clauses of type (3.12)

The reason why exclusive declarations of the form (3.14) are allowed is demonstrated by the following example.

Example 3.2 Suppose, that the concept *db* can be referred to by linguistic constructs $a \wedge b$ or $c \wedge d$. Unfortunately, AET does not allow equivalence with exclusive or on the LHS as in

$$(a \wedge b)xor(c \wedge d) \equiv db$$

The axioms (3.17) and (3.18)

$$a \wedge b \equiv db \tag{3.17}$$
$$c \wedge d \equiv db \tag{3.18}$$

do not provide expected results. One of the examples of the logical consequences of the axioms (3.17) and (3.18) that is probably not expected is a formula

$$a \wedge b \rightarrow d \tag{3.19}$$

Formula (3.19) would mean that anywhere in the theory where the concept d is required to be proven, $a \wedge b$ can be used. Imagine that a = "id", b =

"of professor", c = "teacher's", d = "number" and $db = db\_faculty\_id$. This would mean that anytime during the translation process where "number" is needed to be proven, "id of professor" can be used. So for example suppose that there is an axiom in the theory that translates the word "telephone" into the concept $db\_telephone\_number$ if word "telephone" occurs in the context of word "number" (e.g. in the phrase "telephone number"). The axiom would also allow one to translate the word "telephone" into $db\_telephone\_number$ if the context was "id of professor" (e.g. "telephone id of professor"). This is certainly not the intended result.

We adopted a solution where axioms (3.17) and (3.18) are changed into (3.20) and (3.21) with additional axiom (3.22) in the negative theory

$$Assume\_that\_db\_is\_refered\_to\_by(a\_and\_b) \rightarrow (a \wedge b \equiv db) \tag{3.20}$$

$$Assume\_that\_db\_is\_refered\_to\_by(c\_and\_d) \rightarrow (c \wedge d \equiv db) \tag{3.21}$$

$$Assume\_that\_db\_is\_reffered\_to\_by(a\_and\_b) \rightarrow \neg Assume\_that\_db\_is\_reffered\_to\_by(c\_and\_d) \tag{3.22}$$

Axiom (3.20) can be used if it is safe to assume the assumption

$$Assume\_that\_db\_is\_reffered\_to\_by(a\_and\_b)$$

Similarly axiom (3.21) can be used if it is safe to assume

$$Assume\_that\_db\_is\_reffered\_to\_by(c\_and\_d)$$

It is clear that both axioms cannot be used at the same time, because their assumptions are exclusive, thus preventing derivation of conclusion (3.19).

To allow for axioms of type (3.22) to be expressed without a lot of effort, especially if there is quite a number of possibilities for arguments of the assumption predicates, exclusive declarations are allowed in restricted LDT, e.g.

$$X \neq Y \rightarrow (Assume\_that\_db\_is\_reffered\_to\_by(X) \rightarrow \neg Assume\_that\_db\_is\_reffered\_to\_by(Y))$$

■

The reason for allowing negative Horn clauses is very simple. Often an additional word changes a meaning of the phrase. Consider the phrase "student" with input logical representation $student(X)$ and the phrase "student organization" with input logical representation

$$student(X) \wedge c(X,Y) \wedge organization(Y)$$

(here $c(X,Y)$ denotes the relation between two words which are part of the compound nominal). In the first case the word "student" can refer to a student recorded in the database, whereas in the second case, it can be just a part of the expression referring to some organization and does not introduce an identity of any student known to the database. To distinguish between such cases, the restricted LDT can contain a negative Horn clause

$$c(X,Y) \rightarrow \neg assumption\_word\_is\_not\_part\_of\_compound\_nominal(X)$$

in its negative theory. Then the assumption

$$assumption\_word\_is\_not\_part\_of\_compound\_nominal(X)$$

can be used in axioms for translation of the word "student".

In comparison with the original LDTs for AET, restricted LDTs lack existential equivalences and functional simplifications (c.f. section 2.4) that can be considered as a special kind of axiom as well. On the other hand, we allow the use of existential equivalences and functional simplifications in the PPLDTs.

Under assumption 3.5 that states that every input atomic formula can be translated into a finite number of the output atomic formulas and under assumption that the translation can be done in finite number of ways, the lack of existential equivalences does not decrease the translation power of the system based on a restricted LDT together with a PPLDT w.r.t. the expressive power of the original LDT for AET. The reason is very simple. Suppose, that $\exists X.(P(X,Y))$ can be translated into the formulas $A_1(Z) \cdots A_n(Z)$ under the conditions $C_1 \cdots C_n$ respectively. In this case X, Y and Z can represent the vectors of variables. We can create n output predicates, e.g. $P_1 \cdots P_n$ and n rules in the restricted LDT of the form:

$$C_i \rightarrow (P(X,Y) \equiv P_i(X,Z)) \text{ for } i = 1 \cdots n$$

and n rules in the output LDT of the form

$$\exists X P_i(X,Z) \equiv A_i(Z) \text{ for } i = 1 \cdots n$$

It is easy to see that using this construction we can achieve the same results as original LDT for AET.

The situation with functional declarations is a little bit different. We did not find a construction that will (even under the assumptions 3.4 and 3.5) create an equivalent theory. However, as Rayner's thesis suggests, the functional declarations are important for the last stage of translation where conceptual predicates are translated into database predicates. In our case, the last stage of translation is handled by a PPLDT, where the functional declarations are allowed.

Assumption 3.7 (Nonrecursivity) *The LDT is nonrecursive.*

The form of axioms declared by assumption 3.6 suggests that the preprocessing can be done using SLD-resolution. However, we restrict the theory even more to guarantee finiteness of refutation trees.

Informally, a theory is nonrecursive, if for any set of facts, any atomic formula, and any Prolog-like derivation of the atomic formula in the LDT enhanced by the set of facts, there is a Prolog like derivation of the same atomic formula in the LDT enhanced by the same facts, without a recursive call [1]

Example 3.3 Theory

$$ancestor(X,Y) \leftarrow parent(X,Y)$$
$$ancestor(X,Y) \leftarrow parent(Z,Y) \wedge ancestor(X,Z)$$

is a recursive theory, because if the theory is enhanced by the facts

$$parent(john, joe)$$
$$parent(joe, fred)$$

[1] Informally, a Prolog program is a set of rules of the form

$$a \leftarrow a_1 \wedge a_2 \cdots \wedge a_n.$$

or

$$b \leftarrow$$

Both of them are called *clauses*. The latter are also called facts. $a_1, a_2, \cdots a_n$ can be called conditions for derivation of a. Prolog can derive b any time, because there are no conditions associated with its derivation. In order to derive a, Prolog has to derive a_1 first, then derive a_2 and so on and at the end Prolog has to derive a_n. Derivation of a can also be viewed as a call to procedure a with statements $a_1 \cdots a_n$ that represent calls to procedures $a_1 \cdots a_n$ respectively. A derivation has a recursive call, if during the derivation Prolog has to "call" some procedure a that again "calls" the same procedure a.

then it is not possible to infer $ancestor(john, fred)$ without a recursive call. On the other hand a theory

$$\forall X.(student(X) \equiv db\_student(X))$$

$$\forall X, E, Y, S.(db\_course(Y, S) \wedge db\_student(X) \rightarrow (take(E, X, Y) \equiv db\_take\_a\_course(E, X, Y)))$$

$$\forall X, S.(assumption\_course(S) \rightarrow (unknown\_word(X, S) \equiv db\_course(X, S)))$$

$$\forall E, X, Y.(db\_take\_a\_course(E, X, Y) \rightarrow take(E, X, Y))$$

is nonrecursive. Even though recursive calls can be made within the theory e.g.

$$student(X) \leftarrow db\_student(X)$$
$$\leftarrow student(X) \leftarrow db\_student(X) \leftarrow student(X) \cdots$$

such recursive calls do not provide any new information. (e.g. from the fact $student(john)$, using the above path it is possible to derive only that $student(john)$ and $db\_student(john)$. Recursive calls derive information $student(john)$ and $db\_student(john)$ again and again, but such information can be derived without them). ∎

The nonrecursivity requirement places quite strong condition upon the restricted LDT. On the other hand, there is no need for recursivity in the above sense in the majority of the practical applications. Even in the case when recursivity is needed, it can be simulated up to a certain level by nonrecursive calls. Another possibility is to use more "intelligent" (in a sense described above) database engines that will take care of the recursive computation, when needed. The nonrecursivity constraint has also an advantage of a simple implementation within a reasoning procedure.

AET does not place any requirement on the nonrecursivity of the LDT. On the other hand, it uses certain heuristics to avoid infinitely recursive branches. The heuristics adds penalties to the derivation of the goals that are subsumed by ancestors. Because a certain cost limit is assumed, this method can quite effectively avoid infinite loops [23].

3.2.4 Definition of the Restricted LDT

In this section we shall provide a formal definition of the Restricted LDT. We shall use the terminology of [14]. For the reader's convenience, relevant terms are defined in appendix A. First we define the terms derivation tree and recursive call. Before we present a formal definition of the derivation tree, we show a simple example that demonstrates the concept.

Example 3.4 Consider theory

$$a \leftarrow b \wedge c$$
$$b \leftarrow d \wedge e$$
$$c \leftarrow f \wedge g$$
$$d \wedge e \wedge f \wedge g$$

and the goal $\leftarrow a$ and its SLD-refutation via R. The derivation tree for $\leftarrow a$ is shown in the figure 3.2

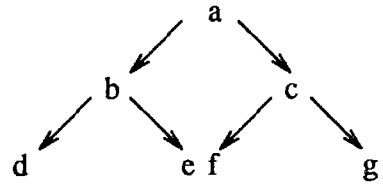

Figure 3.2: Derivation tree from example 3.4

■

Definition 3.1 (Derivation tree) *Let*

$$G_0, G_1, G_2, \cdots; C_1, C_2, \cdots; f_1, f_2, \cdots$$

be an SLD-derivation via R of G with goals $G_0, G_1, \cdots$, clauses $C_1, C_2, \cdots$ and substitutions $f_1, f_2, \cdots$. Then the derivation tree of the SLD-derivation via R of G is the graph (V, E) defined as follows.

Let us assign a distinct number to each atomic formula appearing in formulas $G_0, C_1, C_2, \cdots$. Then we assign a number to each atomic formula in $G_1, G_2, \cdots$ according to the following procedure.

Suppose that

$$G_i = \leftarrow A_1 A_2 \cdots A_n$$

and the numbers assigned to the atomic formulas $A_1 A_2 \cdots A_n$ *are* $x_1 x_2 \cdots x_n$, A_m *is a selected predicate (the predicate being resolved) and*

$$C_{i+1} = A \leftarrow B_1 \cdots B_k$$

(i.e. A_m *will be substituted by* $B_1 \cdots B_k$*) and the numbers assigned to the atomic formulas* $B_1 \cdots B_k$ *are* $y_1 \cdots y_k$*. Then*

$$G_{i+1} = \leftarrow (A_1 \cdots A_{m-1} B_1 \cdots B_k A_{m+1} \cdots A_n) f_{i+1}$$

And the numbers assigned to the atomic formulas in the goal

$$\leftarrow A_1 \cdots A_{m-1} B_1 \cdots B_k A_{m+1} \cdots A_n f_{i+1}$$

are

$$x_1 \cdots x_{m-1} y_1 \cdots y_k x_{m+1} \cdots x_n$$

The set of vertices of the derivation tree V *is the set of the heads of the program clauses* $C_1 f_1, \cdots$ *. The set of edges* E *of the derivation tree contains a pair* (P_j, P_i) *of atomic formulas* P_i *and* P_j*, if* P_i *is a head of the clause* C_i*,* P_j *is a head of the clause* C_j*, and the selected atomic formula in the goal* G_{i-1} *has the same number as one of the antecedents in formula* C_j*.*

Definition 3.2 (Recursive call) *An SLD-derivation via R has a recursive call if there is a path in the derivation tree of the SLD-derivation via R that contains two atomic formulas with the same predicate.*

The next example shows an SLD-derivation via R that has a recursive call.

Example 3.5 Consider the theory

$$student(X) \rightarrow db\_student(X)$$
$$db\_student(X) \rightarrow student(X)$$
$$student(john)$$

the definite goal G

$$\leftarrow student(john)$$

and the SLD-derivation in figure 3.3. The number in {}-brackets next to atomic formulas in the derivation is the number assigned by the definition of derivation tree.

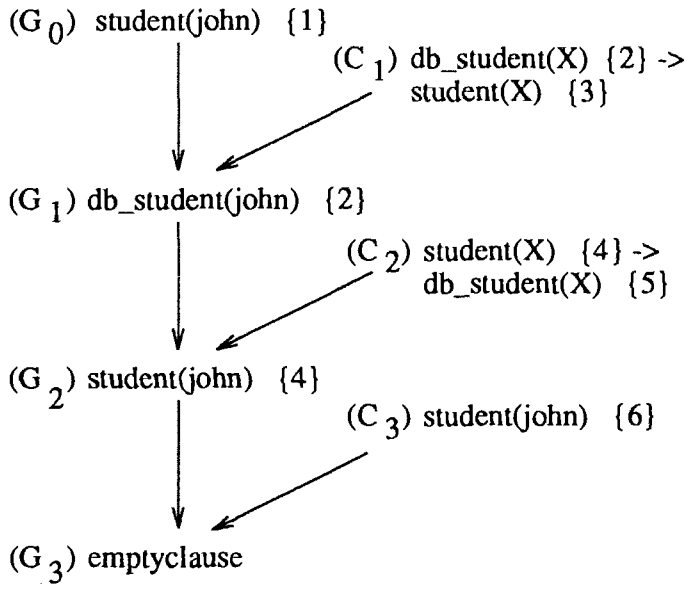

Figure 3.3: An example of SLD - derivation

The derivation tree for the SLD-derivation is depicted in figure 3.4. It is obvious that the derivation tree has a path that contains two atomic formulas with the same predicate (*student*) therefore the SLD-derivation has a recursive call.

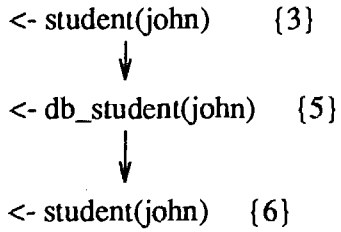

Figure 3.4: An example of derivation tree

With recursive call being defined, the following definition formalizes the concept of nonrecursive theory that was informally presented in assumption 3.7.

Definition 3.3 (Nonrecursive theory) *A finite set of definite program clauses T is a nonrecursive theory, if for each definite goal G, set of facts S and SLD-derivation of $T \cup S \cup \{G\}$ via R, there is an SLD-derivation of $T \cup S \cup \{G\}$ via R that does not have a recursive call.*

Now we are at the point where restricted LDT can be defined. A restricted LDT is a structure consisting of the positive and negative logic theories, sets of input, assumption and output predicates and the dictionary of the input and output formulas that satisfies assumptions (3.4) - (3.7).

Definition 3.4 (Restricted LDT) *The structure*

$$T = (Input, Output, Assumption, Pairs, PositiveTheory, NegativeTheory)$$

is called a restricted LDT, *iff*

- *Input, Output and Assumption are mutually disjoint sets of predicates*
- *Pairs is a set of pairs of ground atomic formulas of the form (I, O), where the predicate of the atomic formula I is a member of the set Input and the predicate of the atomic formula O is a member of the set Output*
- *PositiveTheory is a set of formulas of the form*

  $$(\forall) P_1 \cdots P_n \rightarrow Q$$

  where $P_1 \cdots P_n$ and Q are atomic formulas, the predicate of atomic formula Q cannot be a member of the set Assumption
- *The set PositiveTheory is nonrecursive.*
- *NegativeTheory is a set of formulas of the form*

  $$(\forall) P_1 \cdots P_n \rightarrow \neg Q$$

where $P_1 \cdots P_n$ are atomic formulas whose predicates are not members of the set Assumption and the predicate of atomic formula Q is a member of the set Assumption; or of the form

$$(\forall)(X_1 \cdots X_n \neq Y_1 \cdots Y_n) \rightarrow (P \rightarrow \neg Q)$$

where $\neq$ means not equal and P and Q are atomic formulas whose predicates are members of the set Assumption. Other occurrences of predicates of atomic formulas P and Q are restricted only to the condition part of Horn clauses of PositiveTheory

We shall call members of the set *Input input predicates of the theory T*, members of the set *Output output predicates of the theory T* and members of the set *Assumption assumption predicates of the theory T*. If the context assumes only one restricted LDT, then we omit the clause "of the theory T". Similarly atomic formulas that contain input predicates, output predicates or assumption predicates (of the theory T) are called *input atomic formulas, output atomic formulas or assumption atomic formulas (of the theory T)*. We shall also refer to any logical consequence of *PositiveTheory* as a logical consequence of LDT T.

Note, that the set of pairs was defined as a set of pairs of *ground* atomic formulas, yet we require the set to cover all possible cases of translation. Thanks to the generalization rule of first order logic that allows us to conclude $\forall X.A(X)$ from the formula $A(x)$ if constant x does not occur in the theory, this is not an issue. If the system derives any results that contain constants not occurring in *PositiveTheory* or *NegativeTheory*, such results can be generalized. As an example consider pair $(a(x,y), b(x,y))$ and

$$PositiveTheory = \{a(x,Y) \equiv c(z,Y), c(z,Y) \equiv b(x,Y)\}$$

We can derive equivalence $a(x,y) \equiv b(x,y)$ that can be easily generalized into $\forall Y.(a(x,Y) \equiv b(x,Y))$.

The requirement of ground pairs was adopted to make technical details of the normalization algorithms simpler.

Throughout the informal parts of thesis, we shall use the generalization rule often implicitly, when speaking about the results of the normalization process.

3.3 Normalized LDTs

Having defined restricted LDTs, it is time to say what to do with them. In this section, we present the term normalized LDT and also provide a motivation for the definition. As

was shown in the example at the end of the section 3.1, a normalized LDT is just a set of rules that will provide almost direct translation from input predicates at one end to the predicates closer to the database on the other.

The result of the normalization of the restricted LDT is in our case a set of quadruples (as was described in section 3.1) of the form

$$(P_1 \cdots P_n, A_1 \cdots A_m, I, O) \tag{3.23}$$

where $P_1 \cdots P_n$ and I are input atomic formulas, O is an output atomic formula predicate, $A_1 \cdots A_m$ are assumption atomic formulas and the pair (I, O) was specified in the dictionary. Formula $P_1 \cdots P_n$ corresponds to the positive pattern of the input language, the negation of the formula $A_1 \cdots A_m$ corresponds to the negative pattern of the input language, formula I corresponds to the input word, and formula O corresponds to the output word. The idea is that a quadruple (3.23) can be used in a similar way as the conditional equivalence of the form

$$A_1 \cdots A_m \wedge P_1 \cdots P_n \rightarrow (I \equiv O) \tag{3.24}$$

would be used in the AET process. That is, an input atomic formula I can be translated into an output atomic formula O, if the conjunctive context associated with the atomic formula I can prove input formula $P_1 \cdots P_n$ and does not contradict the assumptions $A_1 \cdots A_m$. The natural condition for quadruple (3.23) occurring in the result of the inflation process is, that (3.24) is a logical consequence of the restricted LDT. Therefore we impose condition 3.1 upon the quadruples that belong to the normalized LDT.

Condition 3.1 *If quadruple*

$$(P_1 \cdots P_n, A_1 \cdots A_m, I, O)$$

belongs to the normalized LDT, then

$$A_1 \cdots A_m \wedge P_1 \cdots P_n \rightarrow (I \equiv O)$$

is a logical consequence of the restricted LDT.

For the simplicity of explanation, now suppose, that there are no abductive assumptions in the theory. Let us consider the following example.

Example 3.6 Suppose that an LDT contains the following specifications:

Input predicates:	$\{i_1, i_2\}$
Output predicates:	$\{o\}$
Assumption predicates:	$\{\}$
Logic:	$i_1 \wedge i_2 \equiv o$
Dictionary:	(i_1, o)
	(i_2, o)

Then it is reasonable to demand that the normalized LDT would contain quadruples

$$(i_2, true, i_1, o)$$
$$(i_1, true, i_2, o)$$

If the only logic condition presented upon the resulting quadruples was condition 3.1, then the translation power of the normalized LDT itself would be very weak. There is simply no way to infer that formula $i_1 \wedge i_2$ of the input language is equivalent to the formula o using only information provided by the results of the normalized LDT. This is because formula $i_1 \wedge i_2 \equiv o$ is not a logical consequence of the theory that contains nonlogical axioms

$$i_2 \rightarrow (i_1 \equiv o)$$
$$i_1 \rightarrow (i_2 \equiv o)$$

However, if we add the axiom

$$o \rightarrow i_1 \tag{3.25}$$

we can obtain expected results. Similarly, axiom

$$o \rightarrow i_2 \tag{3.26}$$

will have the same effect. ∎

This example motivated us to formulate condition 3.1

Condition 3.2 *If quadruple*

$$(P_1 \cdots P_n, true, I, O)$$

belongs to the normalized LDT, then

$$(O \rightarrow I)$$

is a logical consequence of the restricted LDT.

Note, that this condition is possibly stronger than required. The condition 3.2 postulates that both axioms (3.25) and (3.26) have to be present, but either one of the axioms (3.25) or (3.26) is "able" to prove the desired results.

Condition 3.3 generalizes condition 3.2 in the case of the abductive assumptions present in the system.

Condition 3.3 *If quadruple*

$$(P_1 \cdots P_n, A_1 \cdots A_m, I, O)$$

belongs to the normalized LDT, then

$$A_1 \cdots A_m \rightarrow (O \rightarrow I)$$

is a logical consequence of the restricted LDT.

We consider conditions (3.1) and (3.3) as necessary conditions needed to provide the normalized LDT with approximately the same translation power as the original one. The following definition summarizes the above discussion by defining *Conditional Equivalence w.r.t. a restricted LDT named T*, which we shall abbreviate *CE(T)*.

Definition 3.5 Conditional Equivalence w.r.t. RLDT T (CE(T))
Let T be a restricted LDT

$$T = (Input, Output, Assumption, Pairs, PositiveLogic, NegativeLogic)$$

Then the quadruple

$$(P_1 \cdots P_n, A_1 \cdots A_m, I, O)$$

is a CE(T) if the following is true:

- *pair* (I, O) *belongs to the set* $Pairs$
- *predicates of atomic formulas* $P_1 \cdots P_n$ *are members of set* $Input$
- *predicates of atomic formulas* $A_1 \cdots A_m$ *are members of set* $Assumption$
- *formulas*

$$(\forall)A_1 \cdots A_m \wedge P_1 \cdots P_n \rightarrow (I \equiv O)$$
$$(\forall)A_1 \cdots A_m \rightarrow (O \rightarrow I)$$

are logical consequences of the PositiveLogic.

There is, however, one more issue we would like to address w.r.t. results of the normalization process. Consider theory

$$a(X) \rightarrow person(X)$$

(If it is safe to assume that if X represents a person's name, then we can consider X to really represent a person)

$$apple(X) \equiv db\_apple(X)$$

(Word apple always refers to the apple stored in a database)

In this case the following formulas are (undesirable) logical consequences of the LDT:

$$a(X) \wedge db\_apple(X) \rightarrow person(X)$$
$$apple(X) \wedge person(X) \rightarrow db\_apple(X)$$

therefore formula

$$a(X) \wedge apple(X) \rightarrow (person(X) \equiv db\_apple(X))$$

is also a logical consequence, so the quadruple

$$(apple(X), a(X), person(X), db\_apple(X))$$

should be present in the result of the normalization of the LDT. In this case, the problem occurs when the condition of the resulting conditional equivalence (e.g. $apple(X) \wedge a(X)$) together with the LDT can prove truth of both of the input and output words (e.g. formulas $person(X)$ and $db\_apple(X)$). That is, the condition $(apple(X) \wedge a(X))$ does not imply the equivalence in the sense we might expect - input and output words have the same meaning - but rather, that input and output are true. We consider such equivalence to be useless for the translation process, and define the term *nontrivial conditional equivalence*, abbreviated *NCE*, that takes this problem into consideration.

Definition 3.6 Nontrivial Conditional Equivalence w.r.t. RLDT T (NCE(T))
Let T be a restricted LDT

$$T = (Input, Output, Assumption, Pairs, PositiveLogic, NegativeLogic)$$

Then the quadruple

$$(P_1 \cdots P_n, A_1 \cdots A_m, I, O)$$

is a NCE(T) if the following is true:

- $(P_1 \cdots P_n, A_1 \cdots A_m, I, O)$ *is a CE(T)*
- *formulas*

$$P_1 \cdots P_n A_1 \cdots A_m \rightarrow I$$
$$P_1 \cdots P_n A_1 \cdots A_m \rightarrow O$$

  are not logical consequences of the PositiveLogic

Now we are at the point where we can define terms *normalized LDT* and *generalized normalized LDT* w.r.t. restricted LDT T. The term generalized normalized LDT formalizes the idea of using pairs of ground atomic formulas to produce more general rules where some of the constants from the pairs are substituted by unique variables. (cf. end of section 3.2)

Definition 3.7 Normalized LDT w.r.t. to RLDT T (NLDT(T))
Let T be a restricted LDT. Then NLDT(T) is a set of CE(T)s such that for every

$$NCE(T) = (PP, AA, I, O)$$

there is a

$$CE(T) = (PP', AA', I, O)$$

from NLDT(T) and a substitution g such that

$$PP \wedge AA \rightarrow PP'g \wedge AA'g$$

Definition 3.8 Generalized Normalized LDT w.r.t. to RLDT T (GNLDT(T))
Let T be a restricted LDT

$$T = (Input, Output, Assumption, Pairs, PositiveTheory, NegativeTheory)$$

T' be an NLDT(T). Let T'' be a set of all quadruples occurring in T', where all occurrences of constants in Pairs that occur neither in PositiveTheory nor in NegativeTheory are substituted by variables. The substitution maps the occurrence of the same constants to the same variables and occurrences of different constants to different variables. Then T'' is a generalized normalized LDT w.r.t. RLDT T.

In the rest of the thesis, if the context assumes only one RLDT T, we omit the clause "w.r.t. RLDT T" from above defined terms and write simply conditional equivalence (CE), nontrivial conditional equivalence (NCE), normalized linguistic domain theory (NLDT) and generalized normalized LDT, (GNLDT).

3.4 Normalization Algorithms

3.4.1 Overview of the Algorithms

There are two algorithms that take care of the normalization process: *the conditions construction algorithm* and *the conditions combination algorithm.* The condition construction algorithm, given a certain type of logic theory T, goal Q and a set of predicates Pr, finds all possible conditions that consist of atomic formulas with predicates from Pr s.t. the conditions imply the goal Q. The conditions combination algorithm uses the conditions construction algorithm to construct normalized LDTs. In this subsection, we shall describe the motivation for their behavior and a simple example.

All quadruples from a normalized LDT have associated formulas of the form

$$I_1 \cdots I_n \wedge A_1 \cdots A_m \rightarrow (I \equiv O) \tag{3.27}$$

$$A_1 \cdots A_m \rightarrow (O \rightarrow I) \tag{3.28}$$

where $I, I_1 \cdots I_n$ are input atomic formulas, $A_1 \cdots A_m$ are assumption atomic formulas and O is an output atomic formula. The formulas (3.27) and (3.28) are logical consequences of the *PositiveLogic.* It is easy to see that a theory containing formulas (3.27) and (3.28) is equivalent to the same theory where the formulas are substituted by the formulas (3.29) and (3.30).

$$I_1 \cdots I_n \wedge A_1 \cdots A_m \wedge I \rightarrow O \tag{3.29}$$

$$A_1 \cdots A_m \wedge O \rightarrow I \tag{3.30}$$

This means that for each CE there are two Horn clauses. The first one has an output atomic formula as its head and has a body consisting of input and assumption atomic formulas, and the second one has an input atomic formula as its head and has a body consisting of assumption atomic formulas and an output atomic formula. This simple fact is used to produce results that satisfy the definition of CE from the previous section.

Given a pair (I, O), the conditions construction algorithm is run twice to find two sets SO and SI. Set SO contains axioms of the form (3.31):

$$R_1 \cdots R_m \rightarrow O \tag{3.31}$$

where $R_1 \cdots R_m$ are input or assumption atomic formulas, and set SI similarly contains axioms of the form (3.32),

$$Q_1 \cdots Q_l \rightarrow I \tag{3.32}$$

where $Q_1 \cdots Q_l$ are assumption atomic formulas or output atomic formulas with the same predicate as O. Note, that the formulas (3.31) and (3.32) are similar to the formulas (3.29) and (3.30). Naturally, we require all axioms in the sets to be logical consequences of the *PositiveLogic*. The sets are finite and they also *cover* all possible cases that can be derived in the LDT. The latter requirement does not mean that the sets *contain* all possible cases. All we need is that if there is a formula of the form (3.31), that is a logical consequence of the theory, then there is an axiom in the set SO whose condition is equal or weaker. The same is true about the axioms of the form (3.32) and the set SI.

The condition combination algorithm then combines sets SI and SO to obtain the CEs of a normalized LDT.

The combination process is again based on the equivalence of formulas (3.27), (3.28) and (3.29), (3.30).

Example 3.7 As an example consider the restricted LDT

$$(Input, Output, Assumption, Pairs, PositiveLogic, NegativeLogic)$$

where

$$
\begin{aligned}
Input &= \{i_1, i_2, i_3\} \\
Output &= \{o_1, o_2, o_3\} \\
Assumption &= \{a\} \\
Pairs &= \{(i_1, o_1), (i_2, o_2), (i_3, o_3)\} \\
PositiveLogic &= i_1 \equiv o_1 \\
&\quad a \rightarrow (i_2 \equiv o_2) \\
&\quad o_1 \rightarrow m_1 \\
&\quad o_2 \rightarrow m_2 \\
&\quad m_1 \wedge m_2 \rightarrow (i_3 \equiv o_3) \\
&\quad a \wedge m_1 \rightarrow (i_3 \equiv o_3) \\
&\quad o_3 \rightarrow i_3 \\
NegativeLogic &= \{\}
\end{aligned}
$$

Using the example, the sets produced by the conditions construction algorithm for the pair (i_1, o_1) are

$$
\begin{aligned}
SO_1 &= \{i_1 \rightarrow o_1\} \\
SI_1 &= \{o_1 \rightarrow i_1\}
\end{aligned}
$$

for the pair (i_2, o_2) are

$$
\begin{aligned}
SO_2 &= \{a \wedge i_2 \rightarrow o_2\} \\
SI_2 &= \{a \wedge o_2 \rightarrow i_2\}
\end{aligned}
$$

and for the pair (i_3, o_3) are

$$
\begin{aligned}
SO_3 &= \{a \wedge i_1 \wedge i_2 \wedge i_3 \rightarrow o_3, \\
&\quad\ a \wedge i_1 \wedge i_3 \rightarrow o_3\} \\
SI_3 &= \{o_3 \rightarrow i_3\}
\end{aligned}
$$

Notice how these sets contain only the input, output and assumption predicates, and none of the intermediate predicates m_1, or m_2. Then for each pair (I, O) from the set of pairs, corresponding sets SI and SO are combined using the conditions combination algorithm to produce CEs

$$(true, true, I_1, o1)$$
$$(true, a, i_2, o_2)$$
$$(i_1 \wedge i_2, a, i_3, o_3)$$
$$(i_1, a, i_3, o_3)$$

3.4.2 Condition Construction Algorithm

The conditions construction algorithm that constructs sets SI and SO is very similar to the Prolog meta-interpreter [18], [26]. The input to the algorithm is a nonrecursive theory T, a finite set of predicates Pr and a ground atomic formula Q. Given a pair

$$(InputAtomicFormula, OutputAtomicFormula)$$

from the dictionary of a restricted LDT, the algorithm is called with

$$\begin{aligned} T &= PositiveTheory \\ Pr &= Assumption \cup Input \\ Q &= OutputAtomicFormula \end{aligned}$$

to construct the set SO, and with

$$\begin{aligned} T &= PositiveTheory \\ Pr &= Assumption \cup \{\text{Predicate of } OutputAtomicFormula\} \\ Q &= InputAtomicFormula \end{aligned}$$

to construct the set SI. The algorithm tries to prove the formula Q using the theory T. Whenever the algorithm needs to prove an atomic formula with the predicate from the set of predicates Pr, the algorithm either puts the formula aside and considers it to be proven or tries to prove it using the theory T. If the proof of the formula Q is done, the Horn clause, whose condition consists of all atomic formulas that were put aside and whose head is equal to Q, is placed into the output set S. The algorithm then backtracks until there are no other possibilities for proving the atomic formula Q. The algorithm also makes sure that there are no recursive calls within the *PositiveTheory* of the restricted LDT during the derivation of the conditions.

A Prolog version of the algorithm will be presented in section 3.5.

Algorithm 3.1 *Conditions Construction Algorithm*

Input: *nonrecursive theory T, finite set of predicates Pr, ground atomic formula Q*

Output: *Set S of formulas F of the form:*

$$F = P_1 \cdots P_n \rightarrow Q$$

where $P_1 \cdots P_n$ are atomic formulas and predicates of $P_1 \cdots P_n$ belong to Pr, s.t.

$$T \models F \text{ (soundness condition)}$$

if

$$T \models P'_1 \cdots P'_m \rightarrow Q$$

then there is a formula F from S and a substitution f s.t.

$$(\forall) P'_1 \cdots P'_m \rightarrow (P_1 \cdots P_n f) \text{(completeness condition)}$$

Algorithm:

1. *Create set T'' of atomic formulas R the following way.* **For** *each predicate P from the set of predicates Pr construct an atomic formula R, whose predicate (and arity) correspond to P and whose arguments are unique variables.*
   Construct program $T' = T \cup T''$
2. *Construct a partial SLD-tree via R for $\rightarrow Q$ in T' which is equivalent to the whole SLD-tree via R for $\rightarrow Q$ in T', but all branches leading to SLD-derivations with recursive calls are cut at the point where the recursive call occurs.*
3. **For** *each success branch in the part of the SLD-tree which corresponds to the derivation*

   $$\rightarrow Q, G_1 \cdots G_n; C_1 \cdots C_n; f_1 \cdots f_n$$

   insert the rule

   $$P1 \cdots Pm \rightarrow Q$$

   into the result set S, where the sequence $P_1 \cdots P_m$ corresponds to those elements of $\{C_1 f_1 \cdots f_n \cdots C_n f_1 \cdots f_n\}$ whose C_i does not occur in T.

A proof of correctness for the algorithm can be found in appendix A.
The following example shows a run of the algorithm step by step.

Example 3.8 Suppose that

$$\begin{aligned} T &= \{i_1(X) \rightarrow o(X), \\ &\quad i_2(X) \rightarrow o(X), \\ &\quad o(X) \rightarrow i_1(X), \\ &\quad m \rightarrow o(X)\} \\ Pr &= \{i_1, i_2\}, \\ Q &= o(a) \end{aligned}$$

Step 1. The theory T' is assigned the following atomic formulas:

$$T' = T \cup \{i_1(Y), i_2(Z)\}$$

(We suppose that i_1 and i_2 are predicates of arity 1.)

Step 2. All possible branches of the SLD-tree are depicted in figure 3.5

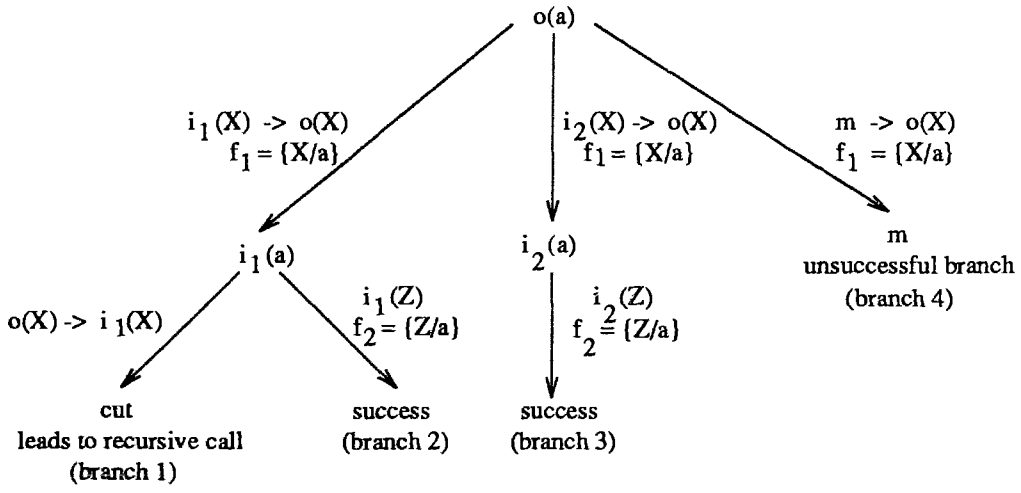

Figure 3.5: SLD - tree for example 3.8

Step 3. In this step the algorithm inspects branches 2 and 3. Branch 2 corresponds to the derivation where $C_1 = i_1(X) \rightarrow o(X)$ and $C_2 = i_1(X)$. The algorithm picks some of the members of the set $\{C_1 f_1 f_2, C_2 f_1 f_2\}$. C_1

occurs in the original theory T, so only member $C_2 f_1 f_2 = i_1(a)$ of the set is considered and formula $i_1(a) \rightarrow o(a)$ is inserted into the resulting set S. Similarly after inspection of branch 3, $i_2(a) \rightarrow o(a)$ is inserted into set S.

■

3.4.3 Conditions Combination Algorithm

After the sets SO and SI are constructed using the conditions construction algorithm for each pair (I, O) with input and output assignments

$$\begin{aligned} T &= PositiveTheory \\ Pr &= Assumption \cup Input \\ Q &= O \\ S &= SO \end{aligned}$$

and

$$\begin{aligned} T &= PositiveTheory \\ Pr &= Assumption \cup \{\text{Predicate of } O\} \\ Q &= IF \\ S &= SI \end{aligned}$$

results SI and SO can be combined in the following way. Suppose that the formulas of the forms (3.33) and (3.34) are members of the sets SO and SI respectively or that such formulas can be constructed out of two members of the sets SO and SI by applying a substitution. (Note, that although set SO can contain Horn clauses with the head equal to O, the body of such Horn clauses can contain formulas that are *subsumed* by the atomic formula I).

$$R'_1 \cdots R'_m \wedge I \rightarrow O \tag{3.33}$$

$$Q'_1 \cdots Q'_l \wedge O \rightarrow I \tag{3.34}$$

Suppose, that $R'_1 \cdots R'_i$ are input atomic formulas and $R'_{i+1} \cdots R'_m$ are assumption atomic formulas. The formulas (3.33) and (3.34) are logical consequences of the *PositiveTheory* of the LDT. Then the formulas (3.35) and (3.36) are also logical consequences of the same LDT.

$$R'_1 \cdots Rm' \wedge Q'_1 \cdots Q'_l \rightarrow (I \equiv O) \tag{3.35}$$

$$Q'_1 \cdots Q'_l \rightarrow (O \rightarrow I) \tag{3.36}$$

If $Q'_1 \cdots Q'_l$ happen to be assumption atomic formulas, the quadruple

$$(R'_1 \cdots R'_i, R'_{i+1} \cdots R'_m \wedge Q'_1 \cdots Q'_l, I, O)$$

is a CE and becomes a part of the normalized LDT.

Algorithm 3.2 *Conditions Combination Algorithm*

Input: *A restricted LDT*

$$(PositiveLogic, NegativeLogic, Input, Output, Assumption, Pairs)$$

Output: *Set S of CEs s.t. if there is a NCE*

$$(PP', AA', I, O)$$

then there is a CE

$$(PP, AA, I, O)$$

from the set S and a substitution f s.t.

$$PP' \wedge AA' \rightarrow PPf \wedge AAf$$

Algorithm:

1. **For** *each I, s.t. there is a pair* (I, O) *that belongs to Pairs run the conditions construction algorithm, where*

$$\begin{aligned} T &= PositiveTheory \\ Pr &= Assumption \cup \{Predicate\ of\ OutputAtomicFormula\} \\ Q &= InputAtomicFormula \\ S &= SI \end{aligned}$$

Construct the set SI that is the union of the results of the conditions construction algorithm.

2. **For** *each* O *s.t. there is a pair* (I, O) *that belongs to* $Pairs$*, run the conditions construction algorithm, where*

$$\begin{aligned} T &= PositiveLogic \\ Pr &= Assumption \cup Input \\ Q &= OutputAtomicFormula \\ S &= SO \end{aligned}$$

Construct the set SO *that is the union of the results of the conditions construction algorithm.*

3. **For** *each pair (I, O) from Pairs* **do**
   for *each formula* F *of the form* $R_1 \cdots R_m \rightarrow O$ *from* SO **do**
   for *each formula F1 of the form* $Q_1 \cdots Q_l \rightarrow I$ *from* SI*, where predicates of* $Q_1 \cdots Q_l$ *are assumption predicates or are equal to predicate of atomic formula* O **do**

4. **for** *all nonempty subsets* SR *of the set of formulas* $\{R_1 \cdots R_m\}$*, s.t. there is a m.g.u.* r *of members of* SR *and* I **do**
   for *all subsets* SQ *of the set of formulas* $\{Q_1 \cdots Q_l\}$ *that contain all atomic formulas with predicate of* O*, s.t. there is a m.g.u.* q *of the members of* SQ *and* O **do**
   if *the variables from formulas* $R_1 \cdots R_m$ *are not different from variables* $Q_1 \cdots Q_l$ **then** *apply the renaming substitution.*
   Construct sets

$$RR' = \{R_1 \cdots R_m\} - SR$$
$$QQ' = \{Q_1 \cdots Q_l\} - SQ$$

Construct set

$$CC = RR'rq \cup QQ'rq$$

Construct a $CE\,(PP, AA, I, O)$*, where* PP *are all atomic formulas with input predicates from* CC *and*

AA are all atomic formulas with assumption predicates from CC.

A proof of correctness for the algorithm can be found in appendix A.
The following example shows the operation of the algorithm on a very simple LDT.

Example 3.9 Suppose that

$$\begin{aligned} PositiveLogic &= \{i_1(X) \wedge i(X,Z) \wedge i_2(Y) \wedge i(Y,Z) \rightarrow o(Z) \\ &\quad a(Z) \wedge o(Z) \rightarrow i(W,Z)\} \\ NegativeLogic &= \{\} \\ Input &= \{i, i_1, i2\} \\ Output &= \{o\} \\ Assumption &= \{a\} \\ Pairs &= \{(i(x,z), o(z))\} \end{aligned}$$

Steps 1 and 2 Sets SO contains axiom (3.37)

$$i_1(X) \wedge i(X,z) \wedge i_2(Y) \wedge i(Y,z) \rightarrow o(z) \tag{3.37}$$

and set SI contains axiom (3.38)

$$a(z) \wedge o(z) \rightarrow i(x,z) \tag{3.38}$$

Steps 3 and 4 Set SR and substitution r iterates through values

$$\begin{aligned} SR &= \{i(X,z)\} && \text{for } r = \{X/x\} \\ SR &= \{i(Y,z)\} && \text{for } r = \{Y/x\} \\ SR &= \{i(X,z), i(Y,z)\} && \text{for } r = \{X/x, Y/x\} \end{aligned}$$

from axiom (3.37) whereas set SQ and substitution q can have only one value:

$$SQ = \{o(z)\} \text{ for } q = \{\}$$

So the normalized LDT will contain these CEs

$$(i_1(x) \wedge i_2(Y) \wedge i(Y,z), a(z), i(x,z), o(z))$$
$$(i_1(X) \wedge i(X,Z) \wedge i_2(x), a(z), i(x,z), o(z))$$
$$(i_1(x) \wedge i_2(x), a(z), i(x,z), o(z))$$

■

3.5 Implementation of Normalization

3.5.1 Complex and Simple Theories

The design of the whole system allows the designer to structure the knowledge of the semantic part of the interface into simple theories that can be combined into complex ones. This design allows one to posit axioms, that are visible within the simple theory, but cannot be reached from other simple theories or from the complex theories. This is similar to the modularization and information hiding in computer programs which has clear advantages from the software engineering point of view. The idea of dividing knowledge into smaller theories was used also in [13].

Another advantage of the design is demonstrated by the following example.

Example 3.10 Suppose, that we would like to talk about persons, students and student numbers. Then we can simply posit an axiom:

$$number(X, Y) \equiv id(X, Y) \tag{3.39}$$

that allows us to infer that the expression "John's student number" refers to the same thing as the expression "John's student id". Similarly "person's number" is the same as "person's id". But this is not true in general. E.g. "John's phone number" is not the same as "John's phone id". The architecture makes it possible to express axioms similar to (3.39) without need to describe complicated conditions under which the axiom can be used. ∎

We propose an implementation in which the restricted LDTs contain not only the parts described in the formal definition as

$$(Input, Output, Assumption, Pairs, PositiveTheory, NegativeTheory)$$

but also a list of names of theories whose knowledge is inherited, so the theories can be arranged as an inheritance hierarchy. Each theory can inherit some information from arbitrary number of other theories and the information (or part of information) from each theory can be inherited by an arbitrary number of other theories. The only condition imposed on the inheritance hierarchy is acyclicity.

The following example shows two simple RLDTs T_1 and T_2, and a complex RLDT T.

Example 3.11

$$
\begin{array}{lll}
T_1: & Input & = \{a_1\} \\
& Output & = \{b_1\} \\
& Assumption & = \{\} \\
& Pairs & = \{(a_1, b_1)\} \\
& PositiveLogic & = \{a_1 \equiv b_1\} \\
& NegativeLogic & = \{\}
\end{array}
$$

$$
\begin{array}{lll}
T_2: & Input & = \{a_2\} \\
& Output & = \{b_2\} \\
& Assumption & = \{\} \\
& Pairs & = \{(a_2, b_2)\} \\
& PositiveLogic & = \{a_2 \equiv b_2\} \\
& NegativeLogic & = \{\}
\end{array}
$$

$$
\begin{array}{lll}
T: & Inherits & = \{T_1, T_2\} \\
& Input & = \{a_1, a_2\} \\
& Output & = \{b\} \\
& Assumption & = \{\} \\
& Pairs & = \{((a_1, b), (a_2, b)\} \\
& PositiveLogic & = \{b_1 \equiv b, b_2 \equiv b\} \\
& NegativeLogic & = \{\}
\end{array}
$$

■

The algorithms that take care of the complex theories are the same as the algorithms described in section 3.4. The only difference is that during the normalization of the complex theory, the partial results from normalization of inherited theories are used. So in example 3.11 theory T inherits partial results from theories T_1 and T_2. The partial results we have in mind here are the sets SO and SI constructed by the steps 1 and 2 of the conditions construction algorithm. Recall that the set SO constructed during the step 1 of the conditions combination algorithm contains all possible formulas of the form

$$R_1 \cdots R_m \rightarrow O$$

where O is an output atomic formula mentioned in the set *Pairs* and $R_1 \cdots R_m$ are assumption or input atomic formulas. Similarly set SI contains formulas of the form

$$Q_1 \cdots Q_l \rightarrow I$$

where $Q_1 \cdots Q_l$ are either assumption atomic formulas or atomic formulas with the same predicate as output atomic formula O for all pairs (I, O) in the *Pairs*.

Example 3.12 As an example consider theories T_1, T_2 and T from example 3.11. During normalization of theories T_1 and T_2, the conditions combination algorithm produced intermediate results

$$\begin{aligned} SI_1' &= \{b_1 \rightarrow a_1\} \\ SO_1' &= \{a_1 \rightarrow b_1\} \end{aligned}$$

$$\begin{aligned} SI_2' &= \{b_2 \rightarrow a_2\} \\ SO_2' &= \{a_2 \rightarrow b_2\} \end{aligned}$$

During the normalization of the theory T, axioms in sets SI_1', SO_1', SI_2' and SO_2' can be used together with the axioms of *PositiveTheory* of RLDT T. ∎

The *NegativeTheory* together with possible assumption declarations and functional declarations of the RLDT are inherited without any preprocessing.

Although the conditions construction algorithm and the conditions combination algorithm obviously preserve the soundness conditions while working with the inheritance hierarchies of theories, the completeness condition cannot be generalized easily.

3.5.2 Implementation of Normalization Algorithms

The implementation contains three logical parts that correspond to the simple preprocessing of conditional equivalences and the two algorithms described in the section 3.4.

Preprocessing of Conditional Equivalences

The formal definitions of the restricted LDTs and normalization algorithms assume, for the purposes of simpler explanation, positive theories containing strictly Horn clauses. The

implementation, on the other hand, allows the definition of axioms of the PositiveTheory in the form

$$P_1 \cdots P_n \rightarrow (R_1 \cdots R_m \equiv Q_1 \cdots Q_l) \tag{3.40}$$

The preprocessing utilizes an equivalence mentioned in section 3.2 between conditional equivalences like (3.40) and the following Horn clauses

$$P_1 \cdots P_n \wedge R_1 \cdots R_m \rightarrow Q_1$$
$$\cdots$$
$$P_1 \cdots P_n \wedge R_1 \cdots R_m \rightarrow Q_l$$

$$P_1 \cdots P_n \wedge Q_1 \cdots Q_l \rightarrow R_1$$
$$\cdots$$
$$P_1 \cdots P_n \wedge Q_1 \cdots Q_l \rightarrow R_m$$

Example 3.13

The set of axioms

$$\{$$
$$a \rightarrow (b \equiv c)$$
$$d \equiv e$$
$$f \rightarrow g$$
$$\}$$

is preprocessed into the set

$$\{$$
$$a \wedge b \rightarrow c$$
$$a \wedge c \rightarrow b$$
$$d \rightarrow e$$
$$e \rightarrow d$$
$$f \rightarrow g$$
$$\}$$

■

3.5.3 Implementation of the Conditions Construction Algorithm

Given a set of Horn clauses T, set of input predicates Pr and a ground atomic formula Q, the conditions construction algorithm finds patterns of every possible condition made from atomic formulas with predicates from the set Pr, such that the ground atomic formula holds in the theory T. We proved that this can be done by searching all successful branches of an SLD-derivation tree with no recursive calls as described in section 3.4. To explain the implementation, we first present the algorithm that searches the SLD-derivation tree given a theory T and ground atomic formula Q. The next step will be an algorithm that cuts recursive branches. Then we extend the algorithm to search the SLD-derivation tree given the theory that contains all axioms from T and all variants of input predicates from Pr. At the end of this section we show a schema of the final implementation of the algorithm that also collects the conditions.

Because the main part of the algorithms used during the normalization process is based on unification, we have chosen Prolog to be a programming language of implementation. We assume that reader is familiar with the basic concepts of this programming language. (A good introduction can be found in [26]).

For demonstration purposes, we assume that for each Horn clause from the theory T of the form

$$Body \rightarrow Head$$

there is an asserted clause:

```
implicationInTheory(Head, Body).
```

if body is nonempty and asserted clause:

```
atomicInTheory(Head).
```

If the body is empty.

We also assume that for each element J of the set Pr of input predicates, there is asserted clause

```
inputPredicate(J).
```

The search for successful branches of derivation tree is based on the Prolog meta-interpreter as presented in ([26], [18]).

```
prove(L and R) :-
        prove(L),
        prove(R).

prove(F) :-
        atomicFormula(F),
        proveAtomicFormula(F).


proveAtomicFormula(F) :-
        implicationInTheory(F, Body),
        prove(Body).

proveAtomicFormula(F) :-
        atomicInTheory(F).
```

Predicate `prove(O)` succeeds, whenever a successful branch in the SLD-derivation tree of `O` is found. Now we add an argument `Nodes` that keeps track of visited predicates and prevents the current branch from being expanded if a recursive call occurs. The collection `Nodes` was implemented as a list although many other implementations are possible. Predicate `predicate(F, P)` succeeds iff `P` is a predicate of atomic formula `F`.

```
prove(F) :- prove(F, [])

prove(L and R, Nodes) :-
        prove(L, Nodes),
        prove(R, Nodes).

prove(F, Nodes) :-
        atomicFormula(F),
        proveAtomicFormula(F, Nodes).


proveAtomicFormula(F, Nodes) :-
```

```
        % check nonrecursivity
      predicate(F, P),
      non_member(P, Nodes),
        % call the prover
      proveAtomicFormula_1(F, Nodes, P).

proveAtomicFormula_1(F, Nodes, P) :-
      implicationInTheory(F, Body),
      prove(Body, [P|Nodes]).

proveAtomicFormula_1(F, _Nodes, _P) :-
      atomicInTheory(F).
```

Then the program was altered to search the SLD-derivation tree of `O` using theory `T` enhanced by all possible variants of predicates in `I`. This was achieved by adding another clause to `proveAtomicFormula1(F, Nodes, P)` that allows the proof of an atomic formula whose predicate is a member of `I` for "free".

```
prove(F) :- prove(F, [])

prove(L and R, Nodes) :-
      prove(L, Nodes),
      prove(R, Nodes).

prove(F, Nodes) :-
      atomicFormula(F),
      proveAtomicFormula(F, Nodes).


proveAtomicFormula(F, Nodes) :-
        % check nonrecursivity
      predicate(F, P),
      non_member(P, Nodes),
        % call the prover
```

```
        proveAtomicFormula_1(F, Nodes, P).

proveAtomicFormula_1(F, Nodes, P) :-
        implicationInTheory(F, Body),
        prove(Body, [P|Nodes]).

proveAtomicFormula_1(F, _Nodes, _P) :-
        atomicInTheory(F).


proveAtomicFormula_1(F, _Nodes, P) :-
        inputPredicate(P).
```

The last change to the program involves collecting all leaf atomic formulas used in the derivation that were produced by the added clause. The predicates

```
prove(F,LeavesIn, LeavesOut, Nodes)
```

and

```
proveAtomicFormula(F, LeavesIn, LeavesOut, Nodes)
```

search the SLD-derivation tree and whenever a successful branch is found, all such leaf atomic formulas are added to the list `LeavesIn` to produce a list `LeavesOut`.

```
% The prover is called with "Nodes" instantiated to "[]" and
% LeavesIn instantiated to "[]".
prove(F, Leaves) :-
        prove(F, [], Leaves, []).

prove(L and R, LeavesIn, LeavesOut, Nodes) :-
        prove(L, LeavesIn, LeavesNext, Nodes),
        prove(R, LeavesNext, LeavesOut, Nodes).
```

```
prove(F, LeavesIn, LeavesOut, Nodes) :-
        atomicFormula(F),
        proveAtomicFormula(F, LeavesIn, LeavesOut, Nodes).


proveAtomicFormula(F, LeavesIn, LeavesOut, Nodes) :-
          % nonrecursivity check
        predicate(F, P),
        non_member(P, Nodes),
          % call the prover
        proveAtomicFormula_1(F, LeavesIn, LeavesOut, Nodes, P).


proveAtomicFormula_1(F, L, L, _, _) :-
        atomicInTheory(F).

proveAtomicFormula_1(F, LeavesIn, LeavesOut, Nodes, P) :-
        implicationInTheory(F, A),
        prove(A, LeavesIn, LeavesOut, [P | Nodes]).

proveAtomicFormula_1(F, LeavesIn, [F |LeavesIn], _, P) :-
        inputPredicate(P).
```

3.5.4 Implementation of the Conditions Combination Algorithm

Before we describe the conditions combination algorithm, we define forward and backward readings of conditional equivalences.

Given a conditional equivalence of the form

$$P_1 \cdots P_n \rightarrow (R_1 \cdots R_m \equiv Q_1 \cdots Q_l) \tag{3.41}$$

we shall call the formulas

$$P_1 \cdots P_n \wedge R_1 \cdots R_m \rightarrow Q_1$$
$$\cdots$$
$$P_1 \cdots P_n \wedge R_1 \cdots R_m \rightarrow Q_l$$

forward readings of the conditional equivalence (3.41) and formulas

$$P_1 \cdots P_n \wedge Q_1 \cdots Q_l \rightarrow R_1$$
$$\cdots$$
$$P_1 \cdots P_n \wedge Q_1 \cdots Q_l \rightarrow R_m$$

backward readings of the equivalence. This distinction together with a simple convention of writing atomic formulas with predicates that are "more lexical" on the LHS of conditional equivalences allows us save some time and energy while constructing normalized theories.

The idea is that forward readings of the conditional equivalences are much more likely to be used while proving output atomic formulas whereas backward reading of equivalences are much more likely to be used while proving input atomic formulas. We would like to exploit this possibility to cut the size of theory in half during the steps 1 and 2 of the conditions combination algorithm.

While deriving all possible conditions for an output atomic formula O during step 1 of the conditions combination algorithm, only forward readings of equivalences together with the result sets $SO'_1 \cdots SO'_n$ of the same step 1 of the conditions combination algorithm from the processing of inherited theories $T_1 \cdots T_n$ are used.

Similarly during derivation of the conditions for an input atomic formula during the step 2 of the condition combination algorithm only backward readings of the equivalences and the results $SI'_1 \cdots SI'_n$ of the theories $T_1 \cdots T_n$ are used. The following example demonstrates the point.

Example 3.14

T_1 :

$$\begin{aligned} Input &= \{a_1\} \\ Output &= \{b_1\} \\ Assumption &= \{\} \\ Pairs &= \{(a_1, b_1)\} \\ PositiveLogic &= \{a_1 \equiv b_1\} \\ NegativeLogic &= \{\} \end{aligned}$$

T_2 :

$$\begin{aligned} Input &= \{a_2\} \\ Output &= \{b_2\} \\ Assumption &= \{\} \\ Pairs &= \{(a_2, b_2)\} \\ PositiveLogic &= \{a_2 \equiv b_2\} \\ NegativeLogic &= \{\} \end{aligned}$$

T :

$$\begin{aligned} Inherits &= \{T_1, T_2\} \\ &\quad \% \text{ we suppose, that theory T} \\ &\quad \% \text{ inherits partial results from} \\ &\quad \% \text{ theories } T_1 \text{ and } T_2 \\ Input &= \{a_1, a_2\} \\ Output &= \{b\} \\ Assumption &= \{\} \\ Pairs &= \{(a_1, b), (a_2, b)\} \\ PositiveLogic &= \{b_1 \equiv b, b_2 \equiv b\} \\ NegativeLogic &= \{\} \end{aligned}$$

During normalization of theories T_1 and T_2, the conditions combination algorithm produced intermediate results

$$SI'_1 = \{b_1 \rightarrow a_1\}$$
$$SO'_1 = \{a_1 \rightarrow b_1\}$$

$$SI'_2 = \{b_2 \rightarrow a_2\}$$
$$SO'_2 = \{a_2 \rightarrow b_2\}$$

To normalize theory T, the algorithm has to compute set SO of patterns of conditions for atomic formula b and sets SI_1 and SI_2 of patterns of conditions for atomic formulas a_1 and a_2.

For computing the set SO, the conditions combination algorithm calls conditions construction algorithm with the following assignment in step 1:

T = {
$a_1 \rightarrow b_1$, % from SO'_1
$a_2 \rightarrow b_2$, % from SO'_2
$b_1 \rightarrow b, b_2 \rightarrow b$ % as forward readings
% of the equivalences from theory T
}
I = $\{a_1, a_2\}$
S = SO

Similarly, the set SI is constructed by a call to the conditions construction algorithm using the assignment:

T = {
$b_1 \rightarrow a_1$ % from set SI'_1
$b_2 \rightarrow a_2$ % from set SI'_2
$b \rightarrow b_1, b \rightarrow b_2$ % as backward readings of
% the equivalences from the theory T
}
I = $\{b\}$
S = SI

■

A similar approach was taken in [23], where only forward readings of the conditional equivalences were used during the justification of the conditions of conditional equivalences during the translation process.

The implementation contains a switch that allows the designer to choose one of the two possibilities. If the switch is on, the above mentioned distinction between the forward and backward readings of equivalences is made: forward readings are used during the construction of the set SO, backward readings are used during the construction of the set SI. If the switch is off, both readings are used all the time.

3.5.5 Simplification of the Results

All results from the conditions construction algorithm and conditions combination algorithm are checked in two aspects - for permissibility of assumptions and for redundancy of rules.

Permissibility of assumptions

Whenever the conditions construction algorithm generates a Horn clause of the form

$$A_1 \cdots A_n \wedge I_1 \cdots I_m \rightarrow O$$

or

$$A_1 \cdots A_n \wedge O_1 \cdots O_m \rightarrow I$$

where $A_1 \cdots A_n$ are assumption atomic formulas, $I, I_1 \cdots I_m$ are input atomic formulas and $O, O_1 \cdots O_m$ are output atomic formulas, each assumption atomic formula A_i is checked for its permissibility w.r.t. *NegativeTheory* and the set of atomic formulas

$$S = \{A_1 \cdots A_n, I_1 \cdots I_m\}$$

or

$$S = \{A_1 \cdots A_n, I\}$$

respectively.

We consider assumption A_i to be nonpermissible if there is a formula in *NegativeTheory* from which it directly follows that $\neg A_i$ holds in

$$NegativeTheory \cup S$$

Speaking more precisely, A_i is nonpermissible w.r.t. to *NegativeTheory* and the set of atomic formulas S if there is an atomic formula F from the set S, substitution g and formula of the form

$$(\forall)(X_1 \cdots X_k \neq Y_1 \cdots Y_k) \rightarrow (A_i' \rightarrow \neg F')$$

or

$$(\forall)(X_1 \cdots X_k \neq Y_1 \cdots Y_k) \rightarrow (F' \rightarrow \neg A_i')$$

from *NegativeTheory* s.t.

$$A_i'g = A_i, F'g = F \text{ and } X_1 \cdots X_k g \neq Y_1 \cdots Y_k g$$

or if there is a subset $\{F_1 \cdots F_l\}$ of S, substitution g and formula of the form

$$(\forall)F_1' \cdots F_l' \rightarrow \neg A_i'$$

from *NegativeTheory* s.t.

$$F_1' \cdots F_l'g = F_1 \cdots F_l \text{ and } A_i'g = A_i$$

Similar checks are performed on the results

$$A_1 \cdots A_n \wedge I_1 \cdots I_m \rightarrow (I \equiv O)$$

of conditions combination algorithm.

Redundancy

It is obvious that if there are two rules with the same conclusion and the condition of the first rule is stronger than the condition of the second rule, then the first rule is redundant and can be removed from the resulting set. The redundancy simplification is performed on the results of both the condition construction and combination algorithms.

We consider condition $P_1 \cdots P_n$ stronger than condition $Q_1 \cdots Q_m$ if there is a substitution f s.t. $\{Q_1 \cdots Q_m\}f$ is a subset of $\{P_1 \cdots P_n\}$.

3.6 Summary

In this chapter we introduced a restricted version of LDTs as logic theories that describe the relationship between input atomic formulas representing words of the natural language

input in their context, and the output atomic formulas representing correspondence to the database. We have also shown that the restricted LDTs can be normalized, i.e. given a restricted LDT, we are able to produce set of rules with input atomic formulas on the "left hand side" and the output atomic formula on the "right hand side". The set of rules (which we call a normalized LDT) depends only on the meaning of the theory. For example, it does not depend on what kind of intermediate predicates the restricted LDT uses or whether the restricted LDT is written in a way that the reasoning process can run within the restricted LDT efficiently or (more importantly) inefficiently.

It is obvious that if we supply an efficient translation algorithm that can use the normalized theory, such an algorithm will, together with the normalization algorithms, form a sound basis for really declarative and efficient semantic processor of a natural language interface to the database.

Chapter 4

Translation Using Normalized LDTs

Now that we have seen how to create a normalized LDT from a restricted LDT, we can turn to the issue of how the normalized LDT can be used to translate linguistic logic representations of English sentences into database logic representations.

We first introduce a sound algorithm for the translation using the normalized LDTs. Then we consider the efficiency of the algorithm, particularly of a part that takes care of selection of the right rule from the set of all possibilities. Finally we shall introduce the actual data structures and algorithms used in an implementation.

The algorithms introduced in this chapter were implemented in Prolog.

4.1 Sound Translation Algorithm

Our translation algorithm is very similar to that presented in [23]. Rayner's algorithm can actually be used for the translation using the normalized theory. The reason why we do not accept his algorithm without changes is very simple. The normalized theory contains all patterns of the rules that directly map input atomic formulas into output atomic formulas, explicitly. It is obvious that reasoning within the theory is not able to produce any more rules of the given form. We believe that it is possible to write an algorithm that makes much less use of the runtime reasoning than Rayner's algorithm does.

We shall first introduce the modified translation algorithm for translation of conjunctions

of atomic formulas. Then we prove its correctness. At the end of the section we introduce the translation algorithm for a richer logical language that uses connectives "$\vee$", "$\rightarrow$", and "$\wedge$" and higher order logical formulas.

Algorithm 4.1 *Algorithm for translation of conjunctions of atomic formulas*

Input: *conjunction (set) of ground atomic formulas*

$$II = \{I_1 \cdots I_n\} \text{ (input atomic formulas)}$$

normalized LDT T (a set of CE(T)s) (input theory)

conjunction of ground atomic formulas CC (conjunctive context)

Output: *set of output atomic formulas OO and set of assumptions AA, s.t.*

$$T \models CC \wedge AA \rightarrow (II \equiv OO)$$

Algorithm:

```
set OO := {}
AA := {}
i = 0
while i < n do
   /* Invariant */
  i = i + 1
(1) if there is CE(T) (A, C, I', O') in normalized LDT T,
      and substitution f  s.t.
      (II - {Ii}) and CC -> C f
      and
      Ii = I' f
    then
      set OO := OO union {O' f}
      set AA := AA union A f
    else
      fail
    endif
  end while
  /* Invariant */
```

Proof. We prove by induction, that in the places marked by the comment `/* Invariant */` the following invariant holds

$$T \models CC \wedge AA \rightarrow (II \equiv OO \wedge (I_{i+1} \cdots I_n)) \tag{4.1}$$

$$T \models AA \wedge OO \rightarrow I_1 \cdots I_i \tag{4.2}$$

Case $i = 0$ This case is trivial

Case $i = m + 1$ Suppose that the invariant holds for $i = m$. We show, that the invariant holds for $i = m + 1$

We shall write AA_m and OO_m to denote AA and OO respectively in the place marked by invariant, when $i = m$.

We shall show (4.1) first

According to Induction Hypothesis (IH):

$$T \models CC \wedge AA_m \rightarrow (II \equiv OO_m \wedge I_{m+1} \cdots I_n) \tag{4.3}$$

and

$$T \models AA_m \wedge OO_m \rightarrow I_1 \cdots I_m \tag{4.4}$$

Consider the formula $OO_m \wedge I_{m+1} \cdots I_n$. Suppose that the algorithm reached the invariant point for $i = m + 1$. By the definition of NCE(T)

$$T \models A \wedge C \rightarrow (I' \equiv O')$$

therefore

$$T \models Af \wedge Cf \rightarrow (I'f \equiv O'f)$$

From $I'f = I_{m+1}$ we have

$$T \models Af \wedge Cf \rightarrow (I_{m+1} \equiv O'f) \tag{4.5}$$

According to IH

$$T \models AA_m \wedge OO_m \rightarrow I_1 \cdots I_m$$

So

$$T \models AA_m \wedge OO_m \wedge I_{m+2} \cdots In \rightarrow (II - \{I_{m+1}\})$$

According to the algorithm

$$CC \wedge (II - \{I_{m+1}\}) \rightarrow Cf$$

So

$$T \models CC \wedge AA_m \wedge OO_m \wedge I_{m+2} \cdots I_n \rightarrow Cf$$

and obviously

$$T \models CC \wedge AA_m \wedge OO_m \wedge I_{m+2} \cdots I_n \wedge Af \rightarrow Cf \wedge Af \tag{4.6}$$

From (4.5) and (4.6) and the translation rules of AET (2.14), it follows that

$$T \models CC \wedge AA_m \wedge Af \wedge OO_m \wedge I_{m+2} \cdots I_n \rightarrow (I_{m+1} \equiv D'f) \tag{4.7}$$

From the translation rule of AET (2.8), it follows that

$$\begin{gathered} T \models CC \wedge AA_m \wedge Af \rightarrow \\ (OO_m \wedge I_{m+1} \cdots I_n \equiv OO_m \wedge O'f \wedge I_{m+2} \cdots I_n) \end{gathered}$$

Because $OO_m \wedge O'f = O_{m+1}$

$$\begin{gathered} T \models CC \wedge AA_m \wedge Af \rightarrow \\ (OO_m \wedge I_{m+1} \cdots I_n \equiv OO_{m+1} \wedge I_{m+2} \cdots I_n) \end{gathered} \tag{4.8}$$

From (4.8) and from IH (4.3) and according to the algorithm that sets AA_{m+1} to $AA_m \wedge Af$

$$T \models CC \wedge AA_{m+1} \rightarrow (II \equiv OO_{m+1} \wedge I_{m+2} \cdots I_n)$$

□

Proof of (4.2) is very similar to that of (4.1).

Proof. By the definition of NCE(T)

$$T \models Af \wedge O'f \rightarrow I'f$$

and therefore

$$T \models Af \wedge O'f \rightarrow I_{m+1} \tag{4.9}$$

From (4.4) and (4.9) it follows that

$$T \models AA_m \wedge Af \wedge OO_m \wedge O'f \rightarrow I_1 \cdots I_{m+1}$$

According to the algorithm $AA_{m+1} = AA_m \wedge Af$ and $OO_{m+1} = OO_m \wedge O'f$, and we have

$$T \models AA_{m+1} \wedge OO_{m+1} \rightarrow I_1 \cdots I_{m+1}$$

□

Now we are at the point where we can introduce a translation algorithm that takes as an input arbitrary logical formulas. Also in this case, the algorithm is very similar to the one in ([23]). The algorithm recurses down the formula while collecting the conjunctive context. When it reaches the basic case (in Rayner's case atomic formula, in our case a conjunction of atomic formulas), it tries to translate it using LDT and the collected conjunctive context.

We describe the main ideas behind the modification first, then we present the algorithm and at the end we demonstrate the idea on a simple example.

The modification touches a few aspects of the algorithm. As was mentioned above, in our algorithm, we consider conjunction of atomic formulas as a basic case, i.e. the algorithm recurses down collecting conjunctive context until it finds a conjunction of atomic formulas. Then the algorithm for translation of conjunctions of atomic formulas is used to translate it.

To accommodate the change described above conveniently, there is a simple preprocessing of the input formula involved. The preprocessing algorithm also does the following. Whenever it finds a set of formulas connected together by the conjunctive "$\wedge$", it sorts the set so that the atomic formulas from the set are grouped together, nonatomic formulas are grouped together and both groups are again connected by the conjunctive "$\wedge$". For example, formula

$$(a \wedge ((b \vee c) \wedge (d \wedge \neg e)))$$

will be transformed into

$$(a \wedge d) \wedge ((b \vee c) \wedge \neg e)$$

where atomic formulas a and d are put together, nonatomic formulas $(b \vee c)$ and $(\neg e)$ are put together and both groups are connected with "$\wedge$".

The modified algorithm also "collects" conjunctive context differently. The algorithm collects only atomic formulas that can be reached by conjunction and disregards anything else. Consider the conjunctive context of atomic formula a in the formula

$$(b \vee c) \wedge d \wedge e \rightarrow a$$

Rayner's original algorithm produces $(b \vee c) \wedge d \wedge e$ as the conjunctive context whereas our algorithm produces only $d \wedge e$. Similarly, the conjunctive context of atomic formula a in the formula $a \wedge (b \vee c)$ would be $(b \vee c)$ in Rayner's case and *true* in our case.

Formally, our algorithm uses a different set of translation schemas. The modified translation schemas for translation of conjunction "$\wedge$" and "$\rightarrow$" are shown bellow, the rest remains the same (cf. chapter 2). Function f from a set of formulas into a set of sets of atomic formulas is the core of the suggested modification. This function simply takes a logical formula and produces a set of atomic formulas that can be reached from the top level of input logical formula by the conjunctive "$\wedge$".

$$\begin{array}{rcll} f(F) & = & \text{if} \quad F = P \wedge Q & \text{then} \quad f(P) \cup f(Q) \\ & & \text{else if} \quad F \text{ is atomic formula} & \text{then} \quad \{F\} \\ & & \text{else} \quad \{\} \ \% \text{ empty set} & \end{array}$$

$$\begin{array}{rl} & Context \wedge f(P) \rightarrow (Q \equiv Q') \\ \Rightarrow & Context \rightarrow (P \wedge Q \equiv P \wedge Q') \end{array} \tag{4.10}$$

$$\begin{array}{rl} & Context \wedge f(P) \rightarrow (Q \equiv Q') \\ \Rightarrow & Context \rightarrow (P \rightarrow Q \equiv P \rightarrow Q') \end{array} \tag{4.11}$$

It is easy to see that $F \rightarrow f(F)$ for each logical formula F. Correctness of the modification of the translation schemas trivially follows.

The fourth aspect is that the algorithm does not translate only one basic element in one run, but translates the whole formula at once. Preprocessing and a reduced collection of conjunctive context allow the recursive processing of an input formula in a special order. This order guarantees that the conjunctive context of a subformula translated at a particular moment consists only of the input atomic formulas and does not contain any already translated atomic formulas.

Example 4.1 As an example consider the already preprocessed formula

$$((ia \vee ib) \wedge ic) \rightarrow ((id \wedge ie) \wedge \neg if) \tag{4.12}$$

and a normalized LDT that translates i-s to o-s

$$(ia \equiv oa)$$
$$(ib \equiv ob)$$
$$(ic \equiv oc)$$
$$(id \equiv od)$$
$$(ie \equiv oe)$$
$$(if \equiv of)$$

In our example, by binary conjunction we mean an operation "$\wedge$" on two arguments, by n-ary conjunction we mean an operation "$\wedge$" on any number of arguments more than zero. E.g. formula $(a \wedge b) \wedge (b \wedge c)$ is a result of binary conjunction of two formulas $(a \wedge b)$ and $(b \wedge c)$ and n-ary conjunction of 4 formulas a, b, c, and d.

The algorithm begins with the whole formula

$$((ia \vee ib) \wedge ic) \rightarrow ((id \wedge ie) \wedge \neg if)$$

The left hand side of the implication can serve as a conjunctive context for the right hand side, but the right hand side has no similar effect to the left hand side. Therefore the algorithm will translate the right hand side first and collect ic as a conjunctive context.

The algorithm tries to translate the conjunction

$$((id \wedge ie) \wedge \neg if$$

Note, that whenever the algorithm finds a binary conjunction, any of its arguments cannot be an n-ary conjunction of atomic formulas mixed with non atomic formulas together. This is due to the preprocessing that puts atomic formulas connected by conjunctive "$\wedge$" together and nonatomic formulas together. Therefore whenever the algorithm finds a binary conjunction that is not already a basic case, its arguments are either basic case - conjunction of atomic formulas, or do not provide any conjunctive context for the other argument whatsoever. The algorithm tries to translate formula $\neg if$ first using a conjunctive context of (ic) (from above) and $id \wedge ie$ (from this level)

The algorithm recurses down from $\neg if$ to if and translates it to of. $\neg if$ is translated to $\neg of$

Then the algorithm translates $id \wedge ie$ using the conjunctive context ic. The algorithm for translation of conjunctions of atomic formulas produces a formula $od \wedge oe$.

The formula

$$(id \wedge ie) \wedge \neg if$$

is translated into

$$(od \wedge oe) \wedge \neg of$$

Then the algorithm translates the LHS of the implication.

$$(ia \vee ib) \wedge ic$$

The conjunctive context in this case is just *true*. Translation proceeds similarly as in the previous case. Disjunction $ia \vee ib$ is translated first using conjunctive context ic into $oa \vee ob$. Then ic is translated into oc using the empty conjunctive context.The whole LHS is translated into

$$(oa \vee ob) \wedge oc$$

The whole formula is thus translated into

$$((oa \vee ob) \wedge oc) \rightarrow ((od \wedge oe) \wedge \neg of)$$

■

Algorithm 4.2 *Translation Using NLDTs*

Input: *input logical formula* F_{input}

normalized LDT w.r.t. RLDT T - NLDT(T)

Output: *logical formula* F_{output} *and set of assumptions A s.t.*

$$T \models AA \rightarrow (F_{input} \equiv F_{output})$$

Algorithm:

Set the set A to the empty set.

Traverse formula F_{input} using the schemas until a conjunction of atomic formulas is reached. Whenever there is a choice regarding which constituent of the formula should be traversed first, use the following rules:

if *the formula has form* $A \wedge B$ **then**
  if
    A is a conjunction of atomic formulas
    or an atomic formula and B is not
  then
    visit B first, then visit A
  else
    visit A first, then visit B
  end if
else if
  the formula has the form $A \rightarrow B$
  or $A \vee B$
then
  visit B first, then A
end if

When a conjunction of atomic formulas II is reached, use the algorithm for translation of conjunction of atomic formulas with conjunctive context CC that was accumulated during the recursive descent.

Then substitute the result OO of the algorithm for subformula II, set the set A to $A \cup AA$ *and continue with traversing the input formula*

4.2 Searching for CEs

The critical part of the algorithms described in the previous section is to find a set of CEs (or rules of a normalized theory) that can be applied within a given conjunctive context. An obvious solution is to check each rule individually, but this can take a substantial amount of time. Fortunately, there are certain features of normalized theories that can be exploited to improve the efficiency of the searching process.

All such features are consequences of the fact that an NLDT can be viewed as a logical theory based on natural language phrases, i.e. given a CE

$$(Context, Assumptions, InputAF, OutputAF)$$

the formula

$$Context \wedge InputAF$$

represents a logical representation of a natural language phrase meaningful to the underlying database.

The rest of the section is structured as follows. We first present an assumption about the structure of the formulas of the form

$$Context \wedge InputAF$$

namely the approximate correspondence of predicates of input atomic formulas and words of the natural language phrases. Then we present some observations about the *sets* of logical formulas representing natural language phrases under the taken assumption. At the end of the section we demonstrate the data-structure upon which a searching algorithm is based and heuristics used to construct such data-structures.

4.2.1 Assumptions and Observations

We assume a close correspondence between predicates of input atomic formulas and words of the natural language utterances. For example, we assume that the noun "book" will be represented by an input atomic formula with predicate *book_entity* in all logical representations of English phrases that contain the noun "book". By close correspondence we mean that exceptions are permitted, but they are very rare.

This assumption allows us to relate certain observations about words that occur in a typical input English sentence to input atomic formulas that occur in the logical representation of the sentence.

Under the above assumption, it is plausible to assume that the structure of input atomic formulas is not deep, i.e. not many functional symbols are used. As an example compare formula

$$student(X) \wedge take(E, X, Y) \wedge course(Y)$$

with formula

$$a(b(c(d(e(f(g(e,h),I),J),K),L),M),O) \wedge f(f(f(f(f(f(f(f(X)))))))))$$

Observations

1. Observation about partitioning of the set of CEs

   Each English phrase meaningful to the database (as represented by an NLDT) contains only a small number of words in comparison to the size of the lexicon, so each word selects a small number of the phrases (in which the word occurs) in comparison to the number of phrases represented by the NLDT. Moreover the phrase selected by one particular word is semantically related to that particular word. So, words that are semantically far away from each other are inclined to select disjoint sets of phrases.

   If we use our assumption about the correspondence between words and input atomic formulas, we can observe that given an NLDT (set of logical representations of phrases), we can find a set of input predicates that can identify almost disjoint subsets of formulas. We also observed that this partitioning can be done on the disjoint subsets recursively.

   As an example, consider the set of logical representations of phrases about phone number and student number (slightly simplified)

$$phone(X) \wedge number(X)$$
$$phone(X)$$
$$student(X) \wedge number(X)$$
$$student(X) \wedge id(X)$$

Input atomic formulas $phone(X)$ and $student(X)$ divide the set into two disjoint subsets:

$$\{phone(X) \wedge number(X), phone(X)\}$$
$$\{student(X) \wedge number(X), student(X) \wedge id(X)\}$$

Subset

$$\{student(X) \wedge number(X), student(X) \wedge id(X)\}$$

can be further partitioned into subsets:

$$\{student(X) \wedge number(X)\}$$
$$\{student(X) \wedge id(X)\}$$

using input predicates *number* and *id*.

Although in general, partitioning does not have to be perfect, we believe that this feature of NLDTs forms a reasonable basis for searching heuristics.

2. Observation about variable sharing

The second observation deals with sharing of variables by atomic subformulas of logical representations of phrases within the NLDT. The term "connected" formalizes this concept.

Definition 4.1 (A is connected to B w.r.t. F) *Atomic subformula A of formula F is connected to atomic subformula B of formula F if A and B contain at least one common variable or constant, or there is an atomic subformula C of formula F s.t. A is connected to C w.r.t. F and C is connected to B w.r.t. F*

We assume that in almost all logical representations F

$$F = Context \wedge InputAF$$

of an English phrase from NLDT, all atomic subformulas of F are connected to each other.

4.2.2 Searching Algorithm

In this subsection we present the main heuristics and the algorithm that uses them to find a subset of the NLDT that matches a set of atomic formulas, i.e. given a set

$$LL = \{L_1 ... L_n\}$$

of atomic formulas and set of CEs

$$(Assumption, Condition, InputAF, OutputAF)$$

we find a CE and a substitution f , such that

$$LL \rightarrow (Condition \wedge InputAF)f$$

This algorithm makes the search for the "right" CEs performed by the algorithm for translation of conjunctions of atomic formulas more efficient than a simple search through all the CEs of the NLDT.

The main idea is similar to the idea behind trie structures [1]. We simply build a search tree with a fixed root whose arcs are atomic formulas. Each path in the search tree represents a conjunction of atomic formulas. If a path from the root of the search tree to a node represents a formula

$$Condition \wedge InputAF$$

for some CE, the node will contain information about the CE. Searching the tree begins at the root. The searching algorithm tries to match one of the arcs coming from the root with one of the atomic formulas from the input set LL. If such an arc to another node is found, the algorithm binds appropriate variables and continues to explore the tree from the new node using depth first search. If such an arc is not found, the algorithm backtracks. The following example demonstrates the idea.

Example 4.2 Suppose, that we have an NLDT that consists of the following CEs:

$$\begin{aligned} & \% \ student\ takes\ a\ course && (4.13)\\ (Context &= student(X) \wedge course(Y),\\ Assumption &= \{\},\\ InputAF &= take(E,X,Y),\\ OutputAF &= db\_take(E,X,Y)) \end{aligned}$$

$$\begin{aligned} & \% \ professor\ teaches\ a\ course && (4.14)\\ (Context &= professor(X) \wedge course(Y),\\ Assumption &= \{\},\\ InputAF &= teach(E,X,Y),\\ OutputAF &= db\_teach\_course(E,X,Y)) \end{aligned}$$

$$\% \; professor \; teaches \; a \; student" \qquad (4.15)$$
$$(Context = professor(X) \wedge student(Y),$$
$$Assumption = \{\},$$
$$InputAF = teach(E, X, Y),$$
$$OutputAF = db\_teach\_student(E, X, Y))$$

and the tree on figure 4.1. $Node3$ contains CE (4.13), $Node6$ contains CE (4.14) and $Node7$ contains CE (4.15)

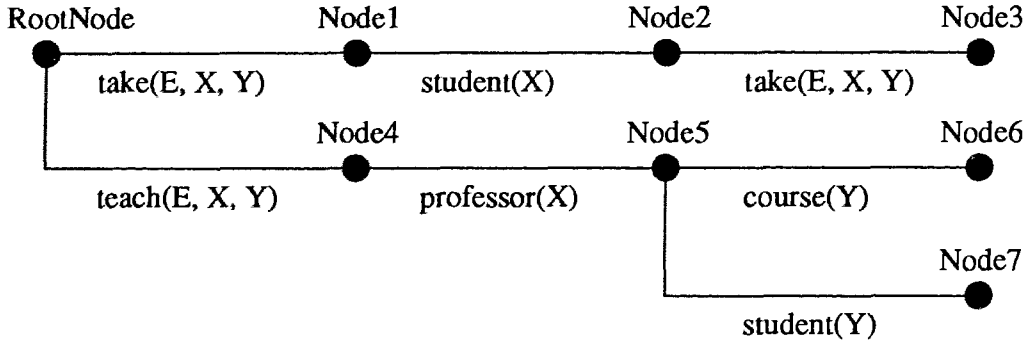

Figure 4.1: Tree representing CEs (4.13 ... 4.15).

For input set

$$LL = \{student(x), take(e, x, y), course(y)\}$$

the searching algorithm will explore nodes

$$\{RootNode, Node1, Node2, Node3\}$$

During this process, variables E, X, Y will be bound to e, x, y respectively. $Node3$ contains information that CE (4.13) can be applied. ∎

4.2.3 Tree Construction Algorithm

The graph structure from the example has two nice properties that deal with determinism for certain kinds of input and the binding of variables during the searching process.

It is easy to see that the graph from our example can be searched without exploring useless branches if the input set LL consists of formulas present in exactly one CE from

the NLDT (i.e. if the input corresponds to exactly one representation of an English phrase from the NLDT), or its variants. In general, this property holds for any NLDT that fully matches the observation about partitioning of the set of CEs (observation 1), and whose logical representations of English phrases do not contain two atomic formulas with the same predicate. In such cases, it is possible to construct a graph with the searching algorithm that searches the NLDT without exploring useless branches for the input sets corresponding to the CEs from the NLDT.

Thanks to the observation about variable sharing (observation 2), it is possible to construct the search tree in a way that only the edges directly flowing from the root node will have all their variables unbound during the search process. The following example shows how this reduces the nondeterminism during the search process.

Example 4.3 Suppose that NLDT contains the following CEs:

$$\begin{aligned} & \% \textit{ student takes a course} && (4.16)\\ (\mathit{Context} &= \mathit{student}(X) \wedge \mathit{course}(Y),\\ \mathit{Assumption} &= \{\},\\ \mathit{InputAF} &= \mathit{take}(E,X,Y),\\ \mathit{OutputAF} &= \mathit{db\_take}(E,X,Y)) \\ & \% \textit{ professor teaches a student"} && (4.17)\\ (\mathit{Context} &= \mathit{professor}(X) \wedge \mathit{student}(Y),\\ \mathit{Assumption} &= \{\},\\ \mathit{InputAF} &= \mathit{teach}(E,X,Y),\\ \mathit{OutputAF} &= \mathit{db\_teach\_student}(E,X,Y)) \end{aligned}$$

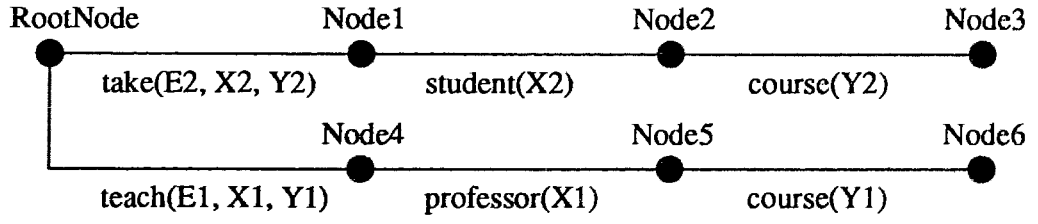

Figure 4.2: Tree representing CEs (4.16) and (4.17).

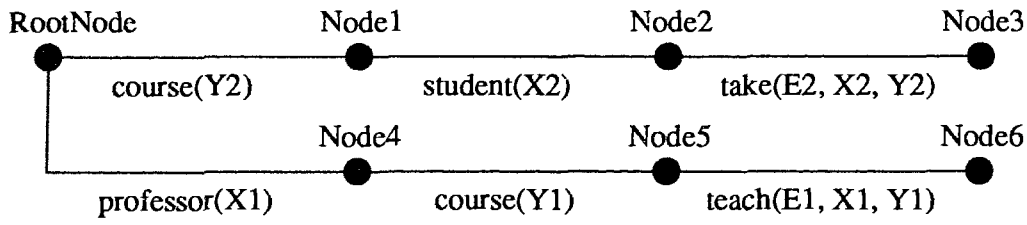

Figure 4.3: Tree representing CEs (4.16) and (4.17). Second version.

Then consider the search tree in figure 4.2 where $Node3$ contains CE (4.16) and $Node6$ contains CE (4.17), and the search tree in figure 4.3 where $Node3$ contains CE (4.16) and $Node6$ contains CE (4.17). Also consider an input sentence that contains two phrases

$$professor(p_1) \wedge teach(e_1, p_1, s_1) \wedge student(s_1)$$

and

$$student(s_2) \wedge take(e_2, s_2, c_2) \wedge course(c_2)$$

To search the tree in figure 4.2, the algorithm starts by matching an atomic formula $take(e_2, s_2, c_2)$ with the arc from $RootNode$ to $Node1$ of the tree. This binds variables E_2,X_2 and Y_2 to e_2, s_2 and c_2 respectively. The arc $student(X_2)$ that goes from Node1 becomes $student(s_2)$ and this can match only atomic formula $student(s_2)$ from the input sentence. On the other hand, when the algorithm starts to search the tree in figure 4.3, e.g. by matching $course(c_2)$ from input sentence, only the binding for variable Y_2 to c_2 is introduced. The nondeterminism occurs when the algorithm tries to match arc $student(X)$ against the input sentence, because two bindings ($X \rightarrow s_1$ and $X \rightarrow s_2$) are possible. ∎

The search tree construction algorithm is a greedy algorithm based on the heuristics described above. The input for the algorithm is a set of CEs. The algorithm constructs the tree from the root node with the whole set of CEs assigned to it.

The main part of the algorithm takes a partially constructed search tree and a leaf node of the tree with a set of CEs assigned to it. If some CEs from the set correspond to the leaf node, they are assigned to it in the output graph. The rest of the CEs are divided into smaller subsets using a sequence of "dividing" atomic formulas. For each "dividing" atomic

formula there is a subset of the rest of the CEs that correspond to it, i.e. it contains a variant of the "dividing" atomic formula.

The "dividing" atomic formulas are picked from the atomic formulas of CEs in a greedy fashion. Preference is given to atomic formulas that are connected to one of the arcs in the path from *Root* to the leaf node. If some part of the sequence of "dividing" atomic formulas was already picked, the next "dividing" atomic formula is the one that selects a set of CEs with minimal intersection with the CEs selected by the previous "dividing" atomic formulas of the part of the sequence. Being "connected" has higher priority then having smaller intersection.

Then the algorithm expands the leaf node by creating a set of arcs from the leaf node to new leaf nodes. The arcs are marked by "dividing atomic formulas". Each new leaf node is assigned a subset of CEs that corresponds to the path from the *Root* to the new leaf node. The paths for the new nodes are the same except for the dividing atomic formulas assigned to the new arcs. If a CE corresponds to more than one new node, only one new node is selected. Then the variables in the "dividing" atomic formulas and corresponding variables in the CEs are bound by unique constants.

The algorithm continues recursively until all CEs are assigned to some nodes in the output graph.

The following example shows how the search graph is constructed given an NLDT.

Example 4.4 Consider the following CEs:

$$\begin{aligned} & \% \textit{ student takes a course} && (4.18)\\ (\textit{Context} &= \textit{student}(X_1) \wedge \textit{course}(Y_1),\\ \textit{Assumption} &= \{\},\\ \textit{InputAF} &= \textit{take}(E_1, X_1, Y_1),\\ \textit{OutputAF} &= \textit{db\_take}(E_1, X_1, Y_1)) \\ & \% \textit{ professor teaches a course} && (4.19)\\ (\textit{Context} &= \textit{professor}(X_2) \wedge \textit{course}(Y_2),\\ \textit{Assumption} &= \{\},\\ \textit{InputAF} &= \textit{teach}(E_2, X_2, Y_2),\\ \textit{OutputAF} &= \textit{db\_teach\_course}(E_2, X_2, Y_2)) \end{aligned}$$

$$\begin{aligned} & \% \; professor \; teaches \; a \; student \qquad (4.20) \\ (Context &= professor(X_3) \wedge student(Y_3), \\ Assumption &= \{\}, \\ InputAF &= teach(E_3, X_3, Y_3), \\ OutputAF &= db\_teach\_student(E_3, X_3, Y_3)) \end{aligned}$$

The algorithm starts with the root node and all three CEs are assigned to it. Then the algorithm randomly picks the first "dividing" atomic formula e.g. $take(E_1, X_1, Y_1)$ and creates the subset of CE(LTD)s that contain a variant of the atomic formula
$take(E_1, X_1, Y_1)$:

$$\begin{aligned} \{(Context &= student(X_1) \wedge course(Y_1), \\ Assumption &= \{\}, \\ InputAF &= take(E_1, X_1, Y_1), \\ OutputAF &= db\_take(E_1, X_1, Y_1))\} \end{aligned}$$

Then the algorithm picks the second "dividing" atomic formula. Atomic formulas

$$professor(X_2), professor(X_3),$$
$$teach(E_2, X_2, Y_2), teach(E_3, X_3, Y_3)$$

are preferred, because they select CEs that are not selected by the previous "dividing" atomic formula $take(E_1, X_1, Y_1)$. Let us suppose that $professor(X_2)$ was picked. The $professor(X_2)$ selects CEs (4.19) and (4.20). The algorithm binds appropriate variables and expands the graph as depicted in figure 4.4. In the search graph, $Node1$ contains CE (4.18) with substitutions $E_1 \rightarrow e_1$, $X_1 \rightarrow x_1$ and $Y_1 \rightarrow y_1$. $Node4$ contains CEs (4.19) and (4.20) with substitutions $X_2 \rightarrow x_2$ and $X_3 \rightarrow x_2$.

The algorithm is called recursively with leaf nodes $Node1$ and $Node4$. Let us expand $Node1$ first. There is only one CE assigned to it and all its atomic formulas are connected to the arc $take(e_1, x_1, y_1)$. The expansion is therefore straightforward and the algorithm expands the node $Node1$ in two steps.

To expand node $Node4$, the algorithm prefers to choose "dividing" atomic formulas from $teach(E_2, x_2, Y_2)$ and $teach(E_2, x_2, Y_3)$, because those are the only atomic formulas that are connected to the previous arcs. Let us suppose that the

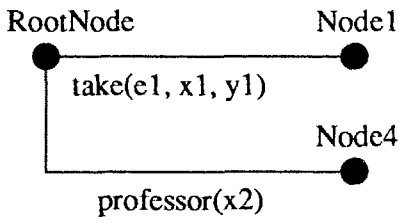

Figure 4.4: Search graph after RootNode expansion.

algorithm has chosen $teach(E_3, x_2, Y_3)$. The algorithm binds E_3 and E_2 to e_2, and Y_3 and Y_2 to y_2. The graph at this stage is depicted on figure 4.5. $Node3$ contains CE (4.18) where $E_1 \rightarrow e_1$, $X_1 \rightarrow x_1$ and $Y_1 \rightarrow y_1$. $Node5$ contains CEs (4.19) and (4.20) where $X_2 \rightarrow x_2$, $X_3 \rightarrow x_2$, $Y_2 \rightarrow y_2$, $E_2 \rightarrow e_2$, $X_3 \rightarrow x_2$, $Y_3 \rightarrow y_2$ and $E_3 \rightarrow e_2$.

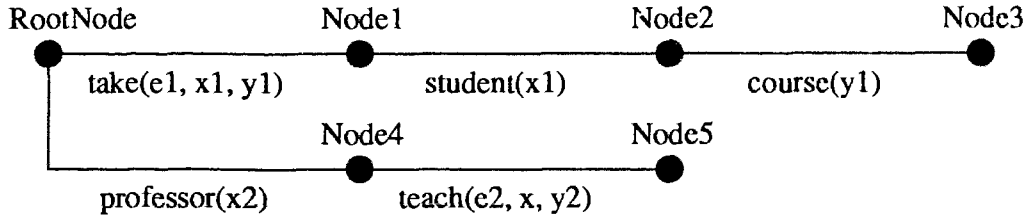

Figure 4.5: Search graph after expansion of nodes Node1 and $Node3$.

Then the algorithm expands $Node5$, which is trivial. At the end, the algorithm substitutes the unique constants used during the graph construction process with unique variables and we obtain the graph depicted in figure 4.6. $Node3$ contains CE (4.18) $Node6$ contains CE (4.19) and $Node7$ contains CE (4.20).

■

All the algorithms presented in this chapter were implemented in Prolog.

4.3 Summary

In this chapter we have shown an algorithm that can use information provided by a restricted LDT for translation of input logic formulas representing English sentences into output logic

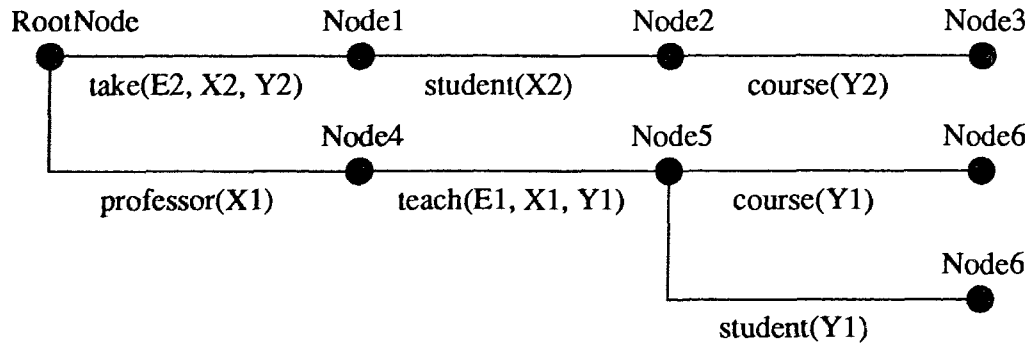

Figure 4.6: Finished search graph.

formulas corresponding to database queries. First, the normalization process is used to normalize the restricted LDT into a normalized LDT. The normalization is done off-line. Then, on-line, the normalized LDT is used by the translation algorithm to actually translate logic representations of sentences input by a user into logic representations understandable by the underlying database. Since the normalization process is done off-line, behaviour (and performance) of the translation algorithm depends only on the information content of the restricted domain theories, thus allowing the designer of the interface to describe the domain more declaratively without worrying about the system's on-line performance.

Chapter 5

Construction of Selectional Restrictions

In this chapter we present an algorithm that constructs selectional restrictions for unification based grammars out of NLDTs. In the first section of this chapter we introduce the general concepts of selectional restrictions and show on an example how they can be used for disambiguation during parsing. The second section deals with the actual process of constructing selectional restriction automatically using the knowledge of the domain recorded in terms of NLDTs.

5.1 Selectional Restrictions

By providing the syntactic parser of NLID with domain knowledge, the parser can use this information to reduce the ambiguity during the parsing process thus increasing efficiency of the NLID and reducing the time needed for producing an answer for the user. As an example consider the sentence similar to one given in [2]

A house was built by a river.

Without knowing that rivers cannot build houses, the sentence allows two readings with the following logical representations:

$$\begin{gathered} \textit{A river built a house.} \\ \exists E, R, H.river(R) \wedge house(H) \wedge build(E, R, H) \end{gathered} \tag{5.1}$$

$$\begin{array}{c} \textit{Somebody built a house close to a river.} \\ \exists E, S, H, R.build(E, S, H) \wedge house(H) \wedge by(H, R) \wedge river(R) \end{array} \quad (5.2)$$

By supplying the syntactic parser with the set of constraints that say which words can fill which argument positions of which other words, the first reading can be ruled out. We shall call such sets of constraints selectional restrictions.

The advantage of selectional restrictions is that they allow more efficient implementations. As described in [2], selectional restrictions can be implemented using a sort hierarchy and sortal constraints. Different sorts are represented using different Prolog terms and unification is used to check sort compatibility. Each logical language constructor (lexical predicates that correspond to the words of English language, quantifier constructions, lambda construction etc.) is assigned a sortal declaration. The sortal declaration consists of a sequence of input sorts that are expected at the constructor's argument positions, and an output sort that is assigned to the expression created using the constructor. The predicate $build(E, S, H)$ from our *house was built by a river* example can be assigned the sortal declaration

```
(
  [
    event,
    object(physical_object(animate(human, _))),
    object(physical_object(inanimate))
  ]
=>
  truth_value
)
```

and the predicate $river(R)$ can be assigned the sort

```
(
  [
    object(physical_object(inanimate))
  ]
=>
  truth_value
)
```

Such sortal declarations then impose sortal constraints on the expressions or variables that fill argument positions. If one variable or expression in a particular reading is assigned a set of incompatible sorts (sorts that do not unify), the reading is ruled out. This is the case for reading (5.1) of our example sentence. Variable R is assigned sort

```
object(physical_object(animate(human,_)))
```

by the predicate $build(E, R, H)$ and sort

```
object(physical_object(inanimate))
```

by predicate $river(R)$. Because these sorts obviously do not unify, the reading (5.1) is ruled out.

The encoding of sortal constraints is discussed in detail in [16].

Strict failure of the parse when the selectional restrictions are not satisfied is not the only way to apply the selectional restrictions. The second option is to view selectional restrictions as preferential information. In this case, the parse that does not satisfy selectional restrictions is not rejected, but the parses with fewer failures are preferred to parses with more failures.

Selectional restrictions are often domain dependent. For example, the verb book will allow different sets of objects in the domain of flight reservations (book a flight, book a ticket etc.) and in the domain of house building (book a carpenter, book Johnny Smith etc.)

5.2 Algorithm for Constructing Selectional Restrictions

In this section, we first introduce some assumptions and observations about the input of the algorithm (i.e. output of the parser and NLDTs). In the next subsection we shall use these assumptions while presenting the actual algorithm.

5.2.1 Indices and Attributes

As in the case of the translation algorithm, we assume close correspondence between predicates of input atomic formulas and words of the natural language utterances. As was mentioned before (c.f. section 4.2), this assumption allows us to relate certain observations about words that occur in a typical English sentence to input atomic formulas that occur

in logical representation of the sentence. Similarly to the translation algorithm from section 4.2 we assume that the structure of input atomic formulas that correspond to the words of the English language is not deep, i.e. there are not many functional symbols in the logic representations of the sentences. Such representations were used by a number of systems ([2], [15]) and even became examples of logical representations in an NLP textbook ([7]).

Under the above assumptions we can observe that it is possible to construct logical representations where most of the arguments of input atomic formulas representing nouns and most of the event arguments of input atomic formulas representing verbs introduce their "own" variables. That is, there are no two above mentioned arguments of nouns or verbs that will be bound to the same variable. We can also observe that for most of the English sentences it is possible to construct a logical representation where each variable is bound by at least one of the arguments of nouns or event arguments of verbs.

Example 5.1 As an example consider the sentence

The guy, that took a red book on a table beside a wall,
ran away.

and its logical representation

$$\exists G, ET, B, T, W, ER$$
$$guy(G) \wedge took(ET, G, B) \wedge book(B)$$
$$\wedge on(B, T) \wedge table(T) \wedge beside(T, W) \wedge wall(W)$$
$$run(ER, G) \wedge away(ER)$$

The variables G, ET, B, T, W, ER correspond to the noun *guy*, event parameter of the verb *took*, noun *book*, noun *table*, noun *wall* and event parameter of the verb *run* respectively. ∎

Our next assumption about the logical representation of English sentences is based on the observations above. We assume that there is a subset S of all the pairs of the form

$$(InputPredicate, ArgumentPosition)$$

that make sense in the given domain, s.t. for each logical representation of an English sentence the following is true. For each variable of the logical representation, there exists no more than one pair

$$(InputPredicate, ArgumentPosition)$$

from the set S s.t. input atomic formula with *InputPredicate* binds the variable at the *ArgumentPosition*. We shall call members of the set S *index argument positions* or in short *indices*. We shall call all other pairs of the form

$$(InputPredicate, ArgumentPosition)$$

that make sense in the given domain *attribute argument positions* or in short *attributes*. A similar assumption was taken (although implicitly) in [3].

We shall also assume that for all variables in the logical representations of English sentences, there is at least one pair

$$(InputPredicate, ArgumentPosition)$$

s.t. input atomic formula with *InputPredicate* binds the variable at the *ArgumentPosition*.

There are two potential problems with our assumption. One can object that there are domains for which there is no set S and correct logical representations for all sentences or phrases. In another words, there are domains and logical representations of sentences or phrases with

- variables that are not bound by an index
- variables bound by more than one index

A solution to the first problem is very simple. We can introduce a special input predicate $universal\_index(X)$ and add pair $(universal\_index, 1)$ to the set S. For each possible logical representation of a sentence with a variable X, that is unbound by an index it is possible to construct a new logical representation that adds term $universal\_index(X)$ to the scope of the quantifier that binds X. This solution also has an advantage for the semantic part of the system, because it provides an easy way to test for the nonpresence of the original indices for given variable. The semantic part of the system can translate $universal\_index(X)$ into *true*.

There is a similar solution to the second problem. We can introduce a predicate

$$same\_vars(X, Y)$$

that a logical representation can use when it needs to make the variables bound by two indices the same.

5.2.2 Computing Index - Attribute Compatibility

Under the assumption that each variable in the input sentence is bound by at most one index, we can construct selectional restrictions using knowledge of the domain encoded by the NLDT.

Unlike [2], we shall not construct selectional restrictions that constrain any pair of argument positions. Our goal is to constrain interaction between attributes and indices. Because each variable in the logical representation of the sentence is bound by exactly one index, by constraining the index - attribute compatibility we can achieve reasonable performance.

Knowledge encoded by the NLDT allows us to access the conditions under which input predicates can be translated into output predicates. The conditions are available in a very convenient form - expressed in terms of sets of input atomic formulas. We demonstrate how this fact can be used to construct selectional restrictions in the following example.

Example 5.2 Consider an NLDT that contains the following CEs for translation of the preposition "of" in the university domain.

$$\begin{gathered} name\ of\ student \\ (name(X) \wedge student(Y), true, of(X,Y), db\_graduate\_student(X)) \end{gathered}$$

$$\begin{gathered} student\ of\ professor \\ (student(X) \wedge professor(Y), true, of(X,Y), \\ db\_graduate\_student(X)) \end{gathered}$$

This means that to translate input atomic formula $of(X,Y)$, the indices that bind variables X and Y in the logical representation of a sentence can be either $(name, 1)$ and $(student, 1)$ or $(student, 1)$ and $(professor, 1)$ and nothing else. (If it is e.g. $(course, 1)$ that binds the variable X, then according to our assumption about the output of the parser, the sentence cannot contain any other index that binds the variable X. Therefore, neither of the CEs can be applied and the translation of the sentence fails). ∎

In general, given an NLDT and a set of indices, it is possible to produce a set of constraints only for some of the attributes (and their predicates) of the NLDT. The problem occurs when an input atomic formula can be translated into output atomic formula using a

CE in which a variable bound by the attribute is not bound by any of the indices. In this case, no constraints for the attribute are produced, and a warning for the designer of the RLDT (from which the NLDT is automatically constructed) is issued. Our practical experience with the system has shown that this situation occurs very rarely. The reason is very simple: a tight RLDT imposes constraints on attribute argument positions of adjectives, adverbs, prepositions or verbs in the form of noun or verb input predicates that introduce an index.

Our assumption that each variable in the logical representation of a sentence is bound by at least one index guarantees that constructed selectional restrictions will impose some constraints on the input sentence. If a variable is not bound by an index, nothing can be said about which attributes can or cannot bind it.

We tried two approaches for construction of the selectional restrictions. These are the template approach and the unification approach.

Template Approach

The template approach constructs templates of atomic formulas that contain at least one attribute. The following templates would be constructed for our example above:

```
selRes(of, (name, 1), (student, 1)).
selRes(of, (student, 1), (professor, 1)).
```

During parsing, the logical representation that is being constructed is simply checked against such templates, whenever the appropriate parts of the logical representation are instantiated. A Prolog implementation of this approach is straightforward.

Unification Approach

The unification approach is slightly weaker, because it constraints only interactions between a single attribute and a single index without taking wider context into account. Considering the above example, "name of professor" would be a valid phrase under this approach, whereas the template approach would not allow it. On the other hand, incorporation of the constraints constructed is easier for unification based grammars.

The algorithm first constructs a set of compatible attribute - index pairs. For the example above, the set of pairs is

$$\begin{aligned}(name, 1) &- (of, 1)\\ (student, 1) &- (of, 1)\\ (student, 1) &- (of, 2)\\ (professor, 1) &- (of, 2)\end{aligned}$$

Given a set of index - attribute pairs, the next goal is to create terms for each of the indices and attributes s.t. the unification of terms will allow the same interactions of attributes and indices as the set of input pairs. Because only the information about the interaction between indices and attributes is present, we assume that any attribute can interact with any other attribute.

The main idea is to create a set of terms of arity n whose arguments can be bound to values "+" or "−" or can be left unbound. We shall refer to the arguments $1 \cdots n$ of these terms as binary features $f_1 \cdots f_n$. Indices will either bind the features to the value "+" or leave it unbound. Attributes will similarly bind the features to the value "−" or leave it unbound. This will guarantee that all attributes can be combined together.

Consider the following example:

Example 5.3

$$\begin{aligned}Indices &- i_1, i_2\\ Attributes &- a_1, a_2, a_3\\ Pairs &- \{\ i_2 - a_1\\ &\quad\ i_1 - a_2\\ &\quad\ i_1 - a_3, i_2 - a_3\\ &\ \}\end{aligned}$$

It is possible to create a set of terms with binary features f_1 and f_2 that will correspond to the indices i_1 and i_2 respectively. Each index will bind its corresponding feature to "+". Each attribute will *not* bind the features that correspond to the indices that can be combined with that attribute, so the following

assignments can be obtained

$$
\begin{aligned}
i_1 &- f(+,-)\\
i_2 &- f(-,+)\\
a_1 &- f(-,-)\\
a_2 &- f(-,-)\\
a_3 &- f(-,-)
\end{aligned}
$$

■

It is easy to see that the procedure can be generalized for any input. Implementation of such an algorithm is again trivial. However, there are some ways how the number of features can be reduced. The simplest is to use only one feature for a set of indices that exhibit the same behavior w.r.t. attributes, i.e. they can interact with the same subsets of attributes. The second simplification can be viewed as a generalization of the first one. Consider the following example:

Example 5.4

$$
\begin{aligned}
\textit{Indices} &- i_1, i_2, i_3, i_4\\
\textit{Attributes} &- a_1, a_2, a_3, a_4\\
\textit{Pairs} &- \{\ a_1 - i_1, a_1 - i_2, a_1 - i_3\\
&\quad\ \ a_2 - i_2\\
&\quad\ \ a_3 - i_3\\
&\quad\ \ a_4 - i_1, a_4 - i_2, a_4 - i_3, a_4 - i_4\\
&\ \}
\end{aligned}
$$

$$
\begin{aligned}
i_1 &- f(+,-,-,-)\\
i_2 &- f(-,+,-,-)\\
i_3 &- f(-,-,+,-)\\
i_4 &- f(-,-,-,+)
\end{aligned}
$$

$$
\begin{aligned}
a_1 &- f(-,-,-,-)\\
a_2 &- f(-,-,-,-)\\
a_3 &- f(-,-,-,-)\\
a_4 &- f(-,-,-,-)
\end{aligned}
$$

Note the behavior of features f_1, f_2 and f_3 for attributes $a_1 \cdots a_4$. We can observe that f_1 is unbound if and only if f_2 is unbound and f_3 is unbound. If we change the assignment as follows (we remove feature f_1 and for all indices, where $f_1 = +$ we assign f_2 and $f_3 = +$)

$$i_1 - f(+,+,\_)$$
$$i_2 - f(+,\_,\_)$$
$$i_3 - f(\_,+,\_)$$
$$i_4 - f(\_,\_,+)$$

$$a_1 - f(\_,\_,-)$$
$$a_2 - f(\_,-,-)$$
$$a_3 - f(-,\_,-)$$
$$a_4 - f(\_,\_,\_)$$

this assignment preserves the original pairs. ∎

Again, the algorithm can very easily be generalized.

The feature representation producing algorithm was implemented in Prolog. The implementation produces selectional restrictions for an HPSG grammar [21] based parser build on top of the ALE system [5].

5.3 Implementation

There are two implementations of the system. HPSG based implementation uses the ALE system as the parser and the unification approach for encoding the selectional restrictions. DCG based implementation uses Prolog's built-in implementation of the DCG parser and the template approach for encoding the selectional restrictions. In this section, we shall present the results of implementation of our DCG version of the system.

5.3.1 System Design

The system was implemented in Prolog. The system accepts natural language queries in the form of list of words and returns responses in the form of tables written on the computer screen.

Syntactic part of the system is based on a small DCG grammar. The grammar is capable of parsing simple sentences and noun phrases that involve relative clauses and compound nominal constructions.

The grammar requires minimal information about the syntactic attributes of words that occur in the lexicon. The grammar relies heavily on the selectional restrictions supplied by the semantic part of the system. For example, the only syntactic information that is needed to define a verb is its category - the system does not need subcategorization information.

The system covers a very small university domain that includes concepts of faculty, students, departments, chairmen of the departments, courses and grades and some of their relationships. The semantic part of the system is able to produce queries for the following Prolog database:

```
dep(cs, 'computer science', idJohnSmith).
dep(math, 'math', idJimmyJones).

faculty(idJohnSmith, john, smith, cs, full).
faculty(idCharlieBrown, charlie, brown, cs, assistant).
faculty(idJimmyJones, jimmy, jones, math, associate).

student(idMilanMosny, milan, mosny, cs, 1).
student(idZuzkaRepka, zuzka, repka, cs, 2).
student(idPaulGreen, paul, greeen, math, 2).


course(idCmpt710, cmpt710, idJohnSmith).
course(idCmpt720, cmpt720, idCharlieBrown).
course(idMath710, math710, idJimmyJones).


enrl(id1, idMilanMosny, idCmpt710, a).
enrl(id2, idMilanMosny, idMath710, a).
enrl(id3, idZuzkaRepka, idCmpt710, a).
```

```
enrl(id4, idPaulGreen, idCmpt710, b).
enrl(id5, idPaulGreen, idCmpt720, b).
```

The RLDT that describes the domain defines a small inheritance hierarchy of the concepts (e.g. a person is an entity, a student is a person, take is an event, name is an entity, first name is a name) and a number of binary relations between them. The binary relations are combined into caseframes that describe the translation of verbs. The database specific part of the RLDT connects concepts and binary relations to the database columns and tables. There is also a domain independent part of the RLDT that says, for example, how the verb "be" is translated.

5.3.2 Sample Terminal Session

The interaction with the system is very simple. There is a top level predicate go that uses university domain theories to parse a sentence or a noun phrase and produce results. When the predicate go is called first time, the theories are normalized, a search graph is constructed and the selectional restrictions are created. Warnings are presented to the user at this time. Then the system processes the query as usual.

```
| ?- go.
|: [faculty].


[faculty]

selres: underspecified relation (arg 1): det
selres: underspecified relation (arg 2): det

********* Parse Done **************
********* Evaluating **************
iFaculty - xVar2

iFaculty - idCharlieBrown
```

```
iFaculty - idJimmyJones
iFaculty - idJohnSmith


yes

| ?-
```

Here the query "faculty" is input in the form of the list of atoms. Then, while constructing the selectional restrictions, the system warns the user about underspecified relation *det* (determiners were ignored in the DCG implementation). After the parse is done, the system produces results - a list of identifiers that identifies faculty records.

The following example shows a translation of the ambiguous word "people". The word "people" can refer either to faculty or to students.

```
[people]
********* Parse Done ***************
********* Evaluating ***************
iFaculty - xVar2

iFaculty - idCharlieBrown
iFaculty - idJimmyJones
iFaculty - idJohnSmith
********* Evaluating ***************
iStudent - xVar2

iStudent - idMilanMosny
iStudent - idPaulGreen
iStudent - idZuzkaRepka
```

The next example shows a simple yes/no question. The ambiguous proper noun "milan" representing a first name is in this case translated as referent to a student whose first name is milan.

```
[did,milan,take,any,courses]
********* Parse Done ***************
********* Evaluating ***************
iEnrl - xVar1        iStudent - xVar2        iCourse - xVar5

iEnrl - id1        iStudent - idMilanMosny        iCourse - idCmpt710
iEnrl - id2        iStudent - idMilanMosny        iCourse - idMath710
```

The next interaction shows a simple sentence involving the verb "be" at the input. In this examples and some others, we deleted some columns from the answers of the system, because of space limitations. The system produces a column for each verb and noun of an input.

```
[who,is,the,chairman,of,'computer science']
********* Parse Done ***************
********* Evaluating ***************
iFaculty - xVar1

iFaculty - idJohnSmith
```

The following query is an example of the sentence with a relative clause. Here the first name "John" is translated as a referent to the faculty.

```
[who,works,in,the,dep,that,john,is,the,chairman,of]
********* Parse Done ***************
********* Evaluating ***************
```

```
iFaculty - xVar10        iFaculty - xVar2

iFaculty - idJohnSmith        iFaculty - idCharlieBrown
iFaculty - idJohnSmith        iFaculty - idJohnSmith
```

The following sentence demonstrates the consequences of using the selectional restrictions during parsing. The RLDT "knows" how to translate the preposition "in" in a context of taking and a department as in "take (something) *in* department" but does not provide for translation of the same preposition "in" in the context of course and department as in "course *in* department". Therefore the parser produces only the parse where prepositional phrase is attached to the main verb.

```
[did,milan,take,any,courses,in,math,department]
********* Parse Done ***************
********* Evaluating ***************
iEnrl - xVar1        iStudent - xVar2        iCourse - xVar5

iEnrl - id2        iStudent - idMilanMosny        iCourse - idMath710
```

The rest of this section contains few more examples that demonstrate the interaction with the system.

```
[who,majors,in,math]
********* Parse Done ***************
********* Evaluating ***************
iStudent - xVar1        iStudent - xVar2        iDep - xVar5

iStudent - idPaulGreen        iStudent - idPaulGreen        iDep - math
```

```
[did,anybody,get,an,a,in,cmpt710]
********* Parse Done ***************
********* Evaluating ***************
iEnrl - xVar1        iStudent - xVar2        iGrade - xVar5

iEnrl - id1        iStudent - idMilanMosny        iGrade - a
iEnrl - id3        iStudent - idZuzkaRepka        iGrade - a



[the,grade,that,paul,made,in,cmpt710]
********* Parse Done ***************
********* Evaluating ***************
iGrade - xVar2        iEnrl - xVar3        iStudent - xVar4

iGrade - b        iEnrl - id4        iStudent - idPaulGreen
********* Parse Done ***************
********* Evaluating ***************
iGrade - xVar2        iEnrl - xVar3        iStudent - xVar4

iGrade - b        iEnrl - id4        iStudent - idPaulGreen



[who,took,a,course,that,the,chairman,of,math,taught]
********* Parse Done ***************
********* Evaluating ***************
iEnrl - xVar1        iDep - xVar10        iStudent - xVar2

iEnrl - id2        iDep - math        iStudent - idMilanMosny
```

```
[who,is,the,instructor,for,cmpt720]
********* Parse Done ***************
********* Evaluating ***************
iFaculty - xVar1        iFaculty - xVar2

iFaculty - idCharlieBrown        iFaculty - idCharlieBrown
```

5.4 Related Work

5.4.1 Comparison with CLE

There are three main differences to the approach adopted in [2]. The first difference is a consequence of the two different logical languages used for representing natural language utterances. CLE [2] has a richer language, where lexical predicates can take formulas as their arguments. Therefore it makes sense to impose sortal constraints not only for variables of logical representations but for predicates and other logical language constructors as well. On the other hand, the logical language supposed in our system is a simple extension of first order logic, where input atomic formulas cannot serve as arguments for other input atomic formulas. Therefore, we impose sortal constraints only on arguments of input predicates (i.e. variables).

Our approach also differs in the way the selectional restrictions are derived. The main idea behind CLEs sortal restrictions derivations can be shown on example taken from [2].

A meaning postulate of the form

$$\forall X, Y S(X,Y) \leftarrow R(X,Y)$$

where S is a lexical predicate and R is a relation in the lexicon, can be used to place sortal restrictions (specified for the arguments of S) on all domain objects for which R is known to hold.

On the other hand, our approach places selectional restrictions also on predicates whose arguments do not directly correspond to the variables of application domain. Although CLE allows automatic derivation of selectional restrictions using knowledge about the domain encoded using logic, the sort hierarchy has to be supplied by the designer of the system. Also the representative part of the application domain relations has to have its sortal constraints known.

The third difference touches the type of constraints imposed. Our approach supposes two disjoint sets of argument positions and constrains only the interactions between elements from different sets. CLEs sortal restrictions constrain interactions between any two argument positions.

5.4.2 Constructing Selectional Restrictions from Corpora

There are methods to extract selectional restrictions automatically or semiautomatically given a set of sentences from the domain. We shall describe two of them that in our opinion are representative of the research done in this area, and compare them to our approach.

Andry et al. [3] uses an untagged corpus and a semiautomatic method. Their idea is to assign initial sorts to some argument positions of some predicates. Then the corpus is parsed and due to the unification of variables of different predicates, new sortal restrictions can be derived.

As an example consider the sentence

*Does any student take cmpt*720?

and its logical representation

$$\exists X, Y, E.student(X) \wedge cmpt720(Y) \wedge take(E, X, Y)$$

and its initial sorts assignment

$$\begin{aligned} student(X) &\rightarrow X \text{ is of the sort "student"} \\ cmpt720(X) &\rightarrow X \text{ is of the sort "course"} \\ take(E, X, Y) &\rightarrow E \text{ is of the sort "event"} \end{aligned}$$

From the logical representation of the sentence, one can derive that one of the uses of predicate *take* can yield the following sorts

$take(E, X, Y)$ → E is of the sort "event"
X is of the sort "student"
Y is of the sort "course"

This method is considered semiautomatic since the parser, given a sentence, can produce a parse that does not correspond to the correct semantic constraints. Therefore incorrect sortal restrictions can also be produced. The user of the tool has to perform manual editing of the derived sorts. After the editing is done, the corpus can be parsed again and newly derived sortal restriction edited etc.

The second method [24] uses a phrasally analyzed corpora and a wide coverage noun taxonomy. The method utilizes statistical measures to determine the appropriate classes of nouns that are used for selectional restrictions.

The main difference between our approach and the above methods is the information source used. Our approach uses the description of semantic part of an NLID which must always be present. On the other hand the above mentioned methods need reasonably sized corpora. Such sets of sentences may not always be available.

Also the degree of automatization is higher for our approach. While [3] is a semiautomatic method itself, [24] needs phrasally analyzed corpora. If such corpora are not present, the task of creating one can be very labour intensive.

On the other hand, sortal restrictions produced by [3] are similar to those used in CLE, therefore similarly to the CLE method, interaction between any argument positions can be restricted.

5.5 Summary

In this chapter we introduced algorithms that are able to construct selectional restrictions using normalized LDT as their input. Since the normalization process has been proven to be sound and complete, the selectional restrictions constructed by the algorithms truthfully reflect the semantic constraints imposed by declarative description of the domain in the form of restricted LDT. We have also compared our algorithms with other approaches to the construction of selectional restrictions.

Chapter 6

Conclusions and Future Work

6.1 Summary

The main goal of this thesis has been to develop preprocessing algorithms, namely, an algorithm for construction of semantic restrictions that can be used during the syntactic processing and the algorithm that can use declarative description of the semantic part of an NLID for reasonably efficient semantic processing.

In achieving this goal, we have introduced restricted LDTs for describing the relationship between natural language and the database. We have developed a sound and complete normalization process that is able to produce a normalized form of the restricted LDTs - normalized LDTs. The normalized LDTs are then used as input of an algorithm for automatic construction of selectional restrictions. An algorithm and heuristics that creates actual data-structures useful for semantic processing have been introduced. The algorithm is again based on normalized LDTs. A simple prototype of a natural language interface has been implemented to illustrate the algorithms described in this thesis.

Advantages of this approach include:

- the syntactic part of the NLID can be reused in different domains and automatically adapted to perform as a domain tailored parser,
- greater degree of modularity of the system. A more modular design results in decreased complexity, a decrease in the number of possible errors in the system, possibility to develop different parts of the system concurrently by different persons therefore also decreasing development time

- due to the soundness and completeness of the normalization process, the designer of the interface has a possibility to express the semantic knowledge in more declarative terms

We believe that the approach we have taken is one step in solving the problem of trade-off between performance, modularity and portability of the NLID systems.

6.2 Future Directions

6.2.1 Real Domain

The system was tested on a "toy" university domain where concepts like faculty, students, courses, departments etc. were introduced. Although the systems performance was good, the need to test the system on a real domain is obvious.

The system has a capability of extracting precise semantic constraints that can be used for disambiguation during parsing automatically from the semantic description of the database domain. It is also capable to handle nonmonotonicity easily which makes it suitable for e.g. interfaces to statistical databases similar to the one described in [6].

6.2.2 Normalization Process and Restriction of RLDTs

The way to normalize an LDT that was suggested in this thesis is certainly not the only possibility. There are at least two directions in which the normalization process can be improved so that the constraints imposed on the form of axioms in RLDTs become less restrictive. The first one deals with the nonrecursivity of the LDTs, second one is concerned with existential equivalences that are used in LDTs but are not allowed in RLDTs.

The nonrecursivity constraint was imposed on RLDTs in order to be able to specify completeness conditions precisely. We have two possibilities for how to explore the relaxation of this constraint. The nonrecursivity condition can be substituted by any other constraint that will guarantee the same results (e.g. we shall allow the theories that are "complete" within a constant number of recursive calls). The second possibility is to explore suitable heuristics for dealing with infinite branches that do not guarantee completeness but still offer attractive features w.r.t. practical applications of the system.

A modification to the normalization algorithm that allows one to use the same LDTs as the original process might explored. The idea is based on a closer imitation of the

AET process. Instead of having pairs of input and output atomic formulas that direct the translation, we can simply explore all the conditions (using the algorithm presented in this thesis) needed to translate each lexical atomic formula into a database (possibly nonatomic) formula required by the AET algorithm.

6.2.3 Search Graph Modifications

One of the disadvantages of the search graph data structure and NLDT in general is its size. The problem is caused by repetition of the same subformulas within the condition part of CEs in NLDT. In the future, we would like to decrease the size of the generated search tree. One possible way to attack this problem is to assume two levels of translation. Each level of translation will have its own search graph associated with it. The first level will translate the most frequent subformulas into single tokens. The second level will then use these tokens wherever the frequently occurring subformulas are required. Also translation models having more than two levels are probably possible. We believe that substituting frequently occurring subformulas by single tokens can reduce the space significantly.

6.2.4 Encoding the Selectional Restrictions using Unification

The main problem with the current version of encoding the selectional restrictions for unification based grammars is the size of the terms. Although the number of features is considerably reduced in comparison with the number of indices and attributes, the number of features can still be large.

In the future, we would like to give the user of the system a possibility to constrain the number of features used. The system would then merge similar classes of indices or attributes in order to keep the number of features within the limit.

Bibliography

[1] Alfred V. Aho and Jeffrey D. Ullman. *Principles of compiler design.* Addison - Wesley Pub., Reading, Mass., 1977.

[2] Hyan Alshawi. *The Core Language Engine.* The MIT Press, Cambridge, Massachusetts, 1992.

[3] Francois Andry, Mark Gawron, John Dowding, and Robert Moore. A tool for collecting domain dependent sortal constraints from corpora. CMP-LG E-Print Archive version of the paper published in Proceedings of COLING '94, January 1995.

[4] W.J.H.J. Bronneberg, H.C. Bunt, S.P.J. Landsbergen, R.J.H. Scha, W.J. Schoenmakers, and E.P.C. van Utteren. The question answering system PHLIQA1. In L. Bolc, editor, *Natural Language Question Answering Systems,* Natural Communication with Computers, 1980.

[5] Bob Carpenter and Gerald Penn. *ALE The Attribute Logic Engine User's Guide.* Carnegie Mellon University, Pittsburgh, PA, August 1994. Version 2.0.

[6] Nick Cercone, Paul McFetridge, Fred Popowich, Dan Fass, Chris Groeneboer, and Gary Hall. The SystemX natural language interface: Design, implementation and evaluation. Technical Report CSS-IS TR 93-03, Simon Fraser University, Burnaby, B.C., November 1993.

[7] Michael A. Covington. *Natural language processing for Prolog programmers.* Prentice Hall, Englewood Cliffs, N.J., 1994.

[8] Subrata Kumar Das. *Deductive databases and logic programming.* Addison - Wesley, Reading, Mass., 1992.

[9] Carole D. Hafner and Kurt Godden. Portability of syntax and semantics in Datalog. *ACM Transactions on Office Information Systems*, 3(2):141–164, April 1985.

[10] Gary G. Hendrix, Earl D. Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum. Developing a natural language interface to complex data. *ACM Transactions on Database Systems*, 3(2):105–147, June 1978.

[11] J.R. Hobbs, M.E. Stickel, D.E. Appelt, and P. Martin. Interpretation as abduction. *Artificial Intelligence*, 63:69–142, 1993.

[12] Richard Lederer. *Anguished English : [an anthology of accidental assaults upon our language].* Dell Pub., New York, N.Y., 1989.

[13] Douglas B. Lenat. CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, November 1995.

[14] J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Germany, second edition, 1987.

[15] M.C. McCord. Natural language processing in prolog. In A. Walker, editor, *Knowledge Systems and Prolog*, 1987.

[16] C. S. Mellish. Implementing systemic classification by unification. *Computational Linguistics*, 14(1):40 – 51, 1988.

[17] Milan Mosny. Semantic information preprocessing for natural language interfaces to databases. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 314–316 (student session), June 1995.

[18] Richard A. O'Keefe. *The craft of Prolog.* MIT Press, Cambridge, Mass., 1990.

[19] F.C.N. Pereira and S.M. Shieber. *Prolog and Natural-Language Understanding.* CLSI Lecture Notes. 1985.

[20] C.R. Perrault and B.J. Grosz. Natural language interfaces. In *Exploring Artificial Intelligence*, Survey Talks from the National Conferences on Artificial Intelligence, 1988.

[21] Carl Pollard and Ivan A. Sag. *Head-driven phrase structure grammar.* University of Chicago Press, Chicago, 1994.

[22] S. G. Pulman, H. Alshawi, D.Carter, R. Crouch, M. Rayner, and A. Smith. CLARE: a combined language and reasoning engine. Technical Report CRC-042, SRI International, Cambridge, Mass., 1993.

[23] Manny Rayner. *Abductive Equivalential Translation and its application to Natural Language Database Interfacing.* Ph.D. thesis, Royal Institute of Technology, Stockholm, September 1993.

[24] Francesc Ribas. On learning more appropriate selectional restrictions. In *Proceedings of the 7th Conference of the Europian Chapter of the Association of Computational Linguisics*, pages 112–118, March 1995.

[25] D. G. Stallard. A terminological simplification transformation for natural language question-answering systems. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, pages 241–246, June 1986.

[26] L. Sterling and S. Shapiro. *The Art of Prolog.* Addison-Wesley, Reading, MA, 1985.

[27] David L. Waltz. An english language question answering system for a large relational database. *Communications of the ACM*, 21(7):526–539, July 1978.

Appendix A

Definition A.1 (Definite program clause) *A* definite program clause *is a clause of the form*

$$A \leftarrow B_1, \cdots, B_n$$

which contains precisely one atom (viz.A) in its consequent. A is called the head *and* $B_1, \cdots, B_n$ *is called the* body *of the program clause.*

Definition A.2 (Definite program) *A* definite program *is a finite set of definite program clauses.*

Definition A.3 (Definite goal) *A* definite goal *is a clause of the form*

$$\leftarrow B_1, \cdots, B_n$$

that is, a clause which has an empty consequent. Each B_i *is called a* subgoal *of the goal.*

Definition A.4 (Substitution) *A* substitution f *is a finite set of the form* $\{v_1/t_1, \cdots, v_n/t_n\}$, *where each* v_i *is a variable, each* t_i *is a term distinct from* v_i *and the variables* $v_1, \cdots, v_n$ *are distinct. Each element* v_i/t_i *is called a* binding *for* v_i. f *is called a* ground substitution *if the* t_i*s are all ground terms.* f *is called a* freezing substitution *if* t_i*s are unique constants.*

Definition A.5 (Variant) *Let* E *and* F *be expressions. We say* E *and* F *are* variants *if there exist substitutions* f *and* g *such that* $E = Ff$ *and* $F = Eg$. *We also say* E *is a variant of* F *or* F *is a variant of* E.

Definition A.6 (Most general unifier (m.g.u.)) *Let* S *be a finite set of simple expressions. A substitution* f *is called a* unifier *for* S *if* Sf *is a singleton. A unifier* f *for* S *is called a* most general unifier (m.g.u.) *for* S *if, for each unifier* g *of* S, *there exists a substitution* h *such that* $g = fh$.

Definition A.7 (SLD-derivation) *Let P be a definite program and G a definite goal. An* SLD-derivation *of $P \cup \{G\}$ consists of a (finite or infinite) sequence $G_0 = G, G_1, \cdots$ of goals, a sequence $C_1, C_2, \cdots$ of variants of program clauses of P and a sequence $f_1, f_2, \cdots$ of m.g.u.s such that each G_{i+1} is derived from G_i and C_{i+1} using f_{i+1}.*

Definition A.8 (SLD-refutation) *An* SLD-refutation *of $P \cup \{G\}$ is a finite SLD-derivation of $P \cup \{G\}$ which has the empty clause $\square$ as the last goal in the derivation. If $G_n = \square$, we say the refutation has* length n.

Definition A.9 (Computation rule) *A* computation rule *is a function from a set of definite goals to a set of atoms such that the value of the function for a goal is an atom, called the* selected *atom, in that goal.*

Definition A.10 (SLD-refutation via R) *Let P be a definite program, G a definite goal and R a computation rule. An* SLD-refutation *of $P \cup \{G\}$* via *R is an SLD-refutation of $P \cup \{G\}$ in which the computation rule R is used to select atoms.*

Definition A.11 (Unrestricted SLD-refutation via R) *An* unrestricted SLD-refutation via *R is an SLD-refutation via R, except that we drop the requirement that the substitutions f_i be m.g.u.s. They are required to be unifiers.*

We suppose that a definite program does not contain two clauses that are variants of each other.

We shall write $origin(C_i)$ to denote program clause C s.t. C_i is a variant of C. Under our above mentioned assumption about definite programs, it is obvious, that $origin(C_i)$ is a function.

Similarly $origin(CC)$, where $CC = \{C_1 \cdots C_n\}$ denotes $\{origin(C_1) \cdots origin(C_n)\}$.

Theorem A.1 *The conditions construction algorithm always terminates.*

Proof. *Step 1* terminates, because set T is finite and the set Pr is finite. Therefore, the set T' is finite.

It is obvious that given a finite set of definite program clauses, there are only a finite number of SLD-derivations without recursive calls. Therefore the part of the SLD-tree where SLD-derivations with recursive calls are disregarded is finite. Thus *Step 2* terminates.

This SLD-tree has also only finite number of branches. So *Step 3* terminates as well. □

Soundness of the conditions construction algorithm trivially follows from the next theorem.

Theorem A.2 *Let G be a definite goal $\leftarrow A_1 \cdots A_l$, T be a definite program and G; $G_1 \cdots G_n$; $C_1 \cdots C_n$; $f_1 \cdots f_n$ be a SLD-refutation of G in T. Let $PP = \{P_1 \cdots P_m\}$ be an arbitrary subset of facts of $CC = \{C_1 \cdots C_n\}$ s.t. $origin(CC) - origin(PP) = origin(CC - PP)$. (In other words, either all or none of the C_is that come from the same origin are members of PP). Then*

$$T - \{origin(P_1) \cdots origin(P_m)\} \models$$
$$P_1 f_1 \cdots f_n \wedge \cdots \wedge P_m f_1 \cdots f_n \rightarrow A_1 \; cdots A_l f_1 \cdots f_n$$

Proof. We shall prove this theorem by induction on the length of the SLD-refutation.

If we have a length 1 SLD-refutation of G in T, obviously G is of the form $\leftarrow A_1$ and C_1 is a fact and f_1 is a m.g.u. of A_1 and C_1. Then there are two cases.

Case 1: $(m = 0)$ C_1 is not a member of $\{P_1 \cdots P_m\}$ (m is the number of Ps. Ps are selected $C_i s$).

In this case it is obvious that

$$T - \{\} \models C_1$$

because $origin(C_1)$ is a member of T and $C_1 = origin(C_1)f$ for some renaming substitution. Therefore also

$$T - \{\} \models C_1 f_1$$

Because f_1 is a m.g.u. of C_1 and A_1

$$T - \{\} \models A_1 f_1$$

Case 2: $(m = 1)$ In this case $P_1 = C_1$. Clearly $C_1 f_1 = A_1 f_1$, because f_1 is a m.g.u. of C_1 and A_1, therefore also $P_1 f_1 = A_1 f_1$. Thus for any theory T''

$$T'' \models P_1 f_1 \rightarrow A_1 f_1$$

so

$$T - \{origin(P1)\} \models P_1 f_1 \rightarrow A_1 f_1$$

Suppose that the theorem holds for all SLD-refutations of length $n-1$. There are two cases.

Case 1: (C_1 is not a member of $\{P_1 \cdots P_m\}$), i.e. $origin(C_1)$ is not a member of $origin(P_1 \cdots P_m)$ so

$$T - \{origin(P_1) \cdots origin(P_m)\}) \models C_1$$

Suppose that G_1 is of the form $\leftarrow AF_1$. By the inductive hypothesis

$$T - \{origin(P_1) \cdots origin(P_m)\} \models$$
$$P_1 f_2 \cdots f_n \wedge \cdots \wedge P_m f_2 \cdots f_n \rightarrow AF_1 f_2 \cdots f_n$$

By definition of SLD-derivation, G is of the form

$$A_1 \cdots A_l$$

where $A_1 \cdots A_l$ are atomic formulas, AF_1 is of the form

$$A_1 \cdots A_{m-1} B_1 \cdots B_q A_{m+1} \cdots A_l f_1$$

and C_1 is of the form $A \leftarrow B_1 \cdots B_q$, where f_1 is a m.g.u. of A and A_m. Obviously

$$T - \{origin(P_1) \cdots origin(P_m)\} \models (\forall) A \leftarrow B_1 \cdots B_q$$

and therefore

$$T - \{origin(P_1) \cdots origin(P_m)\} \models (\forall) A f_1 \cdots f_n \leftarrow B_1 \cdots B_q f_1 \cdots f_n$$

Also

$$T - \{origin(P_1) \cdots origin(P_m)\} \models$$
$$(\forall) A_m f_1 \cdots f_n \leftarrow B_1 \cdots B_q f_1 \cdots f_n \qquad \text{(A.1)}$$

because f_1 is a m.g.u. of A and A_m.

From the inductive hypothesis and from A.1, it follows that

$$T - \{origin(P_1) \cdots origin(P_m)\} \models$$
$$(\forall) P_1 f_2 \cdots f_n \cdots P_m f_2 \cdots f_n \rightarrow A_1 \cdots A_l f_1 \cdots f_n$$

$P_1 \cdots P_m$ are variants with unique variables, therefore they do not contain variables present in formulas $A_1 \cdots A_l$ and C_1. So

$$P_i = P_i f_1 \quad \text{for } \forall i \text{ s.t. } 1 \leq i \leq m.$$

Thus

$$T - \{origin(P_1) \cdots origin(P_m)\} \models$$
$$P_1 f_1 \cdots f_n \cdots P_m f_1 \cdots f_n \rightarrow A_1 \cdots A_l f_1 \cdots f_n.$$

Case 2: (C_1 is a member of $\{P_1 \cdots P_m\}$)

WLOG C_1 is of the form P_1, G is of the form $\leftarrow A_1 \cdots A_l$ and G_1 is of the form

$$G = \leftarrow AF_1 = A_1 \cdots A_{m-1} A_{m+1} \cdots A_l f_1$$

where f_1 is a m.g.u. of P_1 and A_m, so C_1 is a fact.

By the inductive hypothesis

$$T - \{origin(P_2) \cdots origin(P_m)\} \models$$
$$P_2 f_2 \cdots f_n \cdots P_m f_2 \cdots f_n \rightarrow AF_1 f_2 \cdots f_n$$

There are two cases. Either there is an i s.t. $2 \leq i \leq n$ and $origin(C_i) = origin(C_1)$ or there is not.

In the first case according to the assumptions of the theorem, C_i is a member of $\{P_2 \cdots P_m\}$ so

$$\{origin(P_2) \cdots origin(P_m)\} = \{origin(P_1) \cdots origin(P_m)\}$$

Thus

$$T - \{origin(P_1) \cdots origin(P_m)\} \models$$
$$P_2 f_2 \cdots f_n \cdots P_m f_2 \cdots f_n \rightarrow AF_1 f_2 \cdots f_n$$

In the second case $origin(C_1) = origin(P_1)$ was not used in the SLD-derivation of $\leftarrow AF_1$, i.e. there exists the same SLD-derivation of $\leftarrow AF_1$ in $T - \{origin(C_1)\}$ and according to the inductive hypothesis also

$$T - \{origin(P_1) \cdots origin(P_m)\} \models$$
$$P_2 f_2 \cdots f_n \cdots P_m f_2 \cdots f_n \rightarrow AF_1 f_2 \cdots f_n$$

Thus in any case

$$T - \{origin(P_1) \cdots origin(P_m)\} \models$$
$$P_2 f_2 \cdots f_n \cdots P_m f_2 \cdots f_n \rightarrow A_1 \cdots A_{m-1} A_{m+1} \cdots A_l f_1 f_2 \cdots f_n$$

Obviously $P_1 f_1 \cdots f_n \rightarrow A_m f_1 \cdots f_n$, because f_1 is a m.g.u. of the P_1 and A_m. So

$$T - \{origin(P_1) \cdots origin(P_m)\} \models$$
$$P_1 f_1 \cdots f_n P_2 f_2 \cdots f_n \cdots P_m f_2 \cdots f_n \rightarrow A_1 \cdots A_l f_1 \cdots f_n$$

By an argument similar to that used for case 1

$$T - \{origin(P_1) \cdots origin(P_m)\} \models$$
$$P_1 f_1 \cdots f_n P_2 f_1 \cdots f_n \cdots P_m f_1 \cdots f_n \rightarrow A_1 \cdots A_l f_1 \cdots f_n$$

□

Theorem A.3 (Completeness of the conditions construction algorithm) *Let T be a nonrecursive theory. Let Q is a ground atomic formula. Let $P_1 \cdots P_m$ be variants of predicates $P_1 \cdots P_m$ from I and f a substitution s.t.*

$$T \models (\forall) P_1 f \cdots P_m f \rightarrow Q f$$

then there are substitutions g and h and subsequence j of the sequence $1 \cdots m$ s.t. formula

$$P_{j_1} g \cdots P_{j_k} g \rightarrow Q g$$

belongs to the result of the conditions construction algorithm and

$$gh = f$$

for variables of the variants $P_1 \cdots P_m$.

Proof. We shall write h^* to denote substitution h narrowed to the variables of variants $P_1 \cdots P_m$.

Let e be a freezing substitution, which substitutes all occurrences of the free variables in the formula $P_1 f \cdots P_m f \rightarrow Q f$ by unique constants. Then

$$T \models P_1 fe \cdots P_m fe \rightarrow Q fe$$

From the deduction theorem we have

$$T \cup \{P_1 fe \cdots P_m fe\} \models Q fe$$

So Q is a logical consequence of $T \cup \{P_1 fe \cdots P_m fe\}$. By strong completeness of SLD-resolution [14] and by nonrecursivity of the theory T, there exists an SLD-refutation via R of $T \cup \{P_1 fe \cdots P_m fe\} \cup \{\leftarrow Q\}$ without recursive calls with substitutions $a_1 \cdots a_n$, clauses $C_1 \cdots C_n$ and goals $\leftarrow Q, G_1 \cdots G_n$.

Let us put

$$a'_i = \begin{cases} a_i & \text{if } origin(C_i) \text{ belongs to } T \\ fea_i & \text{if } origin(C_i) \text{ belongs to } \{P_1 fe \cdots P_m fe\} \end{cases}$$

$$C'_i = \begin{cases} C_i & \text{if } C_i \text{ belongs to } T \\ P_{j_l}, \text{where} C_i = P_{j_l} fe & \text{otherwise} \end{cases}$$

(here j_l is the l-th element of sequence of numbers with range $1 \cdots m$. It is used to project $P_1 \cdots P_m$ into the set $P_{j_1} \cdots P_{j_k}$ of variants of predicates, that were actually used in the $C_1 \cdots C_n$)

It is easy to see that $\leftarrow Q, G_1 \cdots G_n; C'_1 \cdots C'_n; a'_1 \cdots a'_n$ is an unrestricted SLD-refutation via R of $\leftarrow Q$ in $T \cup \{P_1 \cdots P_m\}$ without a recursive call.

From the proof of the m.g.u. lemma in [14], there is an SLD-refutation via R of $\leftarrow Q$ without recursive calls with substitutions $b_1 \cdots b_n$, which follows the steps of the unrestricted refutation S'' by using the same clauses and choosing the same literals to be resolved, but substitutions $b_1 \cdots b_n$ are m.g.u.s. Moreover, there exists a substitution h' s.t. $a'_1 \cdots a'_n = b_1 \cdots b_n h'$. This refutation is a part of constructed SLD-tree, therefore

$$P_{j_1}(b_1 \cdots b_n)^* \cdots P_{j_k}(b_1 \cdots b_n)^* \rightarrow Q fe(b_1 \cdots b_n)^* = Q$$

belongs to the result set S of conditions construction algorithm .

It follows that

$$\text{for } \forall i \text{ from } 1 \leq i \leq n, \quad fea_i = a_i fe$$

There are no variables from P_i mentioned in the $\leftarrow Q, G_1 \cdots G_n$ because fe binds them to constants for all $P_i s$. So a_i works only on the variables of $\leftarrow Q, G_1 \cdots G_n$ and substitutions can be easily switched.

Also

$$fefe = fe$$

because e is a freezing substitution. Thus $a'_1 \cdots a'_n = a_1 \cdots a_n.fe$, Therefore, there is a substitution h' s.t.

$$a_1 \cdots a_n fe = b_1 \cdots b_n h'$$

If we consider only terms with variables occurring in the variants $P_1 \cdots P_m$ we obtain

$$fe = (b_1 \cdots b_n)^* h'$$

Then considering the fact that e is a freezing substitution, i.e. there is a "thawing" mapping that maps unique constants back to the original variables, and the fact that substitutions $f, b_1 \cdots b_n$ do not contain freezing constants, we can conclude that there is a substitution h s.t.

$$f = gh \text{ , where } g = (b_1 \cdots b_n)^*$$

for all variables from the variants $P_1 \cdots P_m$.

By setting $(b_1 \cdots b_n)^*$ for g we have proven the existence of substitutions g, h and subsequence j of $1 \cdots m$ s.t. $f = gh$ and $P_{j_1} g \cdots P_{j_k} g \rightarrow Qg$ belongs to the result of the algorithm. □

Theorem A.4 *Conditions combination algorithm terminates for any input.*

Proof. Termination of *step 1* and *step 2* of the conditions combination algorithm follows from the termination of the conditions construction algorithm.

Step 3 terminates, because results SI and SO are finite and the set $Pairs$ is finite, and *step 4* obviously always terminates. □

Soundness of the conditions combination algorithm trivially follows from the following theorem.

Theorem A.5 *For each pair (I, O) and for each condition CC constructed for the pair by the condition combination algorithm*

$$T \models CC \rightarrow (I \equiv O)$$

Proof. All results of the algorithm are constructed in *step 4*. The proof follows the substeps of *step 4*. Due to the soundness of the conditions construction algorithm, we have

$$T \models (\forall)\ R_1 \cdots R_m \rightarrow O$$

Because O is a ground atomic formula,

$$T \models (\forall)\ R_1 \cdots R_m r \rightarrow O$$

for any substitution r.

Since all members of the set SR are unified with I using m.g.u. r and I and O are ground formulas, it follows that

$$T \models (\forall) RR'r \wedge I \rightarrow O$$

so

$$T \models (\forall) RR'r \rightarrow (I \rightarrow O)$$

Similarly

$$T \models (\forall) QQ'q \rightarrow (O \rightarrow I)$$

Also note, that QQ' are assumption predicates. Substitution r substitutes some of the variables of RR' with constant terms. Substitution q substitutes some of the variables of QQ' with constant terms. Note that variables of RR' are different from variables of QQ'. Therefore

$$T \models (\forall)\ RR'rq \rightarrow (I \rightarrow O)$$

$$T \models (\forall)\ QQ'rq \rightarrow (O \rightarrow I)$$

So

$$T \models (\forall)\ (RR' \wedge QQ')rq \rightarrow (I \equiv O)$$

□

Theorem A.6 (Completeness of the conditions combination algorithm)

Proof.

Lemma A.1 *Let $P_1 \cdots P_n$, $Q_1 \cdots Q_m$ be atomic formulas and $X_1 \cdots X_k$ be all the variables that occur in $P_1 \cdots P_n$, $Q_1 \cdots Q_m$*

$$(\forall)\ P_1 \cdots P_n \rightarrow Q_1 \cdots Q_m \tag{A.2}$$

is a theorem iff $\{Q_1 \cdots Q_m\}$ is a subset of $\{P_1 \cdots P_n\}$

Proof.

If direction is trivial.

Proof of *only if* direction by contradiction.

WLOG suppose that Q_1 is not a member of $\{P_1 \cdots P_n\}$. Then consider a Herbrand interpretation where the domain contains "$X_1'' \cdots$"X_k'' and all the constants, that occur in A.2 plus all possible function applications. Let "Q_1'' in this interpretation be *false*, and everything else be *true*. Then $P_1 \cdots P_n$ will be *true* under each assignment of variables, because Q_1 does not occur in $P_1 \cdots P_n$, but $Q_1 \cdots Q_m$ will be *false* under assignment $X_1 =$ "$X_1'' \cdots X_k =$ "X_k''. From the completeness of FOL follows that (A.2) is not a theorem. □

Consider quadruple (PP', AA', I, O). For the notational convenience we shall write CC' to denote $PP' \wedge AA'$.

We shall show that if (PP', AA', I, O) is an NCE, then our algorithm will construct a condition RR' that is weaker than CC', and the algorithm will at the same time construct condition QQ', that is also weaker than CC'. Then we shall show that the union of constructed conditions CC is weaker than CC'.

Suppose that (PP', AA', I, O) is an NCE. Then from the definition of NCE

$$T \models (\forall)\ CC' \wedge I \rightarrow O$$

From the completeness of the conditions construction algorithm, there is a condition $R_1 \cdots R_m$ in the set $S0$ in the algorithm, s.t.

$$T \models (\forall)\ R_1 \cdots R_m \rightarrow O \tag{A.3}$$

and there is a substitution f s.t.

$$(\forall)\ CC' \wedge I \rightarrow R_1 f \cdots R_m f \tag{A.4}$$

It is obvious that at least one of the formulas in $R_1 f \cdots R_m f$ is I. If it is not the case then from the lemma it follows that

$$(\forall)\ CC' \rightarrow R_1 f \cdots R_m f \tag{A.5}$$

From A.3 and A.5, we obtain

$$T \models (\forall)\ R_1 f \cdots R_m f \rightarrow O \tag{A.6}$$

and from A.5 and A.6

$$T \models (\forall)\ CC' \rightarrow O$$

which is a contradiction with the fact that (PP', AA', I, O) is an NCE.

So there exists a nonempty subset SR of the formulas $R_1 \cdots R_m$ that contains all the formulas R_i, s.t. $R_i f = I$. So according to the lemma A.1 and from A.4

$$(\forall)\ CC' \rightarrow (\{R_1 \cdots R_m\} - SR)f$$

Let us consider the most general unifier r of the set SR and I. Then there is a substitution h s.t. $f = rh$ and

$$(\forall)\ CC' \rightarrow (\{R_1 r \cdots R_m r\} - SRr)h$$

Note, that all possible sets SR are considered in the *step 4* of the algorithm. Therefore there is a set RR' and substitution r considered in the algorithm, s.t. there exists substitution h s.t.

$$(\forall)\ CC' \rightarrow RR'rh \tag{A.7}$$

Similarly, from the definition of NCE we have

$$T \models (\forall) AA' \wedge O \rightarrow I$$

From the completeness of the conditions construction algorithm, there is a condition $Q_1 \cdots Q_l$ in the set SO in the algorithm, s.t.

$$T \models (\forall)\ Q_1 \cdots Q_l \rightarrow I$$

and there is a substitution g s.t.

$$(\forall)\ AA' \wedge O \rightarrow Q_1 g \cdots Q_l g \tag{A.8}$$

From A.8, from the lemma A.1 and from the fact, that AA' contains only assumption predicates, it follows that there is a subset (possibly empty) SQ of $\{Q_1 \cdots Q_l\}$ that contains all Q_is s.t. the predicate of Q_i is the same as the predicate of O and g is a unifier of SQ and O.

Therefore there exists a m.g.u. q of SQ and O and a substitution h' s.t. $g = qh'$. From lemma A.1 and from A.8, it clearly follows, that

$$AA' \rightarrow QQ'qh' \tag{A.9}$$

So from A.7 and A.9 we have

$$(\forall)\ CC \rightarrow RR'rh \wedge QQ'qh'$$

WLOG we can assume that the three sets of variables that occur in CC', QQ' and RR' are mutually disjunctive. Also substitutions r and q bind variables with constant terms, because formulas I and O are ground. Thus we can write

$$(\forall)\ CC \rightarrow (QQ' \cup RR')rqhh'$$

So there is a CE in the results of the conditions combination algorithm that fulfills the completeness requirement. □