A FLEXIBLE AMERICAN SIGN LANGUAGE INTERFACE TO DEDUCTIVE DATABASES

by

Eli Hagen

B.Sc. Queen's University, Kingston, Ontario, 1991

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
December 1993

ISBN 0-612-00967-X

APPROVAL

Name: Eli Hagen

Degree: Master of Science

Title of thesis: A Flexible American Sign Language Interface to Deductive Databases

Examining Committee:

Dr. Slawomir Pilarski
Professor, Computing Science
Simon Fraser University
Chair

Dr. Veronica Dahl
Professor, Computing Science
Simon Fraser University
Senior Supervisor

Dr. Frederick Popowich
Assistant Professor, Computing Science
Simon Fraser University
Supervisor

Dr. Paul Tarau
Professeur Agregé, Département d'Informatique
Université de Moncton
External Examiner

Date Approved: December 7th, 1993

PARTIAL COPYRIGHT LICENSE

Title of Thesis/Project/Extended Essay

A Flexible American Sign Language Interface to Deductive Databases.

Author: ________________

(signature)

Eli Hagen

(name)

16 Dec 1993

(date)

Abstract

This thesis investigates interfaces to deductive databases in order to allow deaf people easy consultation using their native language (in particular, American Sign Language), and develops one specific approach to the problem. Our approach consists of developing a multiple-valued logic system which serves both as the internal representation of American Sign Language (ASL) and as the database consultation language. We exemplify our ideas with a concrete system that assumes preprocessing of visual images translating them into a written form. Our system then translates this written form into a rigorously defined logical system with multiple truth values which allow richness of expression. For instance, this system can differentiate between different kinds of plural and detect failed presuppositions. The translation in terms of our logical system can then be used directly to consult deductive databases. While our exemplifying focus is database consultation in ASL, it is clear that the logical system developed around this application has other possible uses in natural language processing in general, and database interfacing in particular.

Acknowledgement

I would like to thank my senior supervisor Veronica Dahl for her guidance, encouragement, and support throughout my studies. Thanks to the other member of my committee Fred Popowich for most helpful discussions and comments on the thesis. Financial support from Veronica Dahl's NSERC grant #31-611024, the School of Computing Science, and Simon Fraser University is gratefully acknowledged. Thanks to the Logic and Functional Programming Laboratory for use of their facilities. My deepest appreciation to my friends for their support and understanding throughout the entire effort. Especially Jörg, AllanBB and Pepe and the "chiquilinas" Diana, Alicja and Brigitte who made my stay at SFU so enjoyable and to Patrice who patiently solved all my UNIX and LaTeX problems. Finally, thanks to my family whom I didn't see for two years.

Contents

Abstract iii

Acknowledgement iv

1 Introduction 1

1.1 Natural Language Interfaces for Manual Languages 1

1.2 Cooperative Answering 2

1.3 Related Work in Cooperative Answering 3

1.3.1 Beliefs and Expectations 4

1.3.2 Presuppositions 5

1.3.3 Misconceptions 5

1.3.4 Intensional Answers 6

1.3.5 Generalizations 6

1.4 The Main Challenges and Our Solutions 7

1.5 Organization of the Thesis 8

2 Natural Language Processing 9

2.1 Colmerauer's Three-Valued Logic 9

2.1.1 Elementary Statements 10

2.1.2 Quantification Hierarchy 10

2.1.3 Negation 14

2.1.4 Conjunction between Statements, Relative Clauses 15

2.1.5 A Single Quantification Mechanism 16

2.1.6 Three Logical Values 18

2.1.7 A Logical System for a Subset of Natural Language 18

2.2 L3-a Natural Language Oriented Database Query Language 19

2.2.1 L3's Quantification Mechanism 20

2.2.2 Safe Negation as Failure 20
2.3 Incomplete Types 21
2.3.1 Intensional Answers 22
2.4 Logic Grammars 23
2.4.1 Definite Clause Grammar 28

3 Multiple Logic Values 29
3.1 Motivation 29
3.1.1 Pointless 29
3.1.2 Absurd 30
3.1.3 Vague 30
3.1.4 Mixed 31
3.1.5 Unknown 31
3.1.6 Inconsistent 32
3.2 Implicit and Explicit Logical Values 33
3.3 Semantic Types 34
3.3.1 Intensional Replies 36
3.3.2 Presupposition Formulae 36

4 A Typed Multiple-Valued Deductive Database 37
4.1 Partitioning of Relations 37
4.2 Rigorous Definition of a Typed Multiple-Valued Logic Database 38
4.2.1 Distributive Relations 41
4.2.2 Inherently Collective Relations 45
4.2.3 Partially Collective Relations 47
4.2.4 Respective Relations 50
4.2.5 Single Attribute Relations 56
4.3 Rigorous Definition of A Multiple-Valued Typed Query Language 58
4.3.1 Syntax 58
4.3.2 Semantics 59

5 American Sign Language 62
5.1 What is a sign? 62
5.2 Economization 63
5.3 Basic Sentence Structures 64
5.3.1 Declarative Sentences 65

5.3.2 Non-Manual Signals; Questions and Negated Sentences 66
5.3.3 Compound Sentences 68
5.3.4 Conditional Sentences 68
5.3.5 Subordination 69
5.3.6 Infinitives and Gerunds 69
5.3.7 Plural, Quantification 69
5.3.8 Dual Inflection 70
5.4 Verbs in ASL 71
5.4.1 Non-Directional Verbs 71
5.4.2 Multi-Directional Verbs 72
5.4.3 Surface Structure in Discourse 73
5.5 Time 74
5.5.1 Time Markers 74
5.5.2 Verb Tenses, Time in the ASL Discourse 75

6 ASL—The Computational Challenge 77

6.1 An ASL Front End 78
6.1.1 Overview 78
6.1.2 The Visual Interface and the Sign Interpreter 80
6.2 The Parser 82
6.2.1 The Parsing Algorithm 82
6.2.2 The Interface between the Lexical and Non-Lexical Parsing Routines 84
6.2.3 Initialization 86
6.3 The Non-Lexical Parsing Routine 86
6.3.1 Input to the Non-Lexical Parsing Routine 87
6.3.2 The Routine 88
6.4 ASL-Grammar 91
6.4.1 Input to the ASL-Grammar 92
6.4.2 Basic Sentences 93
6.4.3 Establishment of Spatial References and Roles 94
6.4.4 Indexing and Role Play 96
6.4.5 Question, Negation, and Topicalization 100
6.4.6 Time 102
6.4.7 Subject Deletion 104
6.5 Translating ASL into LM 107

6.5.1 Elementary Statements 107
6.5.2 Determiners 107
6.5.3 ASL and LM's Quantification Hierarchy 109
6.5.4 Negation 112

7 Conclusion **114**

7.1 Contributions 114
7.2 Limitations and Future Work 115

A ASL Grammar **116**

B ASL Dictionary **120**

C Grammar Subroutines **121**

D Some Sample ASL Queries in LM **124**

Bibliography **125**

Chapter 1

Introduction

This thesis describes research in two areas: cooperative answering in deductive database systems and natural language front ends for manual languages (in particular, American Sign Language) to databases. Below we will motivate our research in each of these areas and review related work.

Our contributions in each of these areas have significance and application independently of those in the other area. We have also combined them into an integrated approach for human, in particular ASL speakers, consultation of deductive databases with cooperative answering.

1.1 Natural Language Interfaces for Manual Languages

Deaf people learn spoken languages as their second language. In addition to facing the usual difficulties related to mastering a second language, they feel that their expressive capacity is severely restricted by the sequential nature of spoken language[1]. These concerns create a significant barrier between deaf people and computer applications and hopefully this research will help reduce these barriers by making computer applications available to deaf people in their native language.

The linguistic structure of sign languages is different from spoken languages and therefore a different type of interface has to be developed. ASL linguists have identified two areas where ASL differs substantialy from spoken languages: 1. ASL's paralell use of non-lexical information and 2. ASL's built-in memory.

Parallelism also exists in spoken languages, e.g., body language (and tone of voice) is

[1] Roger Carver, Director of Programs and Services, Deaf Children's Society of British Columbia. Personal communication, Sept. 1992.

used for emphasis, in irony and satire, but it is the degree to which ASL uses non-lexical information that makes it interesting. ASL uses facial expressions to convey fundamental linguistic concepts like negation and questioning, which one has to account for in a natural language interface to, for example, a database. In spoken languages these concepts are conveyed through lexical items, word order and inflection, so we can detect questions and negation through simple sequential processing of lexical items. We do not have to account for body language since, for example, irony, is rarely part of a query for factual information. However, in, for example, a translation system, body language analysis is interesting also for spoken languages.

ASL uses locations in the signing space to represent people and objects, i.e., the signer establishes a contextual reference somewhere in the space and all later reference to that person/object is done by pointing to the location. This memory aspect of ASL makes the parsing task more complicated since traditinal parsers are not equiped with memeory, but it makes it easier to produce a semantic representation since pronominal references are unique.

Traditionally, research on natural language front ends to computer systems has focused on spoken languages, especially English, and to the best of our knowledge, no-one has attempted a computational model of manual languages. We develop such a model for a subset of ASL, tailored for use as a front end to deductive databases. Although, our specific application is a database, this model is applicable anywhere where an ASL interface is wanted (e.g., expert systems). Our model presupposes a vision system that transforms a signer's manual signs into distinct tokens (names of signs) and that recognizes relevant non-manual behaviour performed by the signer. Work in this area is already underway (see [17]).

1.2 Cooperative Answering

Providing *correct* answers to users is not good enough since such answers may be ambiguous. Also the user may have misconceptions about the database or the world such that a merely correct answer may in fact be more misleading than helpful. As we shall see below, alternative information or extra information may be more useful and less misleading to the user.

We propose a new class of deductive databases that provides a multiple valued semantics as a means of providing natural and cooperative answers to queries. The database is tailored to be used in conjunction with a natural language front end (our front end is the ASL front end discussed above, but our database is of course independent of language). We propose multiple logical values in view of more informative answers, matching the more nuanced input allowed by natural language. For example, through our new logical values, we can detect queries with false presuppositions, queries that are inconsistent with the current state of the knowledge base, and queries that are semantically anomalous. Our system also easily accommodates intensional answers.

Our work extends the L3 formalism developed by V. Dahl into a multi-valued formalism for deductive databases, which we shall name LM. Our line of research makes it possible to use logic throughout the entire database system: as the database definition language, as the database query language, as the data manipulation language, and as the analyser's language. This uniformity of representation minimizes the interfaces between the different components of a natural language consultable database. A preliminary version of the system presented here has been implemented in a prototype system.

Below we briefly review some of the previous work done in this area. The material in this section is mostly collected from [25].

1.3 Related Work in Cooperative Answering

The origin of the field of cooperative query answering can be traced back to work by Joshi and Webber in the mid/late 1970s and a workshop held at the University of Pensylvania in 1978, which resulted in the book "Elements of Discourse Understanding" [30] based on the papers presented there. From this initial work, research has grown out of the following three areas in which question-and-answer discourses arise:

1. Natural Language Interface

2. Databases

3. Logic Programming and Deductive Databases

In the words of Gaasterland et al. [25]: Re. 1: "Some researchers are interested in modeling human conversation. Others hope to use human discourse as a normative standard for information systems' behaviour." (See [29, 35, 49].) Re. 2: "A number of researchers in the area of databases [7, 8, 33, 43] have recognized the practical need for cooperative answering behaviour in standard, widely available information systems." Re. 3: "Logic offers a suitable representation for expressing cooperative answers, whereas SQL and other database languages must be extended to do this. The rules and integrity constraints of deductive databases allow for a rich semantics, which plays a vital role in generating cooperative responses."

The techniques that have grown out of each of the above areas can be separated into the following five categories (from [25]):

1. Consideration of specific information about a user's state of mind
2. Evaluation of presuppositions in a query
3. Detection and correction of misconceptions in a query (other than false presupposition)
4. Formulation of intensional answers
5. Generalization of queries and of responses

We consider each in turn.

1.3.1 Beliefs and Expectations

These techniques are concerned with the question "Can the system interpret the query properly?" For example, natural language queries might need disambiguation to be answered or a query might be asked with different intentions or beliefs.

Webber [49, 31] is concerned with user's beliefs and expectations and how to provide extra information to avoid misconceptions. Consider the following example:

> **Q:** "Is Sam an associate professor?"
> User believes that associate professors have tenure.
> Sam does not have tenure. Sam is an associate professor.
> **A:** "Yes, but he does not have tenure."

Lehnert [35] is concerned with the users intent. Her system, QUALM, parses the query into a conceptual dependency representation. Then it analyses the question and the previous test in the conversation to decide the user's intent. Consider the following question and its three potential answers:

> **Q:** "How did John take the exam?"
> $\mathbf{A_1}$: "He crammed the night before." (enablement)
> $\mathbf{A_2}$: "He took it with a pen." (instrumental/procedural)
> $\mathbf{A_3}$: "He took it badly." (emotional)

1.3.2 Presuppositions

Kaplan [32, 33] states that presuppositions are statements that must be true for the query to have an answer and if a presupposition is false, the query is nonsensical. For example, a student might ask a database

"Who passed CMPT710 in the fall semester of 1991?"

The database answers "No one". The student then asks

"Who failed CMPT710 in the fall semester of 1991?"

Again, the database answers "No one". The student becomes suspicious and asks

"Who taught CMPT710 in the fall semester of 1991?"

and the database answers "No one". Kaplan calls this *stone walling*, i.e., the database answers *yes* or *no* regardless of whether the answer is misleading. If the original question was asked to a human, she would have answered that there was no such course that semester right away.

In the system CO-OP, Kaplan represents queries as a semantic network and the query answering system checks that each connected subgraph is non-empty. An empty subgraph represents a false presupposition that ought to be reported.

Consider the query "Which employee owns a red car?" ($\leftarrow employee(X), owns(X,Y), car(Y), red(Y)$ This query may fail because no employee owns a car at all and reporting this failure is more informative than just answering "No one".

Janas' [28] solution to the problem of failed presuppositions is to report the smallest subquery that fails. If we consider a conjunctive query as a set of atoms, then subqueries are elements of the power set (i.e., there are 2^n subqueries for a conjunctive query with n atoms.) Consider the query "Which employee owns a red car?" ($\leftarrow employee(X), owns(X,Y), car(Y), red(Y)$). This query may fail because no employee owns a car at all and reporting this failure is more informative than just answering "No one". I.e., the subquery $\leftarrow employee(X), owns(X,Y), car(Y)$ is the smallest subquery that would fail.

Colmerauer is also concerned with failed presuppositions, and we will review Colmerauer's work in more detail in section 2.1.

1.3.3 Misconceptions

Misconceptions arise when the user has an unclear understanding of the database's semantics. Consider query

"Which professors took CMPT710?"

with respect to a database in which students *take* courses, while professors *teach* courses. Mays [39] uses the database schema to correct this type of misconception. He introduces aspects of the database's semantics into the answer in order to correct misconceptions with respect to the database schema. The answer to the above query would be:

"None."
"Professors teach courses."
"Students take courses."

McCoy [41] uses world knowledge to correct misconceptions a user might have about the properties of a given object. For instance,

Q: "Where are the gills on a whale?"
The system knows the user probably thinks whales are fish
because fish use gills to breathe.
A: "Whales do not have gills. They breathe through lungs."

For a belief logic based view of ill-formed input see [18, 19].

1.3.4 Intensional Answers

An intensional answer denotes a non-enumerative characterization of a set. For example, if a user asks the query "Which students are enrolled?" in a context in which all students must be enrolled, it is misleading to return a list of all the students. It is better to answer "All students are enrolled". Intensional answers are more succinct than concrete answers and this is important when we are dealing with databases with huge stores of data. The work on intensional answers was started by Imielinski [26] and many researchers have since studied the topic, e.g, [4, 5, 44, 47, 48, 55]

1.3.5 Generalizations

The scope of a query is extended so that one can include information on related topics in the answer.

Cuppens and Demolombe [12] use a meta level definition of a query with additional variables that carry relevant information and these variables are reported together with the original ones in the answer. For example, the query

$\leftarrow travel(vancouver, oslo).$

might be modified such that the answer includes cost. They also introduced the notion of finding answers close to those asked for. For example, if a user asks for a flight from Vancouver to Frankfurt between 17:00 and 21:00 and the only flight to Frankfurt leaves at 21:05, it would be more useful to return that one than nothing.

Wahlster, Marburger, Jameson, and Busemann [58] introduce *over-answering* of yes/no questions when further questions on the same topic are anticipated from the user. Their system uses domain knowledge to guess which types of follow-up questions are likely. For example,

> **Q:** "Has a yellow car gone by?"
> The system adds that the user will want to know where.
> **A:** "Yes, one went by on Hastings St."

Chu, Chen, and Lee [7, 8], use a *type abstraction hierarchy* to provide related answers to queries in a relational database system. For example, if the query

AmericanAirlines_flight(burbank, dulles, at_10am)

fails, it can be abstracted into the more general query

CoastToCoast_flight(los_angeles, washington, morning)

and the database can provide close alternative answers.

Gaasterland, Godfrey, and Minker [24] introduces the notion of *relaxation* of queries to deductive databases. The scope of a query is expanded by relaxing the logical constraints implicit in a query.

1.4 The Main Challenges and Our Solutions

In developing an ASL front end to database knowledge, the main challenge is of course the pioneering character of our task. Because this task has not been previously attempted, and because the features of ASL are so suigeneris with respect to the features of spoken languages for which front ends have been previously studied, we had to extensively research the literature on ASL looking for any kind of helpful clues, and rely on our own imagination to gradually develop insights on useful pieces of representation and on how to put them together.

Because of the special needs of deaf people, we set out to achieve a cooperative environment for question answering. We found that together with those ASL idiosyncrasies that made our task challenging. there were also others that actually lent themselves particularly well (better than oral languages) to the cooperative answering we were intent upon providing.

In the areas of cooperative answering itself, there exists an extensive body of literature, but previous research has focused on individual issues in cooperative answering (e.g., intensional answers, detection of presuppositions, correcting misconceptions) and each solution is cast in a different framework. Our main task was to develop an integrated framework which would simultaneously provide a useful representation of ASL queries and a cooperative evaluation of these queries via such representations, such that most of the issues raised in cooperative answering could be solved within this single framework. To achieve this, we developed the rigorous, typed, multi-valued logical system LM, in terms of which ASL queries can be expressed as (automatically obtained) formulae. In this framework, we formally define a database as an interpretation or situation, i.e. as a multi-valued assignment of a relation to each relational symbol. Database consultation then reduces to the automatic evaluation of a formulae corresponding to a given ASL query with respect to a given situation, and following the also rigorously defined semantics of LM.

1.5 Organization of the Thesis

In chapter one we introduce the topic of the thesis. In chapter two, we review Colmerauer's three-valued formalism, the typed logical representation L3, logic grammars and incomplete types. In chapter three, we motivate each of the new logical values and discuss treatment of some of these values. In chapter four, we present the definition of a multi valued logic database and corresponding query language. In chapter five, we give a brief introduction to American Sign Language. In chapter six, we propose a computational model for a subset of ASL that is suitable as a front end to a database. In chapter seven we summarize and discuss our results.

Chapter 2

Natural Language Processing

In this chapter, we give a brief review of the various natural language processing theories and tools that form the background for this thesis. In section one, we discuss Colmerauer's three-valued logic representation for a subset of natural language. In section two, we discuss V. Dahl's extension of Colmerauer's work into the database query language L3. In section three, we discuss incomplete types and in section four, we discuss logic grammars.

2.1 Colmerauer's Three-Valued Logic

In the late 1970's, A. Colmerauer developed a logical representation for the meaning of a subset of natural language expressions [10]. In his proposal, each quantified expression in a given natural language (expressions introduced by articles such as a, the, all, some, no) is assigned an intermediate three-branched quantifier representation. The final representation is a formula from a three valued logical system, whose main components are the single quantification mechanism $for/3$[1], logical operators that range over three logical values, set formulae, statement formulae and integer formulae. The formal definition can be found in [10] and we provide a summary in 2.1.7. For other work in multi-valued and fuzzy logics, see [40, ch.12].

In this section, we briefly discuss the main features of Colmerauer's treatment of a subset of English: quantification hierarchy, three logical values, the single quantification mechanism and formulae. In the following section, we discuss how to translate natural language statements into rigorously defined formulae.

[1] We borrow notation from logic programming: *predicatename/N* means that predicate *predicatename* has *N* arguments

2.1.1 Elementary Statements

Elementary statements involve proper nouns plus either a) the verb *to be* and a common noun, b) a verb, or c) the verb *to be* and an adjective. Colmerauer hypothesized:

> "To each verb, to each adjective and to each common noun there corresponds a property with n arguments, each argument being a proper noun."

Elementary statements translate directly into logic formulae without any intermediate three-branched quantifier. Consider the following examples[2] and their associated logic formulae.

a) Garfield is a cat. = iscat(Garfield)
Jörg is the son of Heidi. = issonof(Jörg, Heidi).

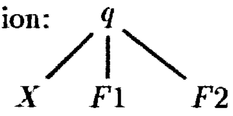
b) Garfield trots. = trots(Garfield)
Diana lent Garfield to Brigitte. = lentto(Diana, Garfield, Brigitte)

c) Garfield is striped. = isstriped(Garfield)
Garfield is happy with Diana. = ishappywith(Garfield, Diana)

2.1.2 Quantification Hierarchy

Natural language determiners are the basis for translating sentences into three-branched quantifiers. Let us first consider, simple sentences consisting of a noun phrase followed by a verb phrase, where the noun phrase contains a determiner. Such sentences can be represented by a three-branched quantifier of the following form:

$q(X, F1, F2)$ In tree notation:



where q represents the determiner, X is a variable, F1 is the noun phrase's representation, and F2 is the verb phrase's representation. Intuitively, F1 specifies the domain of quantification, and q states what portion of this domain F2 holds for. (For an alternative account of quantification see [11].)

In general, for any given natural language quantification, q, a three-branched quantifier relates a variable X, to two formulae $F1$ and $F2$. (As opposed to the classical quantifiers $\exists$ and $\forall$, which relate a variable to a single formula, three-branched quantifiers relate a variable

[2] All the examples in this section are taken from [10] (but modifed a little).

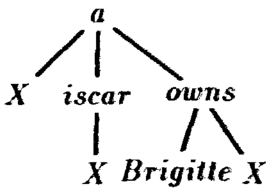
X, to *two* formulae.) Consider the following example of how to translate statements with only one quantification from natural language into three-branched quantifiers:

Brigitte owns a car.

This statement can be be paraphrased as:

for an X such that X is a car, it is true that Brigitte owns X

and represented by the following three-branched *a*-quantifier:

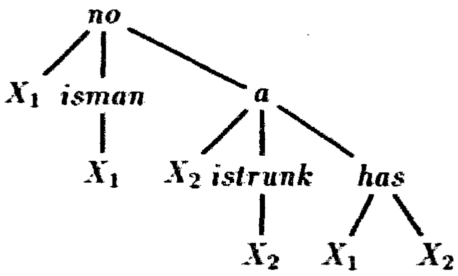


where $iscar(X) = F1$ and
$owns(Brigitte, X) = F2$

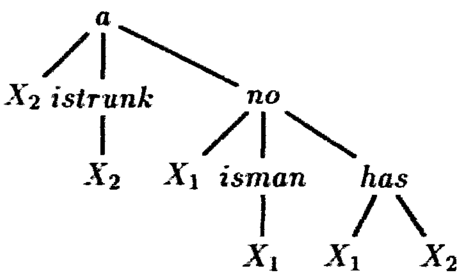
In the rest of this section, we discuss how to translate sentences with more than one quantification into three-branched quantifiers. We note that the hypotheses presented here are valid in most cases, but were not designed to be infallible and we will examine a couple of cases where they would give incorrect representaions. They represent a useful compromise between coverage and simplicity and his hypotheises give the preferred readings in the majority os cases. First we discuss sentences where the subject and the complement of the verb both contain a quantification. Consider the following sentence and its two possible translations into a three-branched quantifier:

No man has a trunk.

(1)



(2)



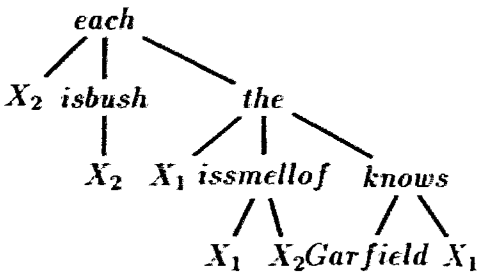
The second representation is not correct (it would mean something like: there exists a trunk that no man owns), and from experimentation with several articles in several sentences, Colmerauer [10] hypothesizes:

> "The quantification introduced by the article of the subject of a verb dominates the quantification(s) introduced by the complements(s) closely related to that verb. In speaking of complements closely related to the verb, we exclude adverbial phrases, which will not be studied here."

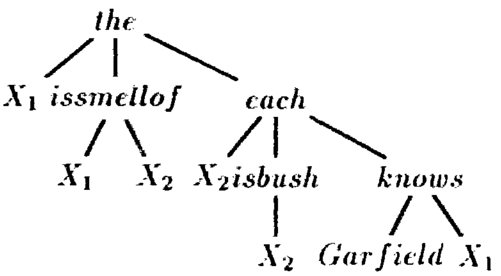
Next we discuss sentences that include a noun and a complement of this noun. Consider the following sentence and its two possible translations into a three-branched quantifier.

Garfield knows the smell of every bush.

(1)



(2)



Again the second translation is not correct (it would mean something like: the smell of every bush is the same and Garfield knows it), and from a series of similar examples, Colmerauer arrived at the following hypothesis:

> "In a construction involving a noun and complement of this noun, the quantification introduced by the article of the complement dominates the quantification introduced by the article of the noun."

We would like to point out that there are situations where this hypothesis does not hold. Consider the following example:

I know the ambition of every politician.

In this example, there is only one 'ambition' so the quantification of the noun should in fact dominate the quantification of the complement. However, if we change *every* to *each* in the above sentence:

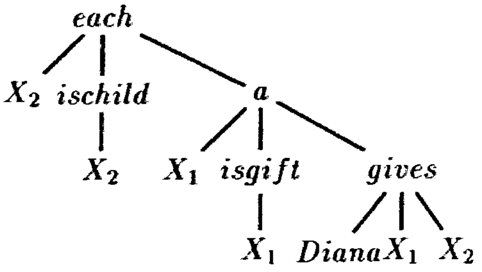
I know the ambition of each politician.

Colmerauer's hypothesis gives us the preferred reading!

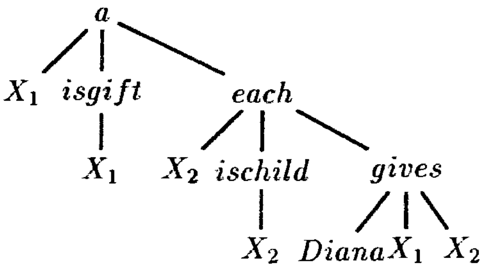
Next, we discuss sentences where a verb, an adjective or a noun has two complements. Consider the following example.

Diana gave a gift to each child.

(1)



2)



The second quantifier representation implies that there was only one gift and the children were sharing this gift. Although, this may have been the author's intention, it is more likely that she meant that each child got a separate gift (the interpretation is context dependent). Colmerauer hypothesized:

> "Whenever a verb, an adjective or a noun has two complements, the quantification is made in the inverse order of the natural order of their appearance; that is, the rightmost complement generates a quantification dominating the quantification generated by the other complement."

This hypothesis is clearly too simple, since the sentence

a) Diana gave each child a gift.

should presumably produce the same representation as

b) Diana gave a gift to each child.[3]

If we follow Colmerauer's hypothesis, a) would give us the representation in 2) above, which we agreed that is not correct. It seems that if the quantification of the recipient is not *a* or *the* and follows some other quantification, we shall adhere to the old hypothesis and process the complements from right to left, while if the quantification of the recipient is not *a* or *the* and precedes another quantification, we process the complements from left to right. Under this hypothesis both a) and b) above will produce the same representation.

Note that this only applies if the quantifications of the recipient is not *a* or *the* (e.g., some, each, all, every). In the following example,

[3] If we take topic and focus into account when analysing these sentences, they should not give the same representation, since differences in emphasis brought about by focus/topic changes, through changes in constituent ordering, should rigorously speaking be accounted for in the representation. However, for the purpose of extracting information from a database such degree of detail is not necessary, and we shall therefore disregard focus/topic distinctions in this thesis. Interested readers can refer to [53].

Allan gave the cat a fish.

Allan gave a fish to the cat.

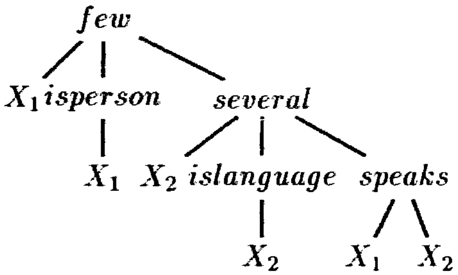
the right to left processing of complements lets us represent the slight difference in emphasis of these two statements.

Finally, we discuss the transformation of a sentence from the active to the passive voice. Consider the following active/passive sentences and their translations into three-branched quantifiers:

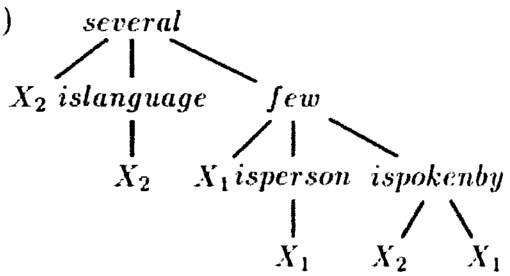
Few people speak several languages.

Several languages are spoken by few people.

(1)

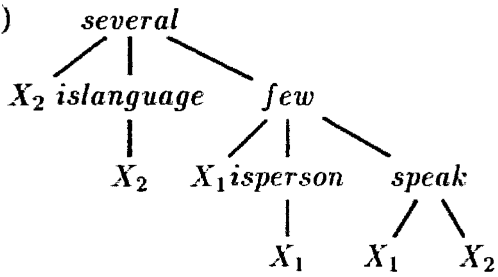


(2)



If we assume that $speak(X_1, X_2) = isspokenby(X_2, X_1)$, (2) becomes:

(2′)



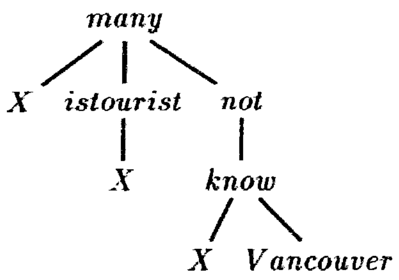
This leads us to conclude that passive sentences reverse the quantification hierarchy between subject and a complement of the verb.

2.1.3 Negation

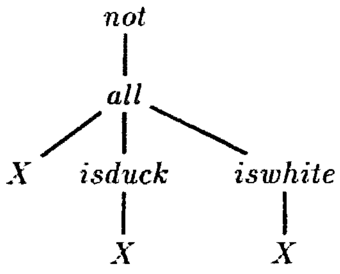
Negated statements introduce the operator *not*. If the quantification introduced by the subject is *every* or *all*, *not* applies to the whole statement and is placed above the quantification.

For all other quantifications it is placed immediately below the quantification introduced by the subject. For other research on negation see for example [42, 53]. Consider the following examples:

Many tourists do not know Vancouver =



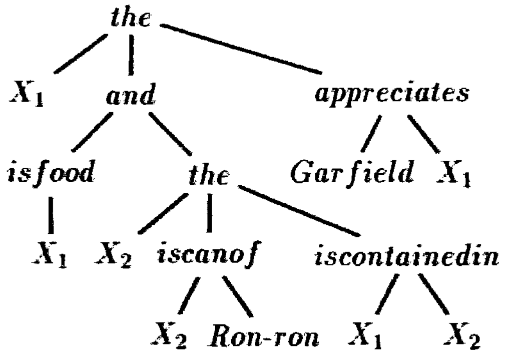
All ducks are not white =



2.1.4 Conjunction between Statements, Relative Clauses

The relative clause itself is treated as an ordinary statement where the relative pronoun is replaced by a variable. It is linked to the noun by the conjunction *and*. Consider the following example:

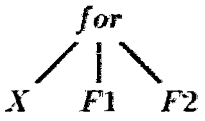
Garfield appreciates the food that is contained in the can of Ron-ron.



The three-branched quantifiers we have seen above are merely devices that simplify the translation into the single quantification mechanism *for/3*, which we will discuss next.

2.1.5 A Single Quantification Mechanism

For each quantifier, q, in the natural language, Colmerauer proposed a translation of its three branched quantifier into the single quantification mechanism $for/3$. This quantification mechanism is represented by the formula:



$= for(X, F1, F2)$ in functional notation,

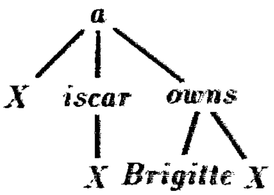
which is interpreted as follows:

for those Xs that satisfy $F1$, condition $F2$ holds.

Recall the following example:

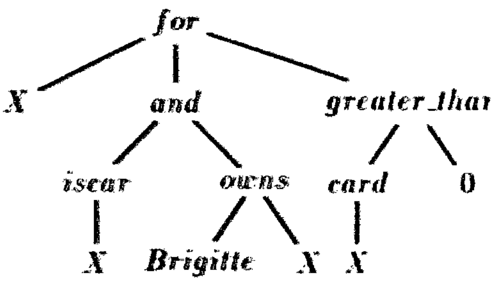
Brigitte owns a car.

which we translated into the three-branched a-quantifier:



where $iscar(X) = F1$ and
$owns(Diana, X) = F2$

and now we translate this three-branched quantifier into the following $for/3$ formula:



In general, three branched a-quantifiers are translated into $for/3$ formulae as follows (from [14]):

$a(X, F1, F2) = for(X, and(F1, F2), greater\_than(card(X), 0))$

Similarly, the three-branched quantifiers $all, no, some$ and integers i translate into for formulae (from [14]):

$$all(X, F1, F2) = for(X, and(F1, not(F2)), equal(card(X), 0))$$

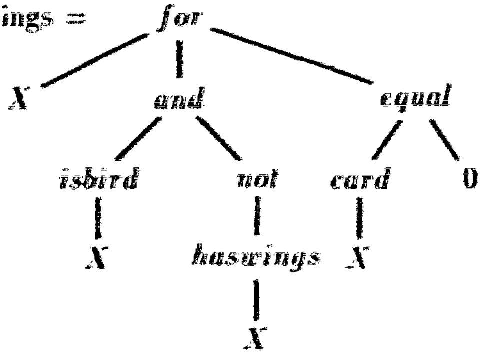
$$no(X, F1, F2) = for(X, and(F1, F2), equal(card(X), 0))$$

$$i(X, F1, F2) = for(X, and(F1, F2), equal(card(X), i))$$

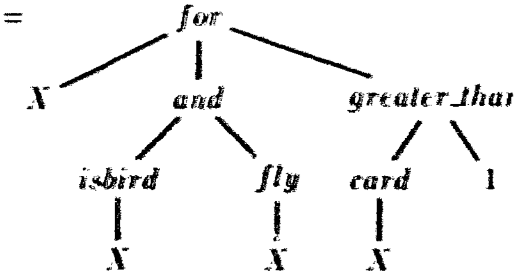
$$some(X, F1, F2) = for(X, and(F1, F2), greater\_than(card(X), 1))$$

Some more examples:

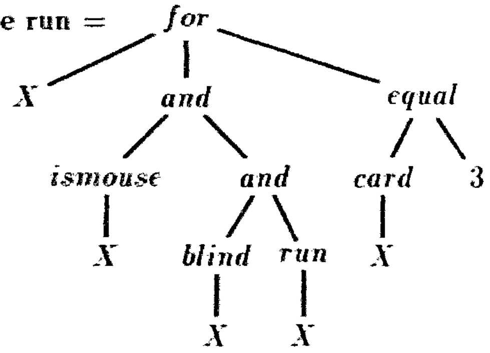
All birds have wings =



Some birds fly =



Three blind mice run =



2.1.6 Three Logical Values

A third logical value *undefined* is introduced in order to detect failed presuppositions (presuppositions are discussed in detail in, for instance, [40, ch.9],[11, ch.6]). Consider the following statement:

> The cat that Jörg is holding is mewing.

If Jörg is actually holding a cat, this statement evaluates to *true* or *false* depending on whether the cat is mewing or not. If Jörg is not holding a cat, the statement is meaningless and evaluates to *undefined*. Note, that we cannot say that the statement is *false* since that implies that its negation:

> The cat that Jörg is holding is not mewing.

is *true*. Clearly, this is no more true than the original statement. The article *the* presupposes existence and uniqueness of its referent, so if Jörg is not holding a cat, we say that the presupposition failed.

Colmerauer defined the operator *if* to deal with presuppositions induced by the definite article. Sentences that contain a definite article translate into L3 as follows (from [14]):

> singular-the(X, F1, F2) = for(X, F1, if(equal(card(X), 1), F2))
>
> plural-the(X, F1, F2) = for(X, F1, if(greater_than(card(X), 1), F2))

2.1.7 A Logical System for a Subset of Natural Language

We briefly summarize the logical system underlying L3. For a complete definition of the logical system see [10].

We define three types of formulae: set formulae s, statement formulae e, and integer formulae n. A set formula can take any of the forms:

- a list of constants
- a variable
- *those*(V, e), where V is a variable and e is a statement formula

A statement formula e can take any of the forms:

- $for(V, e_1, e_2)$, where V is a variable, e_1, e_2 are statement formulae
- $r(s_1, \ldots, s_n)$, where r is a relational symbol, $s_1, \ldots, s_n$ are set formulae
- $and(e_1, e_2)$, where e_1, e_2 are statement formulae
- $if(e_1, e_2)$, where e_1, e_2 are statement formulae
- $not(e_1)$, where e_1 is a statement formula
- $equal(n_1, n_2)$, where n_1, n_2 are integer formulae
- $greater\_than(n_1, n_2)$, where n_1, n_2 are integer formulae

An integer formula n, can take any of the following forms:

- $j \in \mathcal{N}$
- card(s), where s is a set formula

In a well defined situation a statement formula will evaluate to *true*, *false* or *pointless*, a set formula will evaluate to a set, and an integer formula will evaluate to an integer.

2.2 L3-a Natural Language Oriented Database Query Language

V. Dahl extended Colmerauer's theoretical work into the *typed* database query language L3 [13, 15]. Dahl translates a wide range of natural language sentences (elementary statements, conjunctive statements, relative clauses, passive sentences, and negated statements) into L3 through a natural language processor and uses Colmerauer's three-branched quantifiers as intermediate representations, while the final representation in L3 involves the single quantification mechanism $for/5$, where types are used to determine the search domain. Below we discuss the new features of L3 that were not dealt with in the previous section.

2.2.1 L3's Quantification Mechanism

$for/5$ differs from Colmerauer's $for/3$ in that the search domain is explicitly defined through types. Each constant and variable is typed (its domain) so the search space is narrowed to only those constants of a variable X's type (its domain) D. This narrowing of search space clearly improves efficiency. The quantification mechanism $for/5$ is represented by the following formula:

$for(S, X, D, P, C)$[4]

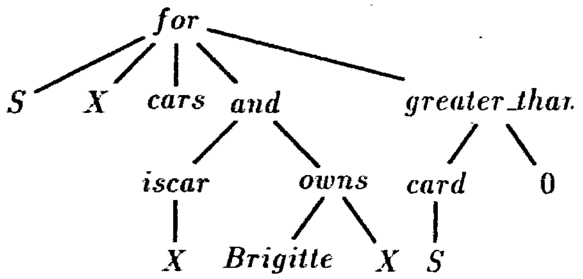
which is interpreted as follows

for the set S of those Xs in domain D that satisfy P, condition C holds.

Consider our previous example

Brigitte owns a car

which translates into the following $for/5$ representation



2.2.2 Safe Negation as Failure

It is only safe to evaluate negated formulae when their arguments are ground at the time of evaluation, but in a natural language front end we cannot guarantee safe negations. For example, the user's request for an inexpensive item, might result in the unsafe query:

a) not(expensive(X)), item(X)

instead of the safe query:

b) item(X), not(expensive(X))

[4] In an actual implementation, S and D appear in less explicit forms.

If there are any expensive items in the knowledge base, the first clause of a) will fail, and no items are returned as an answer. In b), individual items are retrieved and then examined, and the correct list of inexpensive items is returned.

The quantification mechanism $for/5$ makes negation as failure safe through its use of types, and the above query can safely translate into:

$$for(S, X, items, and(not(expensive(X)), item(X)), greater\_than(card(S), 0))$$

During evaluation, elements are successively taken from the domain *items* and added to S if not expensive.

2.3 Incomplete Types

Consider the query *Is Sammy happy?*, given a database where Sammy is a seal, seals are animals, and animals are happy. In Prolog:

```
?- happy(sammy).

happy(A) :- animal(A).
animal(A) :- seal(A).
seal(sammy).
```

From this ordinary untyped database, we can obtain a positive answer to the query in three resolution steps. By introducing a *typed* logic database, we can obtain a solution to the same query in only one resolution step. From the above program we can extract the following hierarchical taxonomy, `animal ⊃ seal ∋ sammy`, and we can write a compiler which transforms the taxonomy and the untyped database into the following typed database:

```
1) happy(A-[animal ⊃ T]).
```

where `A` is a variable of type `[animal ⊃ T]`. This type is incomplete in that it contains a tail variable which allows for further instantiation. Thus `[animal ⊃ T]` stands for "at least of animal type". (Input to the compiler would look something like: `happy(A ∈ animal)`, `reptile ⊂ animal`, `crocky ∈ reptile`. In general: $t_i \subset t_j$, where $t_i, t_j \in T$ and $k \in t_k$ where $t_k \in K$.)

Similarly the compiler can transform the query into:

```
2) ?- happy(sammy-[animal ⊃ seal ∋ sammy]).
```

where `sammy` is a constant of type `[animal ⊃ seal ∋ sammy]`. (Note, the type representation of constants is closed such that no further instantiation is possible.) When this query is presented to the typed database, a solution is obtained in one resolution step. Further, when 1) and 2) are resolved, `A` unifies with `sammy` and `T` with `seal ⊃ sammy`, thus making the type of `A` further known as both seal and animal.

In this example, the number of resolution steps were reduced from three to one. In general, for strictly hierarchical taxonomies including a chain of n set inclusions, the number of resolution steps are reduced from n to *one*. We can obtain the same reduction of resolution steps through partial evaluation of logic programs (see, for example [52]), but since we cannot obtain the other advantages (see below) that types offer through partial evaluation, we prefer to stick to types.

Incomplete types is a natural addition to a database since database relations are typically typed anyway. Incomplete types allow for disambiguation of some statements by reducing semantic agreement to syntactic matching (see 3.3), and they reduce the search space to pertinent domains rather than the whole Herbrand universe.

Definition: An *incomplete type* for t , denoted $h(t)$, is a term of the form:

$$t_1 \supset, \cdots, \supset t_{n-1}, t, \supset V,$$

where V is a variable ranging over the incomplete type $t_1 \supset, \cdots, \supset t_{n-1}, t$ and where there exists no t_0 such that $t_0 \supset t_1$

In Prolog notation this would be list of the form $[t_1, \ldots, t_{n-1}, t \mid V]$, where $|$ is a binary infix operator that separates the tail from the rest of the list.

Property: Let $s, t \in T$. Then $s \subset t \Leftrightarrow \exists h(t), h(s)$ and a substitution θ such that $h(s) = h(t)\theta$. (See proof in [16].)

Remark: As a practical consequence of this property, a type s can be proven to be a subtype of t simply by unifying h(s) and h(t), and checking that t's tail variable has become instantiated.

2.3.1 Intensional Answers

As we saw above, domain sweeping is necessary for safe negation, but it is expensive when the domains are large. In a typed database, we can avoid domain sweeping by producing an intensional answer to negated queries: Consider the query:

> "Which animals are not happy?"

with respect to the animal knowledge base from section 2.3. We can reply

"All but reptiles."

and as before, we may add an option for the user to ask for a complete listing. By allowing intensional answers, negation is reduced to set complement rather than domain sweeping.

2.4 Logic Grammars

The term "logic grammars" refers to a whole family of grammars, that was developed for computational analysis of natural languages (see [1, 45]) (lately they have also been used for parsing of formal languages [1]). Logic grammars are high level grammar description tools which are usually built on top of the Prolog programming language although they can also be considered a separate formalism. They are automatically translated into Prolog such that all the details of the Prolog mechanism are hidden from the users and they can concentrate on linguistic and parsing issues instead of computer issues.

Some logic grammars are: Metamorphosis Grammar, Definite Clause Grammar, Extraposition Grammar, Static Discontinuity Grammar. Even though logic grammars are basically syntactic variants of Prolog, they are recognized as a distinct and powerful formalism in their own right. In the rest of this section we will give a short introduction to logic grammars.

Logic grammars are similar to type-0 grammars in the Chomsky hierarchy of formal grammars [6]. A type-0 grammar, $\mathcal{G}$ is defined:

$\mathcal{G} = (V, T, P, S)$ where V is a finite set of non-terminal grammar symbols
T is a finite set of terminal grammar symbols
P is a finite set of rules
S is a start symbol

Assume $V \cap T = \emptyset$ and P is of the form $\alpha \rightarrow \beta$ where α and β are strings of grammar symbols from $(V \cup T)^*$. Let $\mathcal{L}(\mathcal{G})$ be the language generated by a grammar $\mathcal{G}$, then for every type-0 grammar $\mathcal{G}$ there exists a Turing machine that recognizes $\mathcal{L}(\mathcal{G})$.

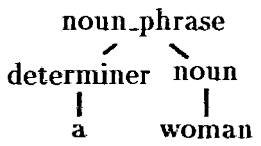
The above definition also applies to logic grammars (In natural language terminology, terminals are words, while non-terminals are grammar constituents.), but logic grammars differ from formal grammars in that the grammar symbols are not atomic—they may be logical *terms* with *arguments*. In general, logic grammar symbols (terms) are of the form:

$< term > ::= < variable > \mid < constant > \mid < complex\_term >$
$< complex\_term > ::= name(< term >, < term >, \ldots, < term >)$

By convention, constants are written in lower case. Terms are often represented as a trees. For example, the term

noun-phrase(determiner(a), noun(woman)).

is represented as the tree



A variable stands for an as yet unidentified constant or complex term. Consider the following grammar, $\mathcal{G}$.

(1) `sentence(ParseTree) → proper_noun(Pers, Num, Subj), verb(Pers, Num, Subj, ParseTree)`

(2) `proper_noun(third, singular, name(diana)) → [diana].`[5,6]

(3) `verb(third, singular, Noun, study(Noun)) → [studies].`

(4) `verb(AnyPerson, plural, Noun, study(Noun)) → [study].`

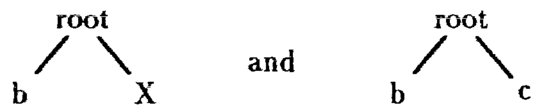
Here, (1)-(4) are the rules of grammar $\mathcal{G}$. `sentence(ParseTree)`, `proper_noun(Pers, Num, Subj)`, `verb(Pers, Num, Subj, ParseTree)`, `proper_noun(third, singular, name(diana)` etc. are nonterminal grammar symbols. `[diana]`, `[studies]` and `[study]` are terminal symbols. `Sent`, `Pers`, `Num` and `Subj` are variables. `third`, `singular` and `plural` are constants. `name(diana)` and `study(Noun)` are trees.

The execution of a logic grammar embedded in a programming language like Prolog is based on *unification* of grammar symbols and their arguments. Unification is defined as follows [1, p.8-9] (For a more elaborate account of unification see [56, p.68-72]:

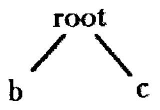
> "Given two terms which may contain variables, unification is the process of finding values for those variables, if such values exist, that will make the two terms identical. The set of value assignments that makes two terms equal is called a *substitution*."

For instance the trees

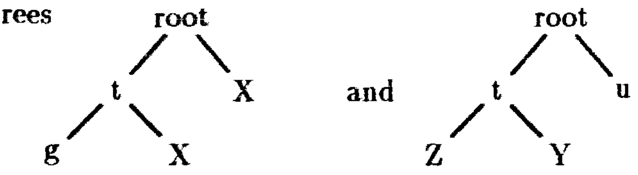
[5] Unlike ordinary English, proper names are written in lower case since variables start with a capital.
[6] Terminals are written between '[' and'['.



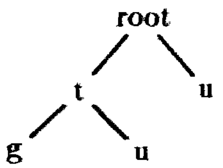
unify into



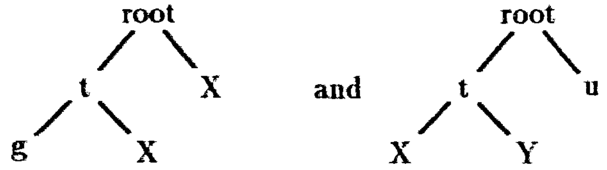
The trees



unify into



If two trees share any variable names, all occurrences of these must be *renamed* in one of the trees before attempting unification. For instance, in the trees:



rename X in the second tree to a new variable, say Z (alternatively, rename both X's in the first tree). Renaming is done automatically in Prolog and as a consequence, we can use the same variable names in different rules without them interfering with each other. Only variables used more than once in the same grammar *rule*, refer to the same entity, i.e., a substitution applies to *all* occurrences of a variable in the same rule. Consider the above grammar, both occurrences of `Noun` in rule 3 refer to the same entity, while `Noun` in rule 3 and `Noun` in rule 4 are considered different variables.

If we execute a logic grammar with no input, it will return/generate all possible sentences in the language described by the grammar. Consider the following simplified version of grammar $\mathcal{G}$.

```
(1) sentence → proper_noun(Pers, Num), verb(Pers, Num).
(2) proper_noun(third, singular) → [diana].
(3) verb(third, singular) → [studies].
(4) verb(AnyPerson, plural) → [study].
```

The execution of this grammar with no input proceeds as follows (All grammar symbols are processed left to right.):

1. sentence rewrites into proper_noun(Pers, Num) and verb(Pers, Num). In Prolog terminology, we say that sentence creates the two subgoals proper_noun(Pers, Num) and verb(Pers, Num).

2. proper_noun(Pers, Num) unifies with proper_noun(third, singular) (rule (2)) with substitutions {Pers=third, Num=singular}

3. proper_noun(third, singular) rewrites to [diana] and since [diana] is a terminal, it is put on the output stream.

4. The second subgoal of rule (1), verb(third, singular), unifies with verb(third, singular) (rule (3)).

5. verb(third, singular) rewrites to [studies] and since [studies] is a terminal, it is added to the output stream.

6. All subgoals of rule (1) are successfully completed so Prolog empties its output buffer and prints [diana, studies] as the first sentence of this grammar. Prolog continues systematically to look for all other substitutions which satisfy the goals through a process called *backtracking*. Next, Prolog tries to satisfy the second subgoal of rule (1) through unification with rule (4). At this point, substitutions {Pers=third, Num=singular} are still valid since they were obtained in a previous subgoal, so the unification procedure elegantly prohibits the generation of [diana, study]: The second argument of verb in rule (1) is singular and the second argument of verb in rule (4) is plural so verb(third, singular) (rule (1)) cannot unify with verb(AnyPerson, plural) (rule (4)). (This example shows syntactic agreement, but the same technique can apply to semantic agreement as discussed in 3.3.)

   Since there are no alternative substitutions which satisfy the second subgoal, Prolog tries to redo the first subgoal. Of course, there are no alternative substitutions which satisfy this goal either, so [diana, studies] is in fact the only sentence in the language generated by $\mathcal{G}$.

So far, we have seen how a grammar can generate all possible sentences of a particular language. Another, and perhaps more useful, property of logic grammars is that one can present them with an input sentence and ask the grammar to parse it. If we ask the above grammar to parse the sentence "Diana studies" (input is in the form of a list: [diana, studies]), it proceeds as described above except that nothing is put on the output stream. At the end of a successful parse it simply prints 'yes'. If the grammar is asked to parse a sentence which is not in the language (e.g., Diana study), it prints 'no'.

In the previous example, we saw how the user can include syntactic information in the arguments in order to guide the parse. If the user wants to build up a representation of a sentence, for instance, a parse tree or a semantic representation, she can do this through clever manipulation of arguments such that the representation is built up through rule application and unification in the course of parsing. Grammar $\mathcal{G}$ above is equipped with arguments for building a parse tree of the input sentence. If we ask it to parse "Diana studies", the execution proceeds as follows:

1. `sentence(Parsetree)` rewrites into `proper_noun(Pers, Num, Subj)` and `verb(Pers, Num, Subj, ParseTree)`.

2. `proper_noun(Pers, Num, Subj)` unifies with `proper_noun(third, singular, name(diana))` (rule (2)) with substitutions {Pers=third, Num=singular, Subj=name(diana)}

3. `proper_noun(third, singular, name(diana))` rewrites to `[diana]` and since `[diana]` is a terminal, the first subgoal of rule (1) is satisfied.

4. The second subgoal of rule (1), `verb(third, singular, name(diana), ParseTree)`, unifies with `verb(third, singular, Noun, study(Noun))` (rule (3)) with substitutions {Noun=name(diana), Parsetree=study(name(diana))}

5. `verb(third, singular, Noun, study(Noun))` rewrites to `[studies]` and since `[studies]` is a terminal, the second subgoal of rule (1) is satisfied.

6. All subgoals of rule (1) are satisfied so Prolog prints 'yes', and in addition it prints the value of the variable `ParseTree`, i.e., `Parsetree = study(name(diana))`.

7. As above, Prolog systematically searches for all alternative substitutions through backtracking. (In this case there are none.)

The output of the grammar is [diana, studies]

2.4.1 Definite Clause Grammar

The particular logic grammar used in this project is called *Definite Clause grammar* or DCG. DCG is a special case of Metamorphosis Grammar (MG) [9], with rules of the form:

$$S \rightarrow \beta$$

where S is restricted to a single nonterminal symbol and β is as before. (In some implementations, additional symbols are allowed on the left-hand side, as long as they are terminals.) Both the above examples are definite clause grammars.

Chapter 3

Multiple Logic Values

In this chapter, we introduce six new logical values in addition to the traditional values *true* and *false*. Further, we motivate each one of the truth values in our multi-valued logic and provide some sample statements where the new values are useful. We discuss which of the new values need to be explicitly defined in the logical system underlying the database and which are useful conceptual tools and need not be explicitly defined. Our database system is typed, and we discuss how the inclusion of types affects query evaluation; some of the new logical values are assigned to semantically anomalous queries as a result of type incompatibility. Also, types offer a simple and efficient way of giving intensional replies.

3.1 Motivation

The truth values *true* and *false* largely retain the meaning they have in a traditional binary logic. However, the new values allow for subtler distinctions than what is possible in a binary logic, so some statements that in a binary logic would evaluate to *false* will now evaluate to *pointless, absurd, mixed, unknown*, or *inconsistent*. Next we motivate each of these new logical values.

3.1.1 Pointless

This value was introduced in L3 (see sections 2.1 and 2.2) for detection of failed presupposition (called *undefined* there), and we only repeat the discussion here for completeness. The idea is that any sentence that contains a failed presupposition evaluates to *pointless* instead of *false*. For instance, if we evaluate

1) The sales report that John wrote was ready in one day.

with respect to a knowledge base representing a world in which Alice, not John, wrote the sales report, it is assigned the value *pointless* and not *false* since *false* implies that the negation of the statement:

2) The sales report that John wrote was not ready in one day.

would have to be acknowledged as *true*. Because the presupposition fails, it is pointless to say whether the report was or wasn't ready in one day.

3.1.2 Absurd

Some statements contradict basic world knowledge rather than making a wrong assumption. Consider the following example:

3) Can Rover speak Latin?

If we evaluate this statement with respect to a knowledge base of ordinary dogs, it evaluates to *absurd* since ordinary dogs don't speak. While in the previous example we can conceive of John having written a sales report, this statement is considered semantically anomalous.

Some statements are syntactically ambiguous, but have only one semantically acceptable interpretation. Humans will not likely choose the wrong interpretation in these cases, but a computer may, since both interpretations are syntactically correct. Consider the following question:

4) What is the price of a recorder which can play stereo music?

It admits two interpretations: the intended one, where the recorder plays stereo music (i.e., "recorder" is the antecedent of the relative clause) and an unintended one, where the price plays stereo music (i.e., "price" is the antecedent of the relative clause). The latter interpretation should evaluate to *absurd*.

3.1.3 Vague

The value, *vague*, helps us deal with statements/questions that are under-specified. For instance, the question

5) Who teaches Norwegian?

could result in the database system sweeping through the domain of all people in the knowledge base, if the interrogative pronoun is used to find a domain for the object queried. Vague-referring pronouns could induce a *vague* truth value, indicating that further domain determination can perhaps be done from the context.

3.1.4 Mixed

Some relationships distribute a certain property among all the elements of their arguments. For example, the relation **like** as in the following example:

6) Ann and Tom like karate.

To evaluate this statement, the database system checks whether Ann likes karate and whether Tom likes karate. If both (none) of them do, the statement evaluates to *true (false)*. But if one likes karate, while the other does not, it evaluates to *mixed*, which is a much more informative reply than what we can expect from a binary logic or L3, where the statement would simply evaluate to *false*.

The value *mixed* is also useful when evaluating relations that are introduced by the word "respectively" in any given sentence. For example, the statement

7) Tom and Ann earn $1000 and $1100 respectively

evaluates to *mixed* if Tom earns $1000 while Ann does not earn $1100, or if Ann earns $1100 and Tom does not earn $1000.

3.1.5 Unknown

Generally speaking, we shall adhere to the closed world assumption (Reiter 1978 [51]) which allows us to infer $\neg A$ from a logical program P, if A is not a logical consequence of P. In logic programming, negation ($\neg A$) is implemented as failure, i.e., only positive information is listed in the database and if a closed world database system cannot find a given ground fact (i.e., prove it), we infer that the fact is false.

Open world databases, admit positive, negative and unknown information, but it is expensive to implement open worlds since the negative and unknown information usually exceed the positive information by several orders of magnitude. Since one has to distinguish between unknown and negative information, one of them has to be defined explicitly and thus the conciseness allowed by negation-as-default is lost.

However, for relations whose arguments are in a one-to-many relationship, we introduce a way of simulating an open world situation, while only including positive information in the knowledge base. In this situation we distinguish between *false* and *unknown* queries by introducing a metalogical treatment of values retrieved from the knowledge base. Consider the following example:

8) Does Ann live in Paris?

This query can be compiled into something like: "Find the place X where Ann lives; if X is ground and equal to "Paris", evaluate the query to *true*; if X is ground and different from "Paris", evaluate the query to *false*, otherwise to *unknown* (i.e., failure implies *unknown*). The *unknown* value allows us to produce an informative reply instead of a misleading negative answer.

Only for one-to-many relations can we allow failure to imply *unknown* and for many-to-many relations we must maintain the traditional interpretation of negation as failure. Consider the following example: knowing that Betsy is David's aunt does not, in an open world, authorize us to conclude that Doreen is not, since David can have many aunts. Thus for many-to-many relationships we shall stay within closed worlds and use negation as failure and only those relationships in which two arguments are in a one-to-many relationship will be tested for *false* versus *unknown*.

This approach should, of course, be used with caution, and it certainly does not solve all open world problems as we have seen. But it does provide a compromise that allows greater flexibility than completely closed worlds.

3.1.6 Inconsistent

Some queries evaluate to null value because they are inconsistent to begin with. For instance, in a world where all secretaries are insured and all part-time employees are not (hence implying that no secretaries are part-time), it would be inconsistent to query:

9) Which part-time secretaries are insured?

Rather than simply answering "None", which does not point out the user's inconsistency, it would be more helpful to reply, for instance, that all secretaries are insured whereas part-time employees are not.

Rigorously speaking, the difference between *inconsistent* and *absurd* is a matter of degree - *absurd* being assigned when the situation is unimaginable in the world considered, and *inconsistent* being assigned for forbidding situations that are not inconceivable but are disallowed in our database (e.g., integrity constraints). Inconsistent queries could also evaluate to *pointless* since the user presupposes something that is not consistent with the state of the database. However, we choose to separate the *inconsistent* case from the *absurd* and the *pointless* cases since the mistakes are indeed different. We think that database systems should have the flexibility of distinguishing between these values in order to provide

Assigned during:	Logical value
Parsing	*absurd, inconsistent, (vague)*
Database	*true, false, mixed, pointless, unknown*

Table 3.1: Implicit and explicit logical values.

appropriately informative responses in each case.

3.2 Implicit and Explicit Logical Values

Some of the truth values introduced in the section need to be explicit in the logical system that underlies our deductive database and query language, while others are useful conceptual tools and need not be part of the formal definition. Our multi-valued query language is intended to work together with a natural language front end, so some queries might be assigned truth values during the natural language parsing stage, while others will be assigned values during query evaluation, i.e., during the database consultation stage. Only queries that are found not to be semantically anomalous by the parser, produce a complete formula and become completely evaluated in the database system. When a query is assigned a truth value during the parsing stage, it is only partially executed and the reply is available without ever having to consult the database system. Table 3.1 summarizes which values are assigned during the parsing stage and which are assigned during the database consultation stage.

The values *absurd* and *inconsistent* indicate semantic anomalies and induce an immediate interruption of the parse. A complete formula representing the query is never generated, and the formula's evaluation with respect to the knowledge base never takes place. Since semantically anomalous queries never reach the database consultation stage, it is unnecessary to include the values *absurd* and *inconsistent* in the definition of the logical system underlying our database and query language. In section 3.3, we will show how the addition of semantic information to the natural language grammar is used to detect semantically anomalous queries.

Vague is just a conceptual tool that prompts for further domain determination and this truth value is never actually assigned to a query. The domain determination is a consequence of adding semantic information to the natural language grammar and knowledge base.

All the other values, *true, false, mixed, unknown* and *pointless*, are assigned during query evaluation and therefore have to be defined in the logical system that underlies our deductive database and query language. In chapter 4, we will give a formal definition of a typed deductive database and query language.

3.3 Semantic Types

As mentioned above, we will introduce semantic information in the grammar and the knowledge base. The semantic information takes the form of *incomplete types* (section 2.3). Here we will discuss how they are used to infer some of the logic values introduced above.

We can require the grammar to enforce type agreement between the arguments of a relation and the arguments of the main verb of a sentence. Returning to example number 3 above:

Can Rover speak Latin?

The concept of speaking may be represented by the following relation in the knowledge base:

speak(Subject-[person | X], Object-[language | Y])

i.e., the first argument of **speak** is of type *person*, while the query may introduce the formula:

speak(rover-[animal, dog, rover], latin-[language, latin]),

where the type of the first argument is *dog*. Since the types of the two first arguments don't match (Subject-[person | X] and rover-[animal, dog, rover] cannot unify) the parse is immediately interrupted and the value *absurd* is assigned to the query. This value is then interpreted by a natural language output module that will print a message indicating the reason for disagreement.

Types can make vague-referring expressions, e.g., interrogative pronouns, more precise as in our previous example:

Who teaches Norwegian?

The concept of teaching may be represented by the following relation in the knowledge base:

teach(Subject-[person, teacher | X], Object-[course | Y]),

i.e., the first argument of **teach**, is of type *teacher*, while the query might introduce the following formula:

teach(Query_Subject-[person | Z], norwegian-[course, norwegian]), i.e., the interrogative pronoun introduces a first argument of type *person*. Since *teacher* is a subtype of *person*, the two Subject argments are not incompatible and we take the intersection of the two types ($person \cap teacher = teacher$) to be the further specified type of Query_Subject. Through semantic agreement and unification (Z unifies with [teacher | X] as a result of parsing the query) the vagueness introduced by the interrogative pronoun simply disappears and the search space is considerably narrowed from *person* to *teacher* without ever having derived *vague* an explicit truth value.

Types can also serve to disambiguate some natural language queries which have more than one syntactically correct reading. Returning to example 4):

> What is the price of a recorder which can play stereo music?

If 'play' is represented by the relation

> **play**(Subject-[inanimate, device | X], Object-[music | Y])

Further if the type of 'price' is *price* and the type of 'recorder' is *device* the reading in which a price is required to play stereo music is made impossible since [inanimate, device | X] and [inanimate, price | Z] can not unify. Again the incorrect reading is assigned the truth value *absurd* at the parsing stage and if this reading can simply be discarded if a semantically acceptable reading exists. If not a natural language output module interprets the truth value and informs the user about the semantic anomaly.

Sometimes ambiguities arise because words mean different things in different situations. Consider the word 'bank' in the following examples:

> With which bank do you have an account?
>
> On which bank did you sit?
>
> On which bank did you fish?

In the first example, bank = money bank, in the second, bank = river bank, and in the last example, bank = fishing bank. If the relations **have_account**, **sit**, and **fish** all insist on a different type of argument, the incorrect readings are easily detected in the parsing stage.

Finally, inconsistent queries can be detected through type incompatibility. Recall our previous example where all full time employees and all secretaries are insured. Then the query:

> Which part-time secretaries are insured?

introduces a type mismatch that interrupts the parse.

3.3.1 Intensional Replies

If the database is *typed*, we can easily obtain intensional replies (see1.3.4). Further, if an extensional reply is requested, the search space is automatically reduced to the domains which are compatible with respect to type.

Let us add the fact that all reptiles crawl to our knowledge base of happy animals in section 2.3, and then consider the query "Which animals crawl?" with respect to this knowledge base. In Prolog:

```
crawl(A-[animal, reptile | X])
?- animal(A), crawl(A)
```

and the database would sweep through the whole domain of animals and test for each whether it crawls, and provide alternative answers upon backtracking. However, a typed database system can provide an intensional reply and only proceed to find specific answer instances if further prompted by the user. We can compile the query into:

```
?- crawl(A-[animal | Y ])
```

when this query is entered in the database system `Y` unifies with `[reptile | X ]` and we can answer the query with something like "All animals of type reptile crawl. Would you like a listing?". If the user requests a listing, the search space is already pruned and only the *reptile* domain will be examined.

3.3.2 Presupposition Formulae

In order to detect a failed presupposition in sentences with definite articles, our natural language analyser constructs formulae of the form:

$presupp(P, Q)$[1]

where P represents a presupposition, and Q is a statement evaluated with respect to P. In our previous example,

The sales report that John wrote was ready in one day.

P is the logical representation of "John wrote the sales report", and Q is the logical representation of "The sales report that John wrote was ready in one day". Thus, if P evaluates to *false*, the whole sentence can be assigned *pointless*, and a message can be produced informing the user of his/her failed assumption. The complete definition of *presupp* is given in the next chapter.

[1] Equivalent to *if* in Colmerauer's terminology

Chapter 4

A Typed Multiple-Valued Deductive Database

In this chapter, we provide a formal definition of a multi-valued typed logic database and its query language with respect to the logic values we introduced in the previous chapter. As discussed earlier, only some of the values need to appear explicitly in the definitions. Every relation in our database evaluates to one of the values *true*, *false*, *mixed*, and *unknown*. However, all four values may not apply to a particular relation, for example, any *many-to-many* relation can *not* evaluate to *unknown* since we adhere to the closed world assumption for *many-to-many* relations. Complete queries evaluate to either *true*, *false*, *mixed*, *unknown*, or *pointless* according to the definition in section 4.3.

The chapter is organized as follows. In section one, we describe the partitioning of relations. In section two we give the rigorous definition of a deductive database and in section three, we define the query language associated with our database.

4.1 Partitioning of Relations

Natural language sentences can introduce different types of plural, i.e., a relation may apply to a whole set as in "The beams are parallel", where the relation *parallel* must apply to the whole set of beams, while in our old example "Ann and Tom like karate" the relation *like* distributes to individual members of the set. Our system can recognize different kinds of plural and they are treated differently, therefore all the relations in our system must be partitioned into the disjoint groups *distributive*, *inherently collective*, *partially collective* and *respective relations*.

Distributive Relations: Relations hold on each individual member of a set. E.g., "Eve and Adam live in Paradise".

Inherently Collective Relations: Relations apply to a whole set of individuals. E.g., if we know that Eli, Jörg and Dia lifted the table, the query "Did Eli, Dia and Jörg lift the table?" evaluates to *true*, while the query "Did Eli and Jörg lift the table?" evaluates to *false* since the lifting cannot be distributed into one person at a time.

Partially Collective Relations: Relations apply on sets, but a subset is sufficient to satisfy a relation in the knowledge base. E.g., if we know that Eli, Jörg and Dia met in the park, the query "Did Eli and Jörg meet in the park?" evaluates to *true*.

Respective Relations: All the above types of relations can participate in respective relations. Arguments must have the same number of elements. E.g. "Eric and Martin study theory and AI respectively".

Each of the above groups of relations is further divided into many-to-many and one-to-many relations according to their arguments' relationship to each other.

Single Argument Relations: Relations with only one argument (a set) and the relation holds on each individual member of the set. E.g., "Leslie and Ann are students".

4.2 Rigorous Definition of a Typed Multiple-Valued Logic Database

Definitions:

— Let K be a finite set of symbols called *proper names.*

— Let T be a finite set of symbols called *types.*

— Let D be a finite set of symbols called *distributive relational symbols*

— Let IC be a finite set of symbols called *inherently collective relational symbols.*

— Let PC be a finite set of symbols called *partially collective relational symbols.*

— Let R be a finite set of symbols called *respective relational symbols.*

— Let S be a finite set of symbols called *single attribute relational symbols.*

— Let *Rel* be a finite set of symbols called *relational symbols* such that $Rel = D \cup IC \cup PC \cup R \cup S$[1].

[1] We will use the notation $A_{i,j}^{m-n}$ to denote the set of relations $A \in Rel/S$ such that the i'th argument is

— Let X be a set of variables.

— To each symbol $k \in K$ corresponds a symbol $t = type(k)$, with $t \in T$.

— To each variable $x \in X$ corresponds a symbol $t = type(x)$, with $t \in T$.

— To each symbol $r \in Rel$ we associate:

- a positive integer $n = degree(r)$.
- a list $[t_1, \ldots, t_n] = domain(r)$, where $t_i \in T$.

— Let $E(t)$ represent the set of proper names whose type is t.

— Let $P(E(t))$ represent the set of all subsets (the power set) of $E(t)$.

— Let $P_{sub}(E(t))$ represent a subset of the power set $P(E(t))$.

— Let $L = \{E(t) \mid t \in T\}$.

— Let $U = \bigcup_{t \in T} E(t) = K$.

Then a lattice is defined by L, the partial ordering relation of set inclusion ($\subseteq$), and the binary operators of set union ($\cup$) and intersection ($\cap$). It is bounded by the top U and the bottom $\{ \}$.

Definition: The *product* of sets $A_1, A_2, \ldots, A_m$ ($A_1 \otimes A_2 \otimes \ldots \otimes A_m$) is defined as follows: $A_1 \otimes A_2 \otimes \ldots \otimes A_m = \{< \alpha_1, \alpha_2, \ldots, \alpha_m > \mid \forall 1 \leq i \leq m,\ \alpha_i = \{a_i\}$ and $a_i \in A_i\}$, where $\forall\ 1 \leq k \leq m$ such that $A_k = \emptyset$, $\alpha_k = \emptyset$. (This definition is a variation of the cross product of sets: individual members of a tuple have to be singleton sets and the ordinary cross product gives $A \times \emptyset = \emptyset$.)

Definition: A relation *schema* $\mathbf{r}(t_1, t_2, \ldots, t_n)$, where $\mathbf{r} \in Rel$ and $\forall 1 \leq k \leq n$, $t_k \in T$, describes a *relation* whose name is $\mathbf{r}$. (We say: describes the relation $\mathbf{r}$.) $t_1, t_2, \ldots, t_n$ are the *attributes* of relation $\mathbf{r}$. $\forall 1 \leq k \leq n$, the domain of $t_k = E(t_k)$. (In logic programming terminology, attributes are called *arguments*. We will use these terms interchangeably.)

Definition: *Instances* of a relation $\mathbf{r}$ are denoted $\mathbf{r}(x_1, x_2, \ldots, x_n)$, where $\forall 1 \leq k \leq n$, $x_k \in P(E(t_k))$ if $\mathbf{r} \in D \cup IC \cup PC \cup S$, or $x_k \subseteq P(E(t_k))$ if $\mathbf{r} \in R$. The set $\{x_1, x_2, \ldots, x_n\}$

in an m-to-n relationship with the j'th argument. For instance, $C_{1,2}^{1-n}$ indicates the set of collective relations in which the first argument is in a 1-to-n relationship to the second. Note, the letters m and n used as *superscripts* and as *subscripts* are never related.

is sometimes denoted $\vec{x}$.

Definition: The *atomization* of an instance's *arguments* is the set of all possible arguments that can be constructed by taking the smallest meaningful decomposition of the arguments. *Meaningful decomposition* depends on the type of relation and will be formally defined below together with the definition of the different relations.

Definition: The smallest meaningful decomposition is called a *tuple*. Tuples are written between '$<$' and '$>$', also sometimes denoted $\vec{x}$.

An example: Consider the instance **speak**(english, [eli, diana])[2]. Since **speak** is a distributive relation, it can be meaningfully decomposed by computing the product of the arguments: $\{english\} \otimes \{eli, diana\} = \{< english, diana >, < english, eli >\}$.

Definition: The *atomization* of an *instance* is the set of instances obtained by applying its relation name on the atomization of the arguments (see example below). Individual members of this set are called *atoms*.

Definition: Some atoms are *defined* to be *true* and this subset of instances is called *facts*. (With respect to a Prolog program, an atom is a *fact* if and only if it is derivable from the knowledge base.)

An example: Assume **speak** $\in D$. Consider the relation schema **speak**$(language, human)$, where $E(language) = \{english\}$ and $E(human) = \{eli, dia\}$. We can construct the following *instances*:

1. **speak**(∅, ∅)
2. **speak**(∅, dia)
3. **speak**(∅, eli)
4. **speak**(english, ∅)
5. **speak**(english, dia)
6. **speak**(english, eli)
7. **speak**(english, [dia, eli])

Further, if we define **speak**(english, dia) to be *true* (e.g., if we list it in the knowledge base), **speak**(english, dia) is a *fact*. The atomization of 7) is {**speak**$< eng, dia >$, **speak** $<$

[2] Note, in examples, set valued arguments are written between '[' and ']', when necessary. In general, all arguments are sets but in this document, set markers are left out when the cardinality of a set is one.

$eng, eli >\}$.

Definition: A *typed multi-valued database* or *situation* g is an application which associates, to each relational symbol $r \in Rel$ of degree n and domain $[t_1, \ldots, t_n]$, an n-ary relation $\rho = g(r)$, which maps

1) $P(E(t_1)) \times \cdots \times P(E(t_n)) \rightarrow \{true, false, mixed, unknown\}$ if $\mathbf{r} \in D \cup IC \cup PC$ and maps

2) $\{< P_{sub}(E(t_1)), P_{sub}(E(t_2)), \ldots, P_{sub}(E(t_n)) > \mid \forall 1 \leq k \leq n,\ P_{sub}(E(t_k)) \subseteq P(E(t_k))$ and

$cardinality(P_{sub}(E(t_k))) = m, m \geq 2\} \rightarrow \{true, false, mixed, unknown\}$ if $\mathbf{r} \in R$, and maps

3) P(E(t)) $\rightarrow \{true, false, mixed\}$ if $\mathbf{r} \in S$.

In logic programming terms, a typed multi-valued database g is a logic program where multiple values are assigned to relations as defined below and where variables and constants are typed (e.g., `A` $\in$ `animal`). The inclusion relationships between types have been declared as described in 2.3, and the simple types associated with variables and constants are transparently compiled into incomplete types.

4.2.1 Distributive Relations

Distributive relations are relations which distribute a certain property to all individuals of a set. For example, in the statement "Eli and Dia speak English", the property "speak" distributes to both Eli and Dia and we may infer that the two statements "Eli speaks English" and "Dia speaks English" are both *true*. Distributive relations are divided into *many-to-many* and *one-to-many* relations.

Many-to-many Distributive Relations

A many-to-many relation **r** with attributes $t_1, t_2, \ldots, t_n$ has two attributes t_i and t_j $(1 \leq i, j \leq n)$ that are in a many-to-many relationship with one another. The atomization of an instance's arguments $\mathbf{r}(x_1, x_2, \ldots, x_n)$ is the product of the arguments $(x_1 \otimes x_2 \otimes \ldots \otimes x_n)$.

A *many-to-many* relation **r** evaluates to *true* (or *holds*) on a tuple $\vec{x}$ if and only if $\mathbf{r}(\vec{x})$ is a fact, and to *false* (or *fails*) if and only if $\mathbf{r}(\vec{x})$ is not a fact. An instance of a many-to-many distributive relation evaluates to *true*, *false*, or *mixed* with respect to its atomization as follows:

— If the relation holds on every member of the atomization, the instance evaluates to *true.*

— If the relation fails on every member of the atomization, the instance evaluates to *false.*

— If the relation holds on some and fails on the rest of the members of the atomization, the instance evaluates to *mixed.*

An example: Consider the following instance of the many-to-many distributive relation **speak**:

> **speak**([eli, dia], [english, spanish]) (= "Eli and Dia speak English and Spanish.")

The atomization of the arguments = {<eli, english>, <eli, spanish>, <dia, english>, <dia, spanish>} and the instance evaluates to:

a. *true* if applied to the knowledge base: { **speak**(dia, spanish),
speak(dia, english),
speak(eli, spanish),
speak(eli, english) }

b. *false* if applied to the knowledge base: { **speak**(jörg, english) }

c. *mixed* if applied to the knowledge base: { **speak**(dia, spanish),
speak(dia, english),
speak(eli, english) }

Formally: If **r** $\in D^{m-n}_{i,j}(m, n \geq 2), \rho$ maps $P(E(t_1)) \times P(E(t_2)) \times \cdots \times P(E(t_n)) \rightarrow \{true, false, mixed\}$ as follows:

1. $\rho(x_1, x_2, \ldots, x_n) = true \Longleftrightarrow$

   $\rho(y_1, y_2, \ldots, y_n) = true$

   for every tuple $< y_1, y_2, \ldots, y_n > \in x_1 \otimes x_2 \otimes \cdots \otimes x_n$

2. $\rho(x_1, x_2, \ldots, x_n) = false \Longleftrightarrow$

   $\rho(y_1, y_2, \ldots, y_n) = false$

   for every tuple $< y_1, y_2, \ldots, y_n > \in x_1 \otimes x_2 \otimes \cdots \otimes x_n$

3. $\rho(x_1, x_2, \ldots, x_n) = mixed$ otherwise

One-to-many Distributive Relations

A one-to-many distributive relation **r** with attributes $t_1, t_2, \ldots, t_n$ is a relation where an attribute t_i is in a one-to-many relationship with attribute t_j $(1 \leq i, j \leq n)$. Similarly to many-to-many distributive relations, the atomization of an instance's arguments $\mathbf{r}(x_1, x_2, \ldots, x_n)$ is the product of the arguments $(x_1 \otimes x_2 \otimes \ldots \otimes x_n)$.

For any given tuple $< y_1, \ldots, y_i, \ldots, y_j, \ldots, y_n > \in x_1 \otimes x_2 \otimes \ldots \otimes x_n$, there is only one y_i value that can satisfy the tuple (e.g., a person can only have been born in one specific place), therefore an instance $\mathbf{r}(x_1, x_2, \ldots, x_n)$ of a one-to-many distributive relation is only defined if the cardinality of $x_i = 1$. Consider the following example. Assume that the one-to-many distributive relation **born_in**(*country*, *people*) represents the concept of people being born in different countries. If the cardinality of *country* is unrestricted, the following relation is valid: **born_in**([norway, sweden], eli) and would mean something like "Eli was born in Norway *and* Sweden", but this is of course meaningless.

A one-to-many distributive relation **r** holds on a tuple $\vec{x}$ if and only if $\mathbf{r}(\vec{x})$ is a fact. In contrast to many-to-many relations, a relation fails on a tuple $\vec{x} =< x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n >$ if and only if there exists another fact that contradicts $\mathbf{r}(\vec{x})$, i.e., there exists a tuple $< x_1, \ldots, y_i, \ldots, x_j, \ldots, x_n >$, where $x_i \neq y_i$ such that $\mathbf{r}(x_1, \ldots, y_i, \ldots, x_j, \ldots, x_n)$ is a fact. Otherwise, the relation evaluates to *unknown* on $\vec{x}$, i.e., $\mathbf{r}(\vec{x})$ is not a fact and there is no other fact which contradicts $\mathbf{r}(\vec{x})$. (For example, if the statement "Ann lives in Grenoble" is a fact, the statement "Ann lives in Oslo" evaluates to *false*. If we have no factual information about Ann's whereabouts, the latter statement evaluates to *unknown*.) An instance $\mathbf{r}(\vec{x})$ of a one-to-many distributive relation evaluates to *true, false, unknown*, or *mixed* with respect to its atomization as follows:

— If the relation holds on every member of the atomization, $\mathbf{r}(\vec{x})$ evaluates to *true*.

— If the relation fails on every member of the atomization, $\mathbf{r}(\vec{x})$ evaluates to *false*.

— If the relation is unknown for one or more members of the atomization, $\mathbf{r}(\vec{x})$ evaluates to *unknown*.

— If the relation holds on some and fails on the rest of the members of the atomization, $\mathbf{r}(\vec{x})$ evaluates to *mixed*.

An example: Consider the following instance of the one-to-many distributive relation **born**:

born_in(finland, [eli, dia, jörg]) (= "Eli, Dia and Jörg were born in Finland.")

The atomization of the arguments = {<finland, eli>, <finland, dia>, <finland, jörg>} and the instance evaluates to:

a. *true* if applied to the knowledge base: { **born_in**(finland, eli),
born_in(finland, dia),
born_in(finland, jörg) }

b. *false* if applied to the knowledge base: { **born_in**(norway, eli),
born_in(uruguay, dia),
born_in(germany, jörg) }

c. *mixed* if applied to the knowledge base: { **born_in**(norway, eli),
born_in(finland, dia),
born_in(germany, jörg) }

d. *unknown* if applied to the knowledge base: { **born_in**(finland, eli),
born_in(germany, jörg) }

e. *unknown* if applied to the knowledge base: { **born_in**(finland, eli),
born_in(finland, jörg) }

f. *unknown* if applied to the knowledge base: { **born_in**(norway, eli),
born_in(germany, jörg) }

Formally: $\forall \mathbf{r} \in D_{i,j}^{1-m}$ $(m \geq 1)$, $\mathbf{r}(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n)$ is defined $\Longleftrightarrow$ cardinality of $x_i = 1$.

If $\mathbf{r} \in$ $D_{i,j}^{1-m}$ $(m \geq 1)$, ρ maps $P(E(t_1)) \times P(E(t_2)) \times \cdots \times P(E(t_n)) \rightarrow \{true, false, unknown, mixed\}$ as follows[3]:

1. $\rho(x_1, x_2, \ldots, x_n) = true \Longleftrightarrow$

   $\rho(y_1, y_2, \ldots, y_n) = true$

   for every tuple $< y_1, y_2, \ldots, y_n > \in x_1 \otimes x_2 \otimes \ldots \otimes x_n$

2. $\rho(x_1, x_2, \ldots, x_n) = false \Longleftrightarrow$

   for every tuple $< y_1, \ldots, y_i, \ldots, y_j, \ldots, y_n > \in x_1 \otimes x_2 \otimes \ldots \otimes x_n$

   $\exists z_i \in P(E(i))$, such that $z_i \neq y_i$ and

   $\rho(y_1, \ldots, y_{i-1}, z_i, y_{i+1}, \ldots, y_j, \ldots, y_n) = true$

[3] We assume that 'many-to-one' relationships are rewritten as 'one-to-many' relationships and treat these as one case.

3. $\rho(x_1, x_2, \ldots, x_n) = unknown \Longleftrightarrow$

   $\exists$ a tuple $< y_1, y_2, \ldots, y_n > \in x_1 \otimes x_2 \otimes \ldots \otimes x_n$ such that

   $\rho(y_1, y_2, \ldots, y_n) \neq true$

   $\rho(y_1, y_2, \ldots, y_n) \neq false$

4. $\rho(x_1, x_2, \ldots, x_n) = mixed$ otherwise

4.2.2 Inherently Collective Relations

Inherently collective relations are relations where a certain task is done collectively by a whole set of individuals, i.e., each set member is necessary but not sufficient for the successful completion of the task. For example, the statement "Ala, Brigitte and Dia lifted the heavy table" describes a situation where Ala, Brigitte and Dia lifted a heavy table *together*, and we cannot infer that, for example, the statement "Ala and Dia lifted the heavy table" is *true*. Since inherently collective relations apply to groups of individuals, an instance $\mathbf{r}(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n)$ of a relation **r** is only defined if cardinality of $x_j \geq 2$. Inherently collective relations are divided into one-to-many and many-to-many relations.

Many-to-many Inherently Collective Relations

A many-to-many inherently collective relation **r** with attributes $t_1, t_2, \ldots, t_n$ has two attributes t_i and t_j ($1 \leq i, j \leq n$) that are in a many-to-many relationship with one another. The atomization of an instance $\mathbf{r}(x_1, x_2, \ldots, x_n)$'s arguments is the one member set $\{< x_1, x_2, \ldots, x_n >\}$.

An instance $\mathbf{r}(\vec{x})$ of an inherently collective many-to-many relation evaluates to *true* or *false* as follows:

— If $\mathbf{r}(\vec{x})$ is a fact, $\mathbf{r}(\vec{x})$ evaluates to *true*.

— If $\mathbf{r}(\vec{x})$ is not a fact, $\mathbf{r}(\vec{x})$ evaluates to *false*.

An example: Consider the following instance of the many-to-many inherently collective relation **raise** (Say, a set of nuns raise a set of orphans and the raising is a collective job such that no child is being raised by any particular nun, and vice versa.):

> **raise**([nun1, nun2], [child1, child2, child3]) (="Nun1 and nun2 raised child1, child2, child3.")

The atomization of the arguments = {<[nun1, nun2], [child1, child2, child3]>} and the instance evaluates to:

a. *true* if applied to the knowledge base: { **raise**([nun1, nun2], [child1, child2, child3]) }

b. *false* if applied to the knowledge base: { **raise**([nun1, nun2, nun3], [child1, child2, child3]) }

c. *false* if applied to the knowledge base: { **raise**([nun1, nun2], [child1, child2, child3, child4]) }

d. *false* if applied to the knowledge base: { **raise**([nun3, nun4, nun5], [child1, child2, child3]) }

Formally: If **r** $\in IC^{m-n}(m, n \geq 2), \rho$ maps $P(E(t_1)) \times P(E(t_2)) \times \cdots \times P(E(t_n)) \rightarrow \{true, false\}$ as follows:

1. $\rho(x_1, x_2, \ldots, x_n) = true \Longleftrightarrow$

   $r(x_1, x_2, \ldots, x_n)$ is a fact

2. $\rho(x_1, x_2, \ldots, x_n) = false \Longleftrightarrow$

   $r(x_1, x_2, \ldots, x_n)$ is not a fact

One-to-many Inherently Collective Relations

A one-to-many inherently collective relation **r** with attributes $t_1, t_2, \ldots, t_n$ is a relation where attribute t_i is in a one-to-many relationship with attribute t_j $(1 \leq i, j \leq n)$. The atomization of an instance $\mathbf{r}(x_1, x_2, \ldots, x_n)$'s arguments is the one member set $\{< x_1, x_2, \ldots, x_n >\}$.

An instance $\mathbf{r}(\bar{x})$ of a one-to-many inherently collective relation evaluates to *true*, *false* or *unknown* as follows:

— If $\mathbf{r}(\bar{x})$ is a fact, $\mathbf{r}(\bar{x})$ evaluates to *true*.

— If there exists a fact $\mathbf{r}(x_1, \ldots, y_i, \ldots, x_j, \ldots, x_n)$ which contradicts $\mathbf{r}(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n)$, i.e, $x_i \neq y_i$, $\mathbf{r}(\bar{x})$ evaluates to *false*.

— If $\mathbf{r}(\bar{x})$ is not a fact and there is no other fact which contradicts $\mathbf{r}(\bar{x})$, $\mathbf{r}(\bar{x})$ evaluates to *unknown*.

An example: Consider the following instance of the one-to-many inherently collective relation **lift**:

lift(table, [ala, dia]) (= "Ala and Dia lifted the table.")

The atomization of the arguments = {<table, [ala, dia]>} and the instance evaluates to

a. *true* if applied to the knowledge base: { **lift**(table, [ala, dia]) }

b. *false* if applied to the knowledge base: { **lift**(table, [andrea, kaci]) }

c. *false* if applied to the knowledge base: { **lift**(table, [ala, dia, jörg]) }

d. *unknown* if applied to the knowledge base: { **lift**(bookcase, [jörg, allan]) }

Formally: $\forall \mathbf{r} \in IC^{1-m}_{i,j}\mathbf{r}(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n)$ is defined $\Longleftrightarrow$ cardinality of $x_i = 1$. If $\mathbf{r} \in IC^{1-m}_{i,j}$ $(m \geq 1)$, ρ maps $P(E(t_1)) \times P(E(t_2)) \times \cdots \times P(E(t_n)) \rightarrow \{true, false, unknown\}$ as follows:

1. $\rho(x_1, x_2, \ldots, x_n) = true \Longleftrightarrow$

   $r(x_1, x_2, \ldots, x_n)$ is a fact

2. $\rho(x_1, x_2, \ldots, x_n) = false \Longleftrightarrow$

   $\exists y_i \in P(E(t_i))$ such that $cardinality(y_i) = 1$ and $x_i \neq y_i$

   $\rho(x_1, \ldots, x_{i-1}, y_i, x_{i+1}, \ldots, x_j, \ldots, x_n) = true$

3. $\rho(x_1, x_2, \ldots, x_n) = unknown$ otherwise

4.2.3 Partially Collective Relations

Partially collective relations are also relations where a certain task is done collectively by a whole set of individuals, but where a subset of the original set may be sufficient to satisfy the relation. For example, if we define **lift** in our previous example to be a partially collective relation, we can infer that the statement "Dia and Ala lifted the heavy table" is true from the statement "Ala, Brigitte and Dia lifted the heavy table".

Many-to-many Partially Collective Relations

A many-to-many partially collective relation $\mathbf{r}$ with attributes $t_1, t_2, \ldots, t_n$ has two attributes, t_i and t_j $(1 \leq i, j \leq n)$ that are in a many-to-many relationship with one another. The atomization of an instance's arguments is the one member set $\{< x_1, x_2, \ldots, x_n >\}$.

An instance $\mathbf{r}(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n)$ of a many-to-many partially collective relation evaluates to *true* or *false* as follows.

— If there exists a fact, $\mathbf{r}(x_1, \ldots, y_i, \ldots, y_j, \ldots, x_n)$ such that x_i is a subset of y_i and x_j is a subset of y_j, $\mathbf{r}(x_1, \ldots, x_n)$ evaluates to *true*.

— If there is no fact, $\mathbf{r}(x_1, \ldots, y_i, \ldots, y_j, \ldots, x_n)$ such that x_i is a subset of y_i and x_j is a subset of y_j, $\mathbf{r}(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n)$ evaluates to *false*.

An example: Reconsider the nuns and children example above, but assume that **raise** is a partially collective relation instead of an inherently collective relation.

raise([nun1, nun2], [child1, child2, child3]) (= "Nun1 and nun2 raised child1, child2, child3.")

The atomization of the arguments is {<[nun1, nun2], [child1, child2, child3]>} and it evaluates to:

a. *true* if applied to the knowledge base: { **raise**([nun1, nun2], [child1, child2, child3]) }
b. *true* if applied to the knowledge base: { **raise**([nun1, nun2, nun3], [child1, child2, child3]) }
c. *true* if applied to the knowledge base: { **raise**([nun1, nun2], [child1, child2, child3, child4]) }
d. *false* if applied to the knowledge base: { **raise**([nun3, nun4, nun5], [child1, child2, child3]) }
e. *false* if applied to the knowledge base: { **raise**([nun2, nun4, nun5], [child1, child2, child3]) }

Formally: If **r** $\in PC^{m-n}_{i,j}$, ρ maps $P(E(t_1)) \times P(E(t_2)) \times \cdots \times P(E(t_n)) \to \{true, false\}$ as follows

1. $\rho(x_1, x_2, \ldots, x_n) = true \iff$

   $\exists y_i \in P(E(t_i)), y_j \in P(E(t_j))$ such that

   $x_i \subseteq y_i, x_j \subseteq y_j$ and

   $\rho(x_1, \ldots, x_{i-1}, y_i, x_{i+1}, \ldots, x_{j-1}, y_j, x_{j+1}, \ldots, x_n) = true$

2. $\rho(x_1, x_2, \ldots, x_n) = false$ otherwise

One-to-many Partially Collective Relations

A one-to-many partially collective relation **r** with attributes $t_1, t_2, \ldots, t_n$ is a relation where attribute t_i is in a one-to-many relationship with attribute t_j $(1 \leq i, j \leq n)$. The atomization of an instance $\mathbf{r}(x_1, x_2, \ldots, x_n)$'s arguments is the one member set $\{< x_1, x_2, \ldots, x_n >\}$. As with other one-to-many relations, $\mathbf{r}(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n)$ is only defined if the cardinality of $x_i = 1$.

An instance $\mathbf{r}(x_1, \ldots, x_i \ldots, x_j \ldots, x_n)$ of a one-to-many partially collective relation evaluates to ***true, false*** or ***unknown*** as follows.

— If there exists a fact, $\mathbf{r}(x_1, \ldots, x_i, \ldots, y_j, \ldots, x_n)$ such that x_j is a subset of y_j, $\mathbf{r}(x_1, \ldots, x_i, \ldots,$
$x_j, \ldots, x_n)$ evaluates to ***true***.

— If there exists a fact, $\mathbf{r}(x_1, \ldots, y_i, \ldots, y_j, \ldots, x_n)$ which contradicts $\mathbf{r}(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n)$, i.e., x_j is a subset of y_j, but x_i is different from y_i, $\mathbf{r}(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n)$ evaluates to ***false***.

— If $\mathbf{r}(x_1, \ldots, x_n)$ is not a fact and there is no other fact which contradicts it, $\mathbf{r}(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n)$ evaluates to ***unknown***.

An example: Consider the following instance of the one-to-many partially collective relation **meet**:

> **meet**(park, [ala, dia]) (= "Dia and Ala met in the park.")

The atomization of the arguments = {<park, [ala, dia]>} and the instance evaluates to:

a. *true* if applied to the knowledge base: { **meet**(park, [ala, dia]) }
b. *true* if applied to the knowledge base: { **meet**(park, [eli, ala, dia]) }
b. *false* if applied to the knowledge base: { **meet**(school, [ala, dia]) }
c. *false* if applied to the knowledge base: { **meet**(school, [eli, ala, dia]) }
d. *unknown* if applied to the knowledge base: { **meet**(park, [jörg, allan]) }

Formally: $\forall \mathbf{r} \in PC^{1-m}_{i,j}$ $(m \geq 1)$ $\mathbf{r}(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n)$ is defined $\Longleftrightarrow$ cardinality of $x_i = 1$.
If $\mathbf{r} \in PC^{1-m}_{i,j}$ $(m \geq 1)$, ρ maps $P(E(t_1)) \times P(E(t_2)) \times \cdots \times P(E(t_n)) \rightarrow \{true, false, unknown\}$ as follows

1. $\rho(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n) = true \Longleftrightarrow$

   $\exists y_j \in P(E(t_j))$ such that $x_j \subseteq y_j$ and

   $\rho(x_1, \ldots, x_i, \ldots, x_{j-1}, y_j, x_{j+1}, \ldots, x_n) = true$

2. $\rho(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n) = false \Longleftrightarrow$

   $\exists y_i \in P(E(t_i)), y_j \in P(E(t_j))$ such that $x_i \neq y_i$ and $x_j \subseteq y_j$ and

   $\rho(x_1, \ldots, x_{i-1}, y_i, x_{i+1}, \ldots, x_{j-1}, y_j, x_{j+1}, \ldots, x_n) = true$

3. $\rho(x_1, x_2, \ldots, x_n) = unknown$ otherwise

Comment

We can collapse the two classes of collective relations into one class if we introduce another logical value *yes_but* and make the partially collective relations that now evaluate to *true* evaluate to *yes_but*, while we keep the interpretations of *true*, *false*, and *unknown* as defined above for inherently collective relations.

4.2.4 Respective Relations

Unlike distributive and collective relations, respective relations do not reflect an inherent property of a given concept. The most common way of introducing a respective relation is to include the word "respectively" in a statement. Since respective relations are not introduced by a concept itself, both distributive and collective relations can participate in respective relations. Consider the following examples,

— The one-to-many distributive relation **born_in**(*country*, *human*); From the statement "Dia and Ala were born in Uruguay and Poland (respectively)" we may infer that the statements "Dia was born in Uruguay" and "Ala was born in Poland" are both *true*, and that, for example, the statements "Dia was born in Poland" and "Dia was born in Norway" are false.

— The one-to-many collective relation **meet**(*place*, *human*); From the statement "Brigitte and Edwin and Eli and Jörg met in the park and at school respectively" we may infer that the statements "Brigitte and Edwin met in the park" and "Eli and Jörg met at school" are true.

— The many-to many distributive relation **speak**(*language*, *human*); From the statement "Ala and Dia speak Polish and Spanish respectively" we may infer that the statements "Ala speaks Polish" and"Dia speaks Polish" are true.

It is the job of the natural language front end to detect the words that introduce respective relations.

Since all the relations discussed earlier can participate in respective relations, we need to create a corresponding "respective" symbol for each one of them such that the database system can distinguish between respective and non-respective queries. The **r_resp** symbol is only for identification purposes in order to generate the correct atomization. The query is evaluated with respect to **r**. In this document, we simply create new symbols by attaching the suffix **_resp** to the original symbol **r** (e.g., **speak** becomes **speak_resp**).

There is one class of respective relations for each class of distributive and collective relations and we name these *respective distributive, respective inherently collective* and *respective partially collective*. In addition to new "respective" constraints, respective relations inherit the argument properties of the original relation (e.g., a one-to-many relation is still one-to-many). Consequently, each respective class is divided into many-to-many and one-to-many relations.

Regardless of origin, an instance $\mathbf{r}(x_1, x_2, \ldots, x_n)$, of a respective relation is only defined if all arguments have the same number of elements, i.e., $\exists m$ such that $m \geq 2$ and $\forall 1 \leq k \leq n$, $x_k = \{x_{k_1}, x_{k_2}, \ldots, x_{\ m}\}$. If domain($\mathbf{r}$) = $[t_1, t_2, \ldots, t_n]$, then $x_{k_i} \in P(E(t_k))$ $(1 \leq k \leq n, 1 \leq i \leq m)$. The atomization of $\mathbf{r}(\vec{x})$'s arguments is $x_{1_1} \otimes x_{2_1} \otimes \ldots \otimes x_{n_1} \cup x_{1_2} \otimes x_{2_2} \otimes \ldots \otimes x_{n_2} \cup \ldots \cup x_{1_m} \otimes x_{2_m} \otimes \ldots \otimes x_{n_m}$. We introduce the following notation $x_1 \circ x_2 \circ \ldots \circ x_n = x_{1_1} \otimes x_{2_1} \otimes \ldots \otimes x_{n_1} \cup x_{1_2} \otimes x_{2_2} \otimes \ldots \otimes x_{n_2} \cup \ldots \cup x_{1_m} \otimes x_{2_m} \otimes \ldots \otimes x_{n_m}$.

Many-to-many Respective Relations

A many-to-many respective relation $\mathbf{r}$ with attributes $t_1, t_2, \ldots, t_n$ has two attributes, t_i and t_j $(1 \leq i, j, \leq n)$ that are in a many-to-many relationship with one another. The atomization of an instance $\mathbf{r}(x_1, x_2, \ldots, x_n)$'s arguments, is $x_1 \circ x_2 \circ \ldots \circ x_n$.

A many-to-many respective *partially collective relation* $\mathbf{r}$ holds on a tuple, $< x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n >$, if there exists a fact $\mathbf{r}(x_1, \ldots, y_i, \ldots, y_j, \ldots, x_n)$ such that x_i is a subset of y_i and x_j is a subset of y_j. It fails on a tuple $< x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n >$ if there is no fact, $\mathbf{r}(x_1, \ldots, y_i, \ldots, y_j, \ldots, x_n)$ such that x_i is a subset of y_i and x_j is a subset of y_j.

All other many-to-many respective relations $\mathbf{r}$ hold on a tuple $\vec{x}$ if and only if $\mathbf{r}(\vec{x})$ is a fact, and fail if and only if $\mathbf{r}(\vec{x})$ is not a fact. An instance $\mathbf{r}(\vec{x})$ of a many-to-many respective relation evaluates to *true*, *false*, or *mixed* with respect to its atomization as follows:

— If the relation holds on every member of the atomization, $\mathbf{r}(\vec{x})$ evaluates to *true*.

— If the relation fails on every member of the atomization, $\mathbf{r}(\vec{x})$ evaluates to *false*.

— If the relation holds on some and fails on the rest of the members of the atomization, $\mathbf{r}(\vec{x})$ evaluates to *mixed*.

An example: Consider the following instance of the many-to-many respective relation **speak_resp**:

speak_resp([[eli, jörg], [veronica, dia]], [german, spanish]) (= "Eli and Jörg and Veronica and Dia speak German and Spanish respectively.")

The atomization of the arguments = {<eli, german>, <jörg, german>, <veronica, spanish> <dia, spanish>} and the instance evaluates to:

a. *true* if applied to the knowledge base: {**speak**(eli, german),
speak(jörg, german),
speak(veronica, spanish),
speak(dia, spanish)}

b. *false* if applied to the knowledge base: { **speak**(brigitte, german) }

c. *mixed* if applied to the knowledge base: { **speak**(dia, spanish),
speak(jörg, german),

One-to-many Respective Relations

A one-to-many respective relation **r** with attributes $t_1, t_2, \ldots, t_n$ is a relation where attribute t_i is in a one-to-many relation with attribute t_j. An instance $\mathbf{r}(\vec{x})$ of a one-to-many respective relation is only defined if the cardinality of $x_{i_p} = 1 \ \forall 1 \leq p \leq m$. The atomization of an instance $\mathbf{r}(x_1, x_2, \ldots, x_n)$'s arguments is $x_1 \circ x_2 \circ \ldots \circ x_n$.

A one-to-many respective relation **r** holds on a tuple $\vec{x} =< x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n >$ if and only if $\mathbf{r}(\vec{x})$ is a fact. It fails on $\vec{x}$ if and only if there exists another fact that contradicts $\mathbf{r}(\vec{x})$, i.e., there exists a tuple, $< x_1, \ldots, y_i, \ldots, x_j, \ldots, x_n >$, where $x_i \neq y_i$, and $\mathbf{r}(x_1, \ldots, y_i, \ldots, x_j, \ldots, x_n)$ is a fact. Otherwise, the relation evaluates to *unknown* on $\vec{x}$, i.e., if $\mathbf{r}(\vec{x})$ is not a fact and there is no other fact which contradicts it. An instance $\mathbf{r}(\vec{x})$ of a one-to-many respective relation evaluates to *true, false, unknown*, or *mixed* with respect to its atomization as follows:

— If the relation holds on every member of the atomization, $\mathbf{r}(\vec{x})$ evaluates to *true*.

— If the relation fails on every member of the atomization, $\mathbf{r}(\vec{x})$ evaluates to *false*.

— If the relation is unknown for one or more members of the atomization, $\mathbf{r}(\vec{x})$ evaluates to *unknown*.

— If the relation holds on some and fails on the rest of the members of the atomization, $\mathbf{r}(\vec{x})$ evaluates to *mixed*.

An example: Consider the following instance of the one-to-many respective relation **earn_resp**:

earn_resp([1000, 1100], [brigitte, ann]) (= "Brigitte and Ann earn $1000 and $1100 respectively.")

The atomization of the arguments = {<1000, brigitte>, <1100, ann>} and the instance evaluates to

a. *true* if applied to the knowledge base: { **earn**(1000, brigitte),
earn(1100, ann) }

b. *false* if applied to the knowledge base: { **earn**(900, brigitte),

earn(1200, ann) }

c. *mixed* if applied to the knowledge base: { **earn**(1000, brigitte),
earn(1200, ann) }

d. *unknown* if applied to the knowledge base: { **earn**(1000, allan) }

Formally:

$\forall$**r_resp** $\in R$, $\mathbf{r}(x_1, x_2, \ldots, x_n)$ is defined $\Longleftrightarrow \exists m \geq 2$ such that $\forall 1 \leq k \leq n$, $x_k = \{x_{k_1}, x_{k_2}, \ldots, x_{k_m}\}$ and $x_{k_i} \in P(E(t_k))$, $\forall 1 \leq i \leq m$.

Respective Distributive Relations

If **r_resp** $\in R$ and **r** $\in D_{i,j}^{m-n}$ $(m, n \geq 2)$, ρ maps $< P_{sub}(E(t_1)), P_{sub}(E(t_2)), \ldots, P_{sub}(E(t_n)) >$, where $\forall 1 \leq k \leq n$, $P_{sub}(E(t_k)) \subseteq P(E(t_k)) \rightarrow \{true, false, mixed\}$ as follows

1. $\rho(x_1, x_2, \ldots, x_n) = true \Longleftrightarrow$

   $\rho(y_1, y_2, \ldots, y_m) = true$

   for every tuple $< y_1, y_2, \ldots, y_m > \in x_1 \circ x_2 \circ \cdots \circ x_n$

2. $\rho(x_1, x_2, \ldots, x_n) = false \Longleftrightarrow$

   $\rho(y_1, y_2, \ldots, y_m) = false$

   for every tuple $< y_1, y_2, \ldots, y_m > \in x_1 \circ x_2 \circ \cdots \circ x_n$

3. $\rho(x_1, x_2, \ldots, x_n) = mixed \Longleftrightarrow$

   $\exists$ a tuple $< y_1, y_2, \ldots, y_m > \in x_1 \circ x_2 \circ \cdots \circ x_n$ and

   $\exists$ a tuple $< z_1, z_2, \ldots, z_m > \in x_1 \circ x_2 \circ \cdots \circ x_n$ such that

   $\rho(y_1, y_2, \ldots, y_m) = true$ and

   $\rho(z_1, z_2, \ldots, z_m) = false$

If **r_resp** in R and **r** $\in D_{i,j}^{1-m}$ $(m \geq 1)$, ρ maps $< P_{sub}(E(t_1)), P_{sub}(E(t_2)), \ldots, P_{sub}(E(t_n)) >$, where $\forall 1 \leq k \leq n$, $P_{sub}(E(t_k)) \subseteq P(E(t_k)) \rightarrow \{true, false, mixed, unknown\}$ as follows

1. $\rho(x_1, x_1, \ldots, x_n) = true \Longleftrightarrow$

   $\rho(y_1, y_2, \ldots, y_m) = true$

   for every tuple $< y_1, y_2, \ldots, y_m > \in x_1 \circ x_2 \circ \cdots \circ x_n$

2. $\rho(x_1, x_2, \ldots, x_n) = \mathit{false} \Longleftrightarrow$

   for every tuple $< y_1, \ldots, y_i, \ldots, y_j, \ldots, y_m > \in x_1 \circ x_2 \circ \cdots \circ x_n$

   $\exists z_i \in P(E(t_i))$ such that

   $\rho(y_1, \ldots, y_{i-1}, z_i, y_{i+1}, \ldots, y_j, \ldots, y_m) = \mathit{true}$

3. $\rho(x_1, x_2, \ldots, x_n) = \mathit{unknown} \Longleftrightarrow$

   $\exists$ a tuple $< y_1, y_2, \ldots, y_m > \in x_1 \circ x_2 \circ \cdots \circ x_n$ such that

   $\rho(y_1, y_2, \ldots, y_m) = \mathit{unknown}$

4. $\rho(x_1, x_2, \ldots, x_n) = \mathit{mixed}$ otherwise

Respective Inherently Collective Relations

If **r_resp** in R and $\mathbf{r} \in IC^{m-n}_{i,j}$ $(m, n \geq 2)$, ρ maps $< P_{sub}(E(t_1)), P_{sub}(E(t_2)), \ldots, P_{sub}(E(t_n)) >$, where $\forall 1 \leq k \leq n$, $P_{sub}(E(t_k)) \subseteq P(E(t_k)) \rightarrow \{\mathit{true}, \mathit{false}, \mathit{mixed}\}$ as follows

1. $\rho(x_1, x_2, \ldots, x_n) = \mathit{true} \Longleftrightarrow$

   for every tuple $< y_1, y_2, \ldots, y_m > \in x_1 \circ x_2 \circ \cdots \circ x_n$

   $\mathbf{r}(y_1, y_2, \ldots, y_n)$ is a fact

2. $\rho(x_1, x_2, \ldots, x_n) = \mathit{false} \Longleftrightarrow$

   for every tuple $< y_1, y_2, \ldots, y_m > \in x_1 \circ x_2 \circ \cdots \circ x_n$

   $\mathbf{r}(y_1, y_2, \ldots, y_n)$ is not a fact

3. $\rho(x_1, x_2, \ldots, x_n) = \mathit{mixed}$ otherwise

If **r_resp** in R and $\mathbf{r} \in IC^{1-m}_{i,j}$ $(m \geq 1)$, ρ maps $< P_{sub}(E(t_1)), P_{sub}(E(t_2)), \ldots, P_{sub}(E(t_n)) >$, where $\forall 1 \leq k \leq n$, $P_{sub}(E(t_k)) \subseteq P(E(t_k)) \rightarrow \{\mathit{true}, \mathit{false}, \mathit{mixed}, \mathit{unknown}\}$ as follows

1. $\rho(x_1, x_2, \ldots, x_n) = \mathit{true} \Longleftrightarrow$

   for every tuple $< y_1, y_2, \ldots, y_m > \in x_1 \circ x_2 \circ \cdots \circ x_n$

   $\rho(y_1, y_2, \ldots, y_n) = \mathit{true}$

2. $\rho(x_1, x_2, \ldots, x_m) = \mathit{false} \Longleftrightarrow$

   for every tuple $< y_1, \ldots, y_i, \ldots, y_j \ldots, y_m > \in x_1 \circ x_2 \circ \cdots \circ x_n$

   $\exists z_i \in P(E(t_i))$ such that $y_i \in z_i$ and

   $\rho(y_1, \ldots, y_{i-1}, z_i, y_{i+1}, \ldots, y_j, \ldots, y_m) = \mathit{true}$

3. $\rho(x_1, x_2, \ldots, x_n) = unknown \iff$

   $\exists$ a tuple $< y_1, y_2, \ldots, y_m > \in x_1 \circ x_2 \circ \cdots \circ x_n$ such that

   $\rho(y_1, y_2, \ldots, y_m) \neq true$ and

   $\rho(y_1, y_2 1, \ldots, y_m) \neq false$

4. $\rho(x_1, x_2, \ldots, x_n) = mixed$ otherwise

Respective Partially Collective Relations

If **r_resp** in R and $\mathbf{r} \in PC_{i,j}^{m-n}$ $(m, n \geq 2)$, ρ maps $< P_{sub}(E(t_1)), P_{sub}(E(t_2)), \ldots, P_{sub}(E(t_n)) >$, where $\forall 1 \leq k \leq n$, $P_{sub}(E(t_k)) \subseteq P(E(t_k)) \rightarrow \{true, false, mixed\}$

1. $\rho(x_1, x_2, \ldots, x_n) = true \iff$

   for every tuple $< y_1, \ldots, y_i, \ldots, y_j, \ldots, y_m > \in x_1 \circ x_2 \circ \cdots \circ x_n$

   $\exists z_i \in P(E(t_i)), z_j \in P(E(t_j))$ such that

   $y_i \subseteq z_i, y_j \subseteq z_j$ and

   $\rho(y_1, \ldots, y_{i-1}, z_i, y_{i+1}, \ldots, y_{j-1}, z_j, y_{j+1}, \ldots, x_n) = true$

2. $\rho(x_1, x_2, \ldots, x_n) = false \iff$

   for every tuple $< y_1, \ldots, y_i, \ldots, y_j, \ldots, y_m > \in x_1 \circ x_2 \circ \cdots \circ x_n$

   $\nexists z_i \in P(E(t_i)), z_j \in P(E(t_j))$ such that

   $z_i, z_j \neq \emptyset$ and $y_i \subseteq z_i, y_j \subseteq z_j$ and

   $\rho(y_1, \ldots, y_{i-1}, z_i, y_{i+1}, \ldots, y_{j-1}, z_j, y_{j+1}, \ldots, y_n) = true$

3. $\rho(x_1, x_2, \ldots, x_n) = mixed$ otherwise

If **r_resp** in R and $\mathbf{r} \in PC_{i,j}^{1-m}$ $(m \geq 1)$, ρ maps $< P_{sub}(E(t_1)), P_{sub}(E(t_2)), \ldots, P_{sub}(E(t_n)) >$, where $\forall 1 \leq k \leq n$, $P_{sub}(E(t_k)) \subseteq P(E(t_k)) \rightarrow \{true, false, mixed, unknown\}$ as follows

1. $\rho(x_1, x_2, \ldots, x_n) = true \iff$

   for every tuple $< y_1, \ldots, y_i, \ldots, y_j, \ldots, y_m > \in x_1 \circ x_2 \circ \cdots \circ x_n$

   $\exists z_j \in P(E(t_j))$ such that $y_j \subseteq z_j$ and

   $\rho(y_1, \ldots, y_i, \ldots, y_{j-1}, z_j, y_{j+1}, \ldots, y_n) = true$

2. $\rho(x_1, x_2, \ldots, x_n) = false \iff$

   for every tuple $< y_1, \ldots, y_i, \ldots, y_j, \ldots, y_m > \in x_1 \circ x_2 \circ \cdots \circ x_n$

   $\exists z_i \in P(E(t_i)), z_j \in P(E(t_j))$

   such that $z_i \neq y_i$ and $y_j \subseteq z_j$ and

   $\rho(y_1, \ldots, y_{i-1}, z_i, y_{i+1}, \ldots, y_{j-1}, z_j, y_{j+1}, \ldots, y_n) = true$

3. $\rho(x_1, x_2, \ldots, x_n) = unknown \iff$

   $\exists$ a tuple $< y_1, y_2, \ldots, y_m > \in x_1 \circ x_2 \circ \cdots \circ x_n$

   $\rho(y_1, y_2, \ldots, y_m) \neq true$ and

   $\rho(y_1, y_2, \ldots, y_m) \neq false$ and

4. $\rho(x_1, x_2, \ldots, x_n) = mixed$ otherwise

4.2.5 Single Attribute Relations

Single attribute relations are relations with only one attribute t. Single attribute relations are often introduced by the verb *to be* as in, for example, "Pepe and Alicja are students". The property ***student*** distributes to all members of the set and we may infer that the two statements "Pepe is a student" and "Alicja is a student" are both *true*.

The atomization of an instance **r**(x)'s argument, where $x \in P(E(t))$ such that $x = \{x_1, x_2, \ldots x_n\}$, is $\{< x_1 >, < x_2 >, \ldots, < x_n >\}$, where $x_k \in E(t)$.

A single attribute relation **r** evaluates to *true* on a tuple x_k if and only if $\mathbf{r}(x_k)$ is a fact and fails if and only if $\mathbf{r}(x_k)$ is not a fact. An instance of a single attribute relation evaluates to ***true***, ***false*** or ***mixed*** with respect to its atomization as follows.

— If the relation holds on every member of the atomization, the instance evaluates to ***true***.

— If the relation fails on every member of the atomization, the instance evaluates to ***false***.

— If the relation holds on some and fails on the rest of the members of the atomization, the instance evaluates to ***mixed***.

An example: Consider the following instance of the single attribute relation **student**:

student([pepe, leslie]) (= "Pepe and Leslie are students".)

The atomization of the argument = {<pepe>, <leslie>} and the instance evaluates to:

a. *true* if applied to the knowledge base: { **student**(pepe),
student(leslie) }

b. *false* if applied to the knowledge base: { **student**(alicja),
student(brigitte) }

c. *mixed* if applied to the knowledge base: { **student**(alicja),
student(leslie) }

Formally: $\forall \mathbf{r} \in S$, ρ maps $P(E(t)) \rightarrow \{true, false, mixed\}$ as follows:

1. $\rho(x) = \boldsymbol{true} \Longleftrightarrow$

   $\rho(x_k) = \boldsymbol{true}$

   for every tuple $< x_k > \in \{< x_1 >, < x_2 >, \ldots, < x_n >\}$ $(1 \leq k \leq n)$

2. $\rho(x) = false \Longleftrightarrow$

   $\rho(x_k) = false$

   for every tuple $< x_k > \in \{< x_1 >, < x_2 >, \ldots, < x_n >\}$ $(1 \leq k \leq n)$

3. $\rho(x) = mixed$ otherwise

Comment

We said above that the word *respectively* introduces respective relations, but this is not the case with single attribute relations. Consider the following example,

> Pepe and Fred are student and professor respectively.

Bad style considerations aside, this sentence introduces *two* relations, namely **student** and **professor**, which one has to tie together with for example, the logic operator *and* (see 4.3), i.e., the above sentence can be represented:

> $and(student(pepe), professor(fred))$

Consequently, the parser must recognize single attribute relations in order to produce an appropriate representation.

4.3 Rigorous Definition of A Multiple-Valued Typed Query Language

4.3.1 Syntax

Our query language has three kinds of expressions: t-typed terms, formulae, and integer terms.

A *t-typed term* s can take any of the following forms:

1. k, where $k \in K$ and $type(k)t \in T$

2. x, where $x \in X$ and $type(x)t \in T$

3. $those(x, t, e)$, where $x \in X$ and $type(x) \in T$, e is a statment formula

4. $[k_1, \ldots, k_n]$ where, $k_i \in K$ and $\forall 1 \leq i \leq n$ $type(k_i) = t \in T$

A *statement formula* e can take any of the following forms:

1. $for(s, x, t, e_1, e_2)$, where $s, x \in X$, $t \in T$, e_1, e_2 are statement formulae

2. $r(s_1, s_2, \ldots, s_n)$ where $r \in Rel$, $degree(r) = n$ and $domain(r) = [t_1, t_2, \ldots, t_n]$ such that $type(s_i) = t_i \in T$

3. $and(e_1, e_2, \ldots, e_n)$, where $\forall 1 \leq i \leq n$ e_i are statement formulae

4. $presupp(e_1, e_2)$, where e_1, e_2 are statement formulae

5. $not(e)$, where e is a statement formula

6. $equal(n_1, n_2)$, where n_1, n_2 are integer terms

7. $greater\_than(n_1, n_2)$, where n_1, n_2 are integer terms

An *integer term* n can take any of the following forms:

1. j, where $j \in \mathcal{N} > 0$

2. $cardinality(s)$, where s is a t-typed formula and $type(s) = t \in T$

4.3.2 Semantics

Definition: An occurrence of a variable x in an expression f is said to be *free* if it does not occur inside a sub-expression of the form: $for(s, x, t, \epsilon_1, \epsilon_2)$ or $those(x, t, \epsilon)$ of f. An expression which contains no free variable occurrences is said to be *closed*.

Definition: In a well defined situation g (see p.41), a closed formula will have a truth value ($true, false, unknown, mixed, pointless$), a closed t-typed term where $t \in T$ will have a subset of $E(t)$ as its value and a closed integer term will nave an integer as its value.

We can now define the value of a closed expression in a given situation g.
The value $val(s)$ of a closed t-typed term s, where $t \in T$ is defined by:

1. if $s = k$ with $k \in I$, then
   $val(s) = \{k\}$

2. if s is an enumerated set E (i.e., a t-typed term of type 4. above), then
   $val(s) = E$

3. if $s = those(x, t, \epsilon)$ where $type(x) = t$, then $val(s) = \{y \in E(t)$ such that $val(\epsilon_{x \leftarrow \{y\}}) = true\}$, where $\epsilon_{x \leftarrow F}$ represents the formula ϵ in which every free occurrence of x has been replaced by the enumeration of the set F

The value $val(\epsilon)$ of a closed statement formula ϵ is defined as follows:

1. if $\epsilon = for(s, x, t, \epsilon_1, \epsilon_2)$ then
   $val(\epsilon) = val(\epsilon_2)$ with $s = val(those(x, t, \epsilon_1))$ where every free occurrence of s in ϵ_2 has been replaced by the enumeration of $val(s)$.

2. if $\epsilon = r(s_1, s_2, \ldots, s_n)$ then
   $val(\epsilon) = \rho(val(s_1), val(s_2), \ldots, val(s_n))$, where $\rho = g(r)$

3. if $\epsilon = and(\epsilon_1, \epsilon_2, \ldots, \epsilon_n)$,
   2.1. $val(\epsilon) = true \iff \forall 1 \leq i \leq n\ val(\epsilon_i) = true$
   2.2. $val(\epsilon) = false \iff \forall 1 \leq i \leq n\ val(\epsilon_i) = false$
   2.3. $val(\epsilon) = mixed \iff \forall 1 \leq i \leq n\ val(\epsilon_i) = true$ or $val(\epsilon_i) = false$ and neither 2.1 nor 2.2 holds
   2.4. $val(\epsilon) = unknown \iff \exists 1 \leq i \leq n\ val(\epsilon_i) = unknown$
   2.5. $val(\epsilon) = pointless \iff \exists 1 \leq i \leq n\ val(\epsilon_i) = pointless$

4. if $\epsilon = presupp(\epsilon_1, \epsilon_2)$,
   if $val(\epsilon_1) = true$, then $val(\epsilon) = val(\epsilon_2)$,
   if $val(\epsilon_1) = false$, then $val(\epsilon) = pointless$,
   otherwise, $val(\epsilon) = val(\epsilon_1)$.

5. if $\epsilon = not(\epsilon_1)$,
   if $val(\epsilon_1) = true$, then $val(\epsilon) = false$,
   if $val(\epsilon_1) = false$, then $val(\epsilon) = true$,
   otherwise $val(\epsilon) = val(\epsilon_1)$.

6. if $\epsilon = equal(n_1, n_2)$,
   if $val(n_1) = val(n_2)$, then $val(\epsilon) = true$,
   otherwise $val(\epsilon) = false$.

7. if $\epsilon = greater\_than(n_1, n_2)$,
   if $val(n_1) > val(n_2)$, then $val(\epsilon) = true$,
   otherwise, $val(\epsilon) = false$.

The value $val(n)$ of a closed integer term n is defined by:

1. if $n = j$, where j > 0, then
   $val(n) = j$

2. if $n = card(s)$ then,
   $val(n)$ = number of elements in the set s

Comment on Our Definition of Logical 'and'

Our definition of the logical operator *and* (p.59) differs from what is usual in logic. Consider the following statements:

a. "Eric and Martin earn $1000."
   = **earn(1000, [eric, martin])**

b. "Eric and Martin earn $1000 and $2000 respectively."
   = **earn_resp([1000, 2000], [eric, martin])**

c. "Eric earns $1000 and martin earns $1000."
   = **and(earn(1000, eric), earn(1000, martin))**

with respect to the following knowledge base:

earn{<$1000, eric>, <$1001, martin>}

Assume **earn** $\in D^{1-m}_{i,j}$. If the value of *and* is defined as

$and(e_1, e_2) = min(val(e_1, e_2))$
where *true* > *unknown* > *false* > *mixed* > *pointless* (say),

which is common in logic, the above statements evaluate to different values: a.) and b.) evaluate to *mixed*, while c.) evaluates to *false*! In a consistent system, all three statements should evaluate to the same truth value, and therefore the definition of *and* was written such that it is consistent with the definition of distributive and respective relations. With our definition for 'and', all three statements evaluate to *mixed*.

Chapter 5

American Sign Language

This chapter provides a brief overview of the linguistic structure of ASL. The chapter is organized as follows: Section 1 introduces ASL's main building block—the sign. In Section 2, we discuss various ASL strategies for cutting the number of words in a sentence, while maintaining comprehension, and in Section 3 we examine ASL's basic sentence structure. Section 4 discusses ASL verbs. In sections 5 and 6, we discuss the concept of time. The first three sections are concerned with isolated sentences, while the last three are concerned with the ASL discourse.

We collected the material in this chapter from the following sources: [27, 37, 59, 34, 22]. The examples (sometimes modified a little) are taken from [27], unless otherwise stated.

5.1 What is a sign?

A sign is defined by the following four parameters: *hand shape*—ASL has 36 distinct hand shapes (see [59, p.22] for a good illustration), *hand position*—relative to the body, *hand movement*—path specification, local movement (e.g., opening or closing hand, flicking a finger) and speed of delivery, and *palm direction*.

Signs can be either one handed or two handed. Which hand the signer uses for one handed signs depends on the signer's handedness. Two handed signs may have one or two *active* (i.e., moving) hands[1]. If both hands are active, the sign must be *symmetric*, i.e., the hand shapes must be the same on both hands, and the movement must be identical for both hands or one hand moves like a mirror image of the other. If only one hand is active, the symmetry condition does not apply.

[1] Also called dominant hand.

Some signs become unintelligible if the signer deviates from the specification, while others change meaning. For example, the sign UNDERSTAND[2] is usually signed with an extended index finger. By substituting the little finger for the index finger, the signer changes the meaning to "understand a little". Also, puns and many art signs are made by deliberately changing one or more of a sign's parameters (see for example, [34]). Some signs are loosely defined for some parameters and we will return to this in section 5.4, where we discuss different verb types in ASL.

The neutral signing space extends like a bubble in front of the signer with boundaries as follows: from the top of the head to the hip, from shoulder to shoulder and approximately an arm's length in front. Normally, all signs are made within the neutral signing space but the signing space may be enlarged, if signing to a large audience, or confined, for secretive purposes.

5.2 Economization

It takes roughly twice as long to produce a single ASL sign as an average English word, yet the information transmission rate is the same for both languages. Consequently, ASL must use many fewer words than English. The basic technique is to eliminate words and phrases with little semantic meaning; **articles, interjections, expletives** (dummy subjects), **idle chatter** and **linking verbs** (e.g., any form of "to be") are always deleted. Consider the following examples,

The boy is *a* star. = BOY FAMOUS.

Gee, you're beautiful! = BEAUTIFUL YOU.

There is nobody home. = HOME PERSON NONE.

I don't want to be personal, but you are a nice person = NICE PERSON, YOU.

The car *has* a flat tire. = CAR FLAT TIRE.

All these examples demonstrate elimination of linking verbs. In addition, the first example shows elimination of articles, the second demonstrates elimination of interjections, the third illustrates deletion of expletives and the fourth shows deletion of idle chatter.

ASL uses **conjunctions** only if the conjunction disambiguates or supplies additional information (also see section 5.3.5 on subordination). Consider the following examples,

[2] Following normal convention, English glosses for ASL signs are written in UPPER CASE.

I bought some apples *and* Coke. = ME BUY APPLE COKE.

I left *because* I had an appointment = ME LEAVE BECAUSE APPOINTMENT.

In the first example, "and" can be left out without distorting the meaning of the sentence. But leaving out "because" in the second example, would leave us with a sentence that differs in meaning from the original: ME LEAVE APPOINTMENT. = "I left for my appointment." therefore ASL includes BECAUSE in this type of sentences.

ASL substitutes a single sign for many English phrases, for example,

through the use of, by means of = WITH

it is doubtful that, if it should = MAYBE

the reason is that, owing to the fact that, owing to the fact that = BECAUSE

ASL uses **prepositions** if the preposition represents an idea that cannot be determined by context alone, for example,

Everyone sit around the table = SIT THERE TABLE.

Put the umbrella under the seat = PUT UMBRELLA UNDER SEAT.

The first example has only one common sense interpretation so the preposition is eliminated (if you want your guests to sit *on* the table, you would have to use a preposition). In the second example, the preposition makes it clear to put the umbrella "*under* the seat" as opposed to "*on* the seat".

5.3 Basic Sentence Structures

Sign order in ASL is still a topic of much discussion; some linguists (e.g., Fisher) argue that the underlying sign order is Subject-Verb-Object, while others (e.g., Friedman) argue for free sign order (for different analyses see, for example, [21, 22, 23, 27]). From a computational point of view, sign order is largely an implementation issue and since sign order does not affect the algorithms described in Chapter 6, we quite arbitrarily chose to follow Isenhath's analysis in our attempt to familiarize the reader with the basic sentence structures in ASL. We shall later see how these basic patterns get altered in a discourse situation (Sections 5.4 and 5.4.3).

5.3.1 Declarative Sentences

The simplest sentence patterns are similar in ASL and English: subject + verb (+ object). For example,

CAROL SWIM. (Carol is swimming.)

DOG BITE MAN (The dog bit the man.)

But as we shall see below, a well formed sentence in ASL may also look quite different from a well formed sentence in English; it may include several repetitions of a main verb or a noun, it may be verb-less or it may not have overt subject and/or object.

As mentioned above, linking verbs are deleted since they don't supply much semantic information. Also, such sentences usually only have one plausible interpretation. Elimination of linking verbs leaves the sentence with subject and a subject complement—a noun or adjectival phrase. For example,

CAR FLAT TIRE. (The car has a flat tire.)

ANN DOCTOR. (Ann is a doctor.)

SUMMER HOT. (The summer is hot.)

Some of the techniques for pluralization of nouns (also see 5.3.7) affect the surface structure of a sentence. One method (used with multi-directional verbs, see section 5.4.2) is reduplication of the sentence's *main verb.* The signer repeats the main verb three or more times, each time at a different spatial location. Consider the following example,

STUDENT CONVENE (move hands) CONVENE (move hands) CONVENE VANCOUVER.
(The students gathered in Vancouver.)

Each change in location symbolizes another person gathering. Two repetitions mean that two students gathered. Three repetitions can mean either three or many students gathered. A small number of nouns can be pluralized by reduplication of the noun itself, for example, TEA, BOOK, CHAIR, PAPER, TABLE. Again, two repetitions mean two items and three repetitions can mean three or many items.

Certain sentences do not have overt subjects and/or objects (both direct and indirect). This type of sentence is discussed together with verbs in section 5.4.

Topicalization

ASL adheres to the principle of "strongest emphasized information first" and is considered a "topic-prone language", where "topic" means the subject of conversation. Following this principle, ASL tends to structure a sentence such that information that is new, with respect to the current discourse, is presented first, and what is already known follows. As a consequence, topicalization is much more common in ASL than in English, but many other spoken languages with freer word order than English (e.g., Slavic languages) may also structure sentences such that new or emphasized information appears first.

In a non-topicalized sentence, the subject comes first, while in a topicalized sentence, another part of the sentence is moved to the front. In addition to being first, the topic is marked: 1. It lasts longer than if the same element were in a non-topicalized position. 2. It is accompanied by an ***intonation break***. An intonation break is a non-manual signal produced ***together with*** the topic; the head is tilted up and back as the eyebrows are raised (see section 5.3.2 for further discussion of non-manual signals). In writing, topics are separated from the rest of the sentence with a comma. For example,

SWIM, HE. (He is swimming.)

DOCTOR, SHE. (She is a doctor.)

BOOK, ME READ. (I read a book.)

READ YOU, ME. (I am reading to you.)

HAIRCUT, TINA GIVE ERIC. (Tina gave Eric a haircut.)

The last example shows how topicalization offers a way of dealing with di-transitive verbs.

5.3.2 Non-Manual Signals; Questions and Negated Sentences

Definition: Non-manual signals are signals produced simultaneously with the sequence of signs by body parts different from the hands (mainly face and head) and that carry grammatical functions.

In addition to topicalized sentences (see previous section), questions and negated sentences are marked by non-manual signals. Both questions and negated sentences assume the same basic sign order as declarative sentences and they are distinguished by the presence or absence of certain facial expressions. During the delivery of a declarative sentence (unless it is topicalized), the facial expression is neutral and eyebrows and body remain relaxed. However, if the sentence is negated, the signer produces a side-to-side head shake together with the signs. If the sentence is a yes/no question, the signer raises her eyebrows and tilts her head slightly forward or to one side. Finally, if the sentence is a wh-question, the signer squints her eyebrows downward. A wh-question also includes an interrogative sign as *the last* of the sequence.

Non-manual signals are written on a line above the sentence; **q** for question and **n** for negation. The line extends over the part of the sentence that is accompanied by the signal. (ASL linguists argue whether non-manual signals accompany the whole sentence or just part of it. We choose to follow Isenhath and present a variety of examples. For an alternative analysis, see [36].) Consider the following examples,

WOMAN $\overset{n}{\overline{\text{BUY DOG}}}$. (The woman didn't buy the dog.)

$\overset{n}{\overline{\text{WOMAN}}}$ BUY DOG. (It wasn't the woman who bought the dog.)

DOCTOR, $\overset{n}{\overline{\text{YOU MUST GO}}}$. (You don't have to go to the doctor.)

$\overset{q}{\overline{\text{DIANA HERE}}}$? (Is Diana here?)

$\overset{q}{\overline{\text{YOU FINISH}}}$? (Did you finish?)

$\overset{q}{\overline{\text{YOU LATE WHY}}}$? (Why are you late?)

$\overset{q}{\overline{\text{WASH DISH WHO}}}$? (Who is going to wash the dishes?)

ASL also has negating signs (e.g., NOT, CAN'T, NEVER, NONE) but the non-manual signal is the fundamental way of negating a sentence and it is always present even if the signer chooses to include a negating sign. The combination of two negatives (the non-manual signal and a negation sign; two negation signs are not allowed in the same sentence) does not make a sentence positive, as is the case in English; the negating sign is only present to emphasize and/or shade the meaning. A negating sign usually follows its target. Some examples,

$\overset{n}{\overline{\text{HE KNOW NOT}}}$. (He doesn't know.)

$\overset{n}{\overline{\text{HE KNOW CAN'T}}}$. (He can't possibly know.)

$\overset{n}{\overline{\text{HE KNOW NEVER}}}$. (He never knows.)

The sign NONE is used to express the absence of something/somebody. NONE follows its target sign.

$\overset{n}{\overline{\text{CHANGE NONE}}}$, ME HAVE. (I have no change)

$\overset{n}{\overline{\text{TRAFFIC NONE}}}$. (There is no traffic.)

5.3.3 Compound Sentences

In English, a compound sentence consists of two or more independent clauses joined together by a coordinating word (and, or, for, etc.), while in ASL, independent clauses are expressed separately. Sentence arrangement is usually based on the actual chronological order of events. Consider the following English sentence, "I am going to write a letter as soon as I get home." In ASL, this sentence is divided into two sentences and rearranged according to the actual chronological order: RETURN HOME, ME. WRITE LETTER.

Although chronological sentence arrangement is most common, other strategies are available: **relative importance**—a main thought is established first and then comments, e.g., WIN, HE. SHE PROUD. ("He won and she is proud of him."), **cause and effect**—the cause is established first and then the result, e.g., LIGHT OUT. ME SCARED ("When the lights go out, I get scared."), **event-reaction**—the main though is immediately followed by a reaction or response, e.g., DATE, HE ASK ME. ALMOST FAINT, ME. and finally, **separation of emotional states**—different emotional conditions of different people must be conveyed in separate clauses, e.g., ME DOUBT. HAPPY FUTURE, SHE. ("I doubt she will be happy."). If the emotional condition of all participants is the same, it can be conveyed in one sentence, e.g., HE SHE LAUGH. ("They laughed.").

5.3.4 Conditional Sentences

A conditional sentence consists of a condition, which can be "real" or "unreal" (here, "unreal" means "cannot possibly be fulfilled"), and a result. Conditional sentences in ASL are syntactically very similar to conditional sentences in English except that the sentence is divided into separate clauses.

ASL has two ways of expressing real conditions: 1. the sign SUPPOSE, as in, SUPPOSE ME PASS TEST. ME BUY YOU DINNER. ("If I pass the test, I will take you out for dinner"). 2. Stating the condition as a rhetorical question, as in, ME PASS TEST? ME BUY YOU DINNER.

A common way of expressing "unreal" conditions is to make it clear that the situation is hypothetical. The signer can start by establishing the "truth value" of the condition, as in, MONEY NONE. SUPPOSE HAVE MONEY. BUY FANCY CAR. ("I don't have the money, but if I did, I would buy a sports car"), or by informing the addressee that the result will never be realized. For example, $\overset{n}{\overline{\text{BUY CAR.}}}$ HAVE MONEY? BUY FANCY CAR ("I am not going to buy a car, but if I had the money I would buy a sports car").

5.3.5 Subordination

A subordinate sentence contains an independent main clause and one or more dependent clauses, which are linked to the main clause by subordinating conjunctions. ASL uses subordinating conjunctions but the frequency of use is uncertain, and they are fewer than and different from English conjunctions. The most common ones are BECAUSE and BUT, which signal causation and exception respectively. BECAUSE is also sometimes equivalent "since" in English. Consider the following examples,

> $\overline{\text{CAR START}}^{n}$ BECAUSE GAS NONE. (The car won't start because it is out of gas.)
>
> WE-TWO TOGETHER NEVER BECAUSE YOUR NEW JOB. (We never see each other since you got your new job.)
>
> ME WANT BUY COAT BUT MONEY NONE. (I want to buy a coat, but I don't have the money.)

This is the only complex sentence type in ASL which is not divided into individual clauses.

5.3.6 Infinitives and Gerunds

Infinitives that are not subcategorized for, are translated into ASL as a separate clause (In the previous example, "want" subcategorizes for an infinitive.). An example,

> "Her transfer prompted him to find a job."
> SHE CHANGE JOB. HE FIND JOB.

Gerunds are verbs with the suffix "-ing" attached to the stem and they are used as nouns (e.g., run*ing* and sing*ing*). ASL does not have gerunds and sentences that in English, contain a gerund, are translated into ASL by breaking them down into separate clauses. For example,

> "She won by running a good race."
> SHE WIN. RUN GOOD RACE.

5.3.7 Plural, Quantification

Nouns may be pluralized by including a plural modifier before the noun. Some plural modifiers are: FEW, SOME, SEVERAL, MANY and ALL, or numerals. Consider the following examples:

TWO ROOM, WE PAINT FINISH. (We have finished painting two rooms already.)

DROP MANY BOOK, ME. (I dropped a lot of books.)

WASH ALL CAR. (Wash all the cars.)

As mentioned before, a signer may choose to pluralize a noun by reduplicating the main verb of the sentence three or more times. Recall the following example:

STUDENT CONVENE (move hands) CONVENE (move hands) CONVENE VANCOUVER.
(The students gathered in Vancouver.)

Also a small number of nouns may pluralize by reduplication of the noun itself (e.g, tea, book, chair).

Plural objects may be indicated by adding a smooth, horizontal arc to the verb (suffix for ordinary multi-directional verbs, prefix for reverse multi-directional verbs). The smooth arc indicates collective plural and implies that the action affects all referents as a whole, i.e., if the plural suffix is added to TELL it means "tell all of them", and we transcribe this form of plural by adding -ALL to the original verb, e.g., TELL-ALL.

If the signer wants to convey that the action affects each referent individually, she adds the exhaustive affix to the verb. (Some researchers argue that the exhaustive affix is not just an affix, but a complete reduplication of the verb [59, p.122].) The exhaustive affix is an arc with 3-5 "bounces". If the verbs has local movement, it is repeated at individual point on the arc. We transcribe the exhaustive plural by adding -EACH to the original verb, e.g., GIVE-EACH.

Yet another inflection is the allocative, which has the meaning "certain, but not all". This affix consists of repetitions of movement at randomly varying points in space, rather than in an arc. We transcribe the allocative by adding -SOME to the original verb, e.g., GIVE-SOME.

5.3.8 Dual Inflection

The signer can modify multi-directional verbs to indicate two objects (some signers also use trial inflection). Dual inflection can be accomplished in two ways. 1) First moving to one reference point and then quickly "bouncing" to the other reference point. (If the verb includes local movement (e.g., INFORM, the local movement is not repeated when moving from the first reference point to the second. This helps in distinguishing between the dual and the exhaustive affixes.) 2) The signer repeats the verb twice, each time with a different endpoint (starting point for reverse agreement verbs). Consider the following examples:

$_{eli}$GIVE$_{jörg,pepe}$ CAKE or

$_{eli}$GIVE$_{jörg}$ $_{eli}$GIVE$_{pepe}$ CAKE

5.4 Verbs in ASL

Verbs are divided into two main groups: 1) *multi-directional*[3] verbs can incorporate subject and object into their movement, or the positioning of the verb conveys semantic information. 2) *non-directional* verbs must have overt subject and/or object.

5.4.1 Non-Directional Verbs

Non-directional verbs (e.g., EAT, LIKE, LOVE, WANT, WRITE) have all four parameters (hand shape, movement, position, palm direction) completely specified and they are always performed the same way, regardless of discourse. The movement usually includes direct contact with the body or movement toward the body and any deviation from the specification will make them unintelligible.

Spatial Indexing

Pronominal references (i.e., subjects and objects) in sentences with non-directional verbs are achieved by *indexing*. If a referent is actually present, the pronominal reference is made by making an indexing motion directly towards the referent. If the referent is not present, the signer must establish a contextual reference; the signer introduces the noun and establishes a reference point in space for referring to that person/object. All later reference to that person/object is made by an indexing motion in the direction of this point. Consider the following discourse,

DIANA (establ) CALL. INDEX(DIANA) PLAN LATE.
"Diana called. She will be late."

In the first sentence, the reference point is established, while in the second, it is being indexed. There are three ways of establishing a reference point: 1. Make the noun sign at the location. 2. Use a POINT sign to the particular location. 3. For some one-handed signs,

[3] Also called agreement verbs.

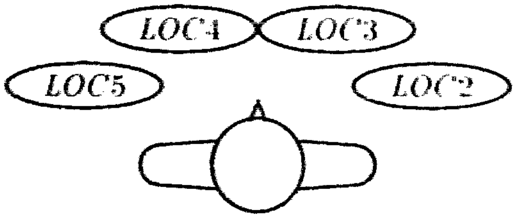


Figure 5.1: A possible arrangement of spatial references.

a POINT sign can be made by the non-active hand, at the same time as the active hand is signing the noun. A signer may have 4–5 active reference points at a time. Figure 5.1 shows a possible arrangement of spatial reference points. (In this example *LOC1* is the signer herself.)

5.4.2 Multi-Directional Verbs

Multi-directional verbs (e.g., ASK, GIVE, MEET, HELP, SEE, BRING) do not include contact with the body and only the hand shape and location parameters are completely specified. The verb's movement and/or palm direction parameters are not determined until the the verb is put in context. (These verbs exist in a dictionary form for which the movement/palm direction parameters are specified.)

Multi-directional verbs use spatial loci and linear movement to differentiate between the subject and the object (direct or indirect) of a sentence. As with non-directional verbs, the signer must establish contextual reference points in space. But instead of indexing the point, pronominal reference is achieved by incorporating the reference points into the verb's movement; the starting point represents the subject and the end point represents the object. (Some verbs have the directions reversed, i.e., the starting point represents the object and the end point represents the subject, e.g., TAKE, INVITE, APPOINT).

Consider the following situation where spatial references for Diana and David are established to the left and the right of the signer, respectively. In this scenario, the sentence "Diana asked David." would be signed by starting ASK at the left and moving it over to the right ($_{left}$ASK$_{right}$). Conversely, the sentence "David asked Diana." would move ASK from the right side to the left side ($_{right}$ASK$_{left}$). Notation: The start point is written as a subscript immediately in front of the verb, while the end point is written as a subscript immediately behind. In written texts, one does not use *left* and *right* but numbers: 1=signer (first person), 2=addressee (second person), 3=third person. I.e., both the above examples are normally transcribed: $_3$ASK$_3$. To avoid any confusion, we will sometimes use names instead of the number 3 for third person references: ${}_{diana}$ASK$_{david}$.

Palm Direction

Some multi-directional verbs use palm direction to distinguish between the subject and the object of a sentence; the back of the hand faces the subject, while the palm faces the object (direct/indirect). Consider the sentence "I tease you" ($_1$TEASE$_2$), when giving the sign for TEASE, the palm is facing the addressee and the back of the hand is facing the signer. Some other verbs in this class are: HATE, PITY.

Body Oriented Verbs

Body oriented verbs are not true multi-directional verbs since their movement parameter is strictly defined and therefore they cannot incorporate subject and/or object into their movement. Instead of a variable movement parameter, their position parameter is loosely defined and they can use location to focus attention on an anatomical part of the body. For example, if HURT is performed in front of the head, the signer conveys the idea of a hurting head, if the verb is performed in front of the stomach, the signer conveys the idea of a hurting stomach, etc. Some other body oriented verbs are HURT, ACHE, CUT, WASH.

5.4.3 Surface Structure in Discourse

The surface structure of a sentence in a discourse can be quite different from that of an isolated sentence. We have already seen how multi-directional verbs can delete the subject and/or object of a sentence by incorporating them into the verb movement. Below we will further discuss the ASL sentence structure from a discourse point of view.

If the subject of a sentence is the same as the subject of the previous sentence, the signer can delete it without any change in meaning. If the first verb in a discourse appears without a subject, first person is assumed. This kind of subject deletion is optional and the signer may choose to use both overt and deleted subjects in the same discourse. One can use this technique with non-directional verbs.

In a situation where first person reference is not relevant, the signer can use her body to refer to a third person. Consequently, first person reference is no longer possible. An index motion towards the body is interpreted as a third person reference, and similarly, for multi-directional verbs; a start/end point near the signers body is interpreted as third person. This is similar to direct quotation in English: *Terminator said, "I'll be back."*

A variation of the above can be used with multi-directional verbs: the signer "takes on" the "roles" of two other referents (first person reference is no longer possible). At the beginning of the discourse, the signer gives the nominal signs for the referents and establishes a body orientation (left/right) for each. When giving the sign for the verb, the signer conveys subject reference by turning her head in the direction established for the referent, say left. Further, if the verb uses *palm direction* to distinguish between its arguments, the verb sign is given with the back of the hand facing left (the same way as the head) and the palm facing right. If the verb uses *movement* to distinguish between its arguments, the sign moves to the right (towards the object of the sentence—the direction opposite to the orientation of the head). This way, both subject and object can be deleted from the surface structure of a sentence and both subject and object reference are conveyed through a simple turn of the head. These concepts are illustrated by the following example (modified from [22]):

> MARY (establish orientation, left) TOM (establish orientation, right) MEET
> (turn head right) FLIRT (moving left)
> (turn head left) HATE (back of hand facing left, palm facing right)
> (turn head right) BOTHER (moving right).
>
> "Mary and Tom met. He flirts with her. She hates him. He bothers her."

5.5 Time

5.5.1 Time Markers

Time markers are signs that either denote a specific time (e.g., YESTERDAY, TOMORROW) or signs which can place an action in the past or future time division (PAST and FUTURE respectively).

Specific Time Markers

A specific time marker must satisfy the following two conditions [27, p.192]:

1. It must define the time span involved.

2. It must place the context into an explicit time division (past, future).

For example, the signs TOMORROW, YESTERDAY and EVERYDAY satisfy both conditions, while NIGHT, WEEK and YEAR only satisfy the first and can therefore not be classified as specific time markers (they are nouns). Most specific time markers are compounds. ASL forms time compounds by fusing a base time sign (e.g., seasons, days of week, months) with a modifier sign (e.g., NOW, PAST, NEXT) such that the resulting compound meets the above conditions. For example, DURING-SUMMER, NEXT-MONDAY, NOW-NIGHT ("tonight"), PAST-WEEK ("last week") are time compounds.

General Time Markers

General time markers are distinguished from specific time markers in that they only supply the time division, i.e., they only satisfy the second condition above. ASL has two general time markers—PAST and FUTURE—which place a sentence/discourse in the past and future time divisions respectively. Present time is unmarked. A general time marker usually follows the target verb.

Indexing Time

Specific time markers can be produced by attaching an index movement to a basic time sign (e.g., WEEK, YEAR). As the time sign nears completion, the active hand changes into an index hand and moves backwards or forwards depending on whether the time expression conveys past or future. For example, LAST-WEEK has an index movement backwards over the signer's shoulder, while the index movement in NEXT-YEAR is extended in the forward direction. A signer can substitute a single-digit-number hand shape for the index hand shape in the indexing motion to convey concepts like TWO-WEEKS-AGO and IN-TWO-WEEKS.

5.5.2 Verb Tenses, Time in the ASL Discourse

ASL does not use verb tense or verb inflection to indicate time. A base time frame is established at the beginning of a discourse and—until the signer deliberately changes the time frame—all further references to time are made with respect to this time frame. The signer usually establishes the precise time of a discourse (the base time frame) by placing a *specific time marker* at the beginning of a sentence/discourse. Later, she can change the time frame by supplying either a *general* or *specific time marker*. Consider the following connected discourse:

> (1) PAST-NIGHT PARTY, ME ATTEND. (2) MEET SPECIAL PERSON. (3) DANCE. (4) TALK. (5) WONDERFUL TIME, WE ENJOY TOGETHER. (6) NEW FRIEND, MAYBE SEE FUTURE AGAIN SOON. (7) WE-TWO MAYBE GO DANCE AGAIN.

> "I went to a party last night. I met a special person there. We danced and talked and had a great time. Maybe I will see my new friend again some time soon. Maybe we will go dancing again."

In this example, the specific time marker PAST-NIGHT establishes 'last night' as the current time frame and the first four sentences are interpreted as describing events that happened 'last night'. In the fifth sentence, the general time marker FUTURE changes the time frame and consequently the last two sentences of the discourse are interpreted as future events.

Now, consider the following *isolated sentences*:

1. PAST-NIGHT PARTY, ME ATTEND.
2. SPECIAL PERSON, MEET PAST.
3. DANCE PAST, WE-TWO.
4. TALK PAST, WE-TWO.
5. WONDERFUL TIME, WE-TWO ENJOY PAST.
6. NEW FRIEND, MAYBE SEE FUTURE AGAIN SOON.
7. MAYBE GO DANCE FUTURE AGAIN.

Since these are isolated sentences, a time marker is required for each sentence.

The base time frame is physically represented by the signer's body and the space immediately in front of the signer (neutral space). The space behind the signer's body represents time prior to the time frame, while the space in front of neutral space represents future time. This is often referred to as the ASL *time line* - an imaginary line running horizontally alongside the body.

The conversational time reference (i.e, the time at which a conversation takes place) is always present. If no time frame is established at the beginning of a conversation, it is interpreted in the present tense. For example, if you walk up to a friend and sign YOU DO WHAT?, = "what are you doing?", she might answer: READ, ME = "I am reading". No time markers are used since present time is unmarked.

Normally the time frame does not change until the discourse has concluded, but ASL includes a mechanism to temporarily switch time frame. If the signer tilts the body slightly forward/backward, neutral space no longer represents the initial time frame but future/past. This remains in effect until the body is returned to upright position.

Chapter 6

ASL—The Computational Challenge

Traditionally, computational analysis of a sentence is based on parsing words one at a time and if the sequence fits an allowable sentence pattern it is accepted as a valid sentence in a given language. A wide range of sentences in spoken languages can be analysed this way, since spoken languages are sequential of nature and one can derive a meaning representation of a sentence from the combination of allowable sentence patterns (as defined by a grammar) and individual lexical features (number, person, case, etc.). However, sequential processing of lexical items (signs) in an ASL sentence is not sufficient since ASL uses non-manual signals and spatial information in parallel with lexical items—instead of inflection and sign order—to convey several grammatical functions.

We propose an algorithm which expands the scope of traditional natural language parsers/grammars to manual languages by adding a non-lexical component to the parser which records spatial and other non-manual behaviour and augmenting the parser with complementary routines for accessing the new information. Both the non-lexical parsing routine and the grammar were developed in a logic programming framework, but our ideas are versatile and can be used to augment other types of parsers. Our application is a natural language front end to a deductive database but the algorithm is, of course, not restricted to this application.

It should be noted that our choice of programming language responds to the type of information we are processing, and our specific application makes logic programming ideally suited. For instance, the logical expressions for the ASL linguistic information presented in section 6.3 translates very cleanly into Prolog. The advantages of using logic programming

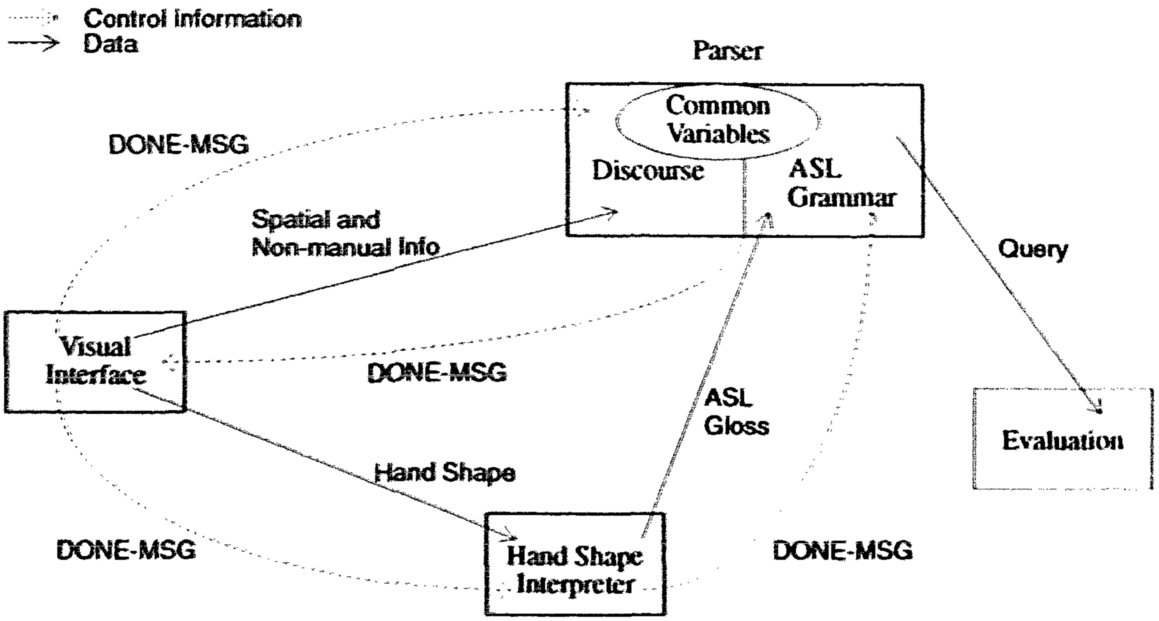


Figure 6.1: An ASL front end.

for such applications have been covered in, for example [56, 2, 38]. The DCG form of logic programming and its applications to language processing is discussed in [46, 14].

The chapter is organized as follows: Section 1 presents an overview of the ASL front end, Section 2 introduces the reader to the new ideas in the parser and how they are incorporated in its two main components: the ASL-grammar and the non-lexical parsing routine. A detailed discussion of the non-lexical parsing routine and the ASL-grammar is presented in sections 3 and 4. In Section 5, we discuss how to translate ASL into our logic representation LM.

6.1 An ASL Front End

We present an overview of the system, describe how the different modules interact and give a brief functional description of each component.

6.1.1 Overview

The front end, as illustrated in Figure 6.1, consists of three main modules: a visual interface, a sign interpreter and a parser, where the parser is divided into two sub-modules; one for lexical items (ASL-grammar, see Section 6.4) and one for non-lexical information (non-lexical parsing routine, see Section 6.3.2). In this thesis, we are concerned with the two parsing modules, but before going into details about the parser, we will give a brief description of the whole front end that we assume it being part of.

A typical sequence of events proceeds as follows. The visual interface gets its input from a camera filming the signer, and from this input, it produces two types of output: lexical and non-lexical. (The visual interface need not be a camera. It could be, for example, a data glove/data suit (see [20]), but since there is work being done at SFU on handshape recognition from images, we decided to think of it as a camera.) Non-lexical output is the spatial and non-manual information that accompanies a sign sequence, while the lexical information consists of descriptions of the individual signs in a sentence. When the visual interface has finished processing the camera input, the sign interpreter starts to translate the sign descriptors into a sequence of English glosses and the parser records the non-lexical information. When they are finished, the ASL-grammar proceeds to produce a semantic representation of the ASL-sentence which is sent to an evaluation procedure (for example, the one described in Chapters 3 and 4) and the parser informs the visual interface that it can deliver the next sentence's data sets. This process repeats until the signer has no more queries to ask and signs off.

With one exception, we assume that input is processed sentence by sentence and that together, the lexical and non-lexical information describe one complete sentence. The exception is establishment of spatial reference points, which we assume is processed as if it were a complete sentence, i.e., the signer must establish reference points one by one and prior to using them. This simplifying assumption avoids the following problem: Consider a sentence with several noun phrases, e.g., "Diana wrote Sergio a letter". If the signer intends to continue speaking about Sergio, she may choose to establish a reference point for him before giving the whole sentence and then index this point, i.e., SERGIO (establish ref.). DIANA WRITE INDEX(SERGIO) LETTER, or she may choose to create the reference point as she gives the sentence, i.e., DIANA WRITE SERGIO (establish ref.) LETTER. If we allow the second type of sentence, the visual interface must tag each sign that is accompanied by an indexing motion, otherwise, the parser cannot tell which noun sign is being indexed since the visual interface only records *the order* in which indexing motions are made. By disallowing the second type (and thereby avoiding the tagging), we achieve a clear distinction between lexical and non-lexical information.

6.1.2 The Visual Interface and the Sign Interpreter

We are not concerned with the low level routines in this thesis, but will include a short description of them because of the importance of them being properly synchronized with the parser (also summarized in figure 6.2). The low level processing in the front end is described in terms of a visual interface and a sign interpreter, but in any real system it will likely be done by many subcomponents. But, assuming a two part low level system, the low level processing proceeds as follows. For initialization purposes (see 6.2.3) the signer must introduce herself to the system, so before entering its main loop, the visual interface provides the sign interpreter with lexical input which identifies the signer (e.g., her name). After having informed the sign interpreter that the input is ready, it enters the main loop and immediately starts processing the next input from the camera. The output is not delivered to the sign interpreter and the non-lexical parsing routine until the visual interface receives a message from the parser saying that the previous sentence is completely processed and it is ready to receive the next sentence's input data. When this message arrives, it sends its output and a DONE-MSG to the sign interpreter and parser. This process repeats until the signer has no more queries to ask and signs off, at which point the visual interface sends a FINISH-MSG to the parser and sign interpreter and the process is shut down.

The visual interface receives its input from a camera which films the signer and transforms the camera input into input to the sign interpreter (a sequence of sign descriptors) and to the non-lexical parsing routine. The sign descriptors are translated into a sequence of English glosses by the sign interpreter, while the non-lexical information is recorded by the non-lexical parsing routine described in Section 6.3.2. According to W. C. Stokoe [57], ASL signs can be described in terms of the following four parameters: hand shape, relative position to body, path movement, and orientation of the hands, so we assume this to be the minimum amount of information present in a sign descriptor.[1] Some spatial information may be common between a sign descriptor and the non-lexical input, but normally (e.g., indexing, role play), the visual interface has to extract spatial references for the non-lexical routine separately. In addition, it has to extract non-manual signals and shifts in body position from the camera input.

The information from the camera must be translated into a format which is recognized by the corresponding routine. Since the exact shape of the low level system is uncertain and we are not concerned about the sign interpreter in this thesis we don't want to speculate on

[1] Interesting work in the area of hand shape recognition and movement is currently being done at SFU, see [17].

```
Process: Visual-Interface( )
    % signer introduces herself
    Get Input from camera
    Produce Lexical-output and Non-lexical-output from Input
    Make Lexical-Output available to Sign Interpreter
    Send DONE-MSG to Sign Interpreter
    % Main loop
    While not done do
        Get Input from camera
        Produce Lexical-output and Non-lexical-output from Input
        Wait for DONE-MSG from Parser
        Make Lexical-output available to Sign Interpreter
        Make Non-lexical-output available to Parser
        Send DONE-MSG to Parser and Sign Interpreter
    End While
    Send FINISH-MSG to Parser and Interpreter
    Shut down process
End Visual-Interface
```

Figure 6.2: A high level description of the visual interface.

its input format. However, the non-lexical output is consumed by the non-lexical parsing routine which we describe later in this chapter so we make the following assumptions about the non-lexical output: All forms of spatial referencing (indexing, palm direction, path movement, i.e., begin and end point of multi-directional verbs, and role play) must yield as output the name of the spatial location being referenced: Hence, spatial references must translate into named locations as follows.

Indexing: LOC1, LOC2, LOC3, LOC4, or LOC5

Role play: LEFT, or RIGHT

Path movement: LOC1, LOC2, LOC3, LOC4, LOC5, LEFT, or RIGHT

Palm direction: LOC1, LOC2, LOC3, LOC4, LOC5, LEFT, or RIGHT

Further, non-manual information must translate into the following names: TOP, QUE, NEG, FORWARD, BACKWARD. (More details of the non-lexical input are presented in Section 6.3.1 together with some examples.)

6.2 The Parser

We give a high level description of the parsing algorithm and the interface between the lexical (ASL-grammar) and the non-lexical parsing routines. (The details of these two routines are discussed in Sections 6.3.2 and 6.4.)

6.2.1 The Parsing Algorithm

A pseudo code version of the parsing algorithm is presented in Figure 6.3. Every time the system is started up, the signer must introduce herself to the system (e.g., by signing her name) and once the signer's identity is known, the parser executes its startup routine to initialize the variables in the interface between the lexical and non-lexical parsing routines (see Sections 6.2.2 and 6.2.3). After having executed the initialization routine, the parser informs the visual interface of this and enters a loop where further processing is blocked until it receives a message from the visual interface; If the signer had no more queries to ask and signed off, the parser receives a "FINISH"-message and the parsing process is shut down. Otherwise, the visual interface provides the parser with input to the non-lexical parsing routine. After the non-lexical information has been recorded, the parser is blocked until the sign interpreter has finished translating sign descriptors into a sequence of English glosses. It is important to ensure that the non-lexical parsing routine is completely finished before the ASL-grammar starts, since the ASL-grammar must use the non-lexical information to build a semantic representation of the ASL query. If the grammar is allowed to start before the non-lexical parsing routine is finished it may build the representation from obsolete information. When done, the sign interpreter sends a "DONE"-message to the parser such that it knows when to pick up its lexical input. The lexical input (English glosses) is analysed by the ASL-grammar and together with the previously recorded non-lexical information, the grammar builds a semantic representation of the input ASL query. When the grammar is done, the parser makes the semantic representation available to the evaluation procedure and sends a "DONE"-message to the visual interface so it can start processing a new query. Now, the parser is again blocked until it receives a message from the visual interface, at which point the above algorithm is repeated.

```
Process: Parser( )
    Wait for DONE-MSG from visual interface
    Wait for DONE-MSG from sign interpreter
    Get Input from sign interpreter
    Call Initialize(Input)
    Send DONE-MSG to visual interface
    While not done do
        Wait for Msg from visual interface
        If Msg = FINISH-MSG then
            Shut down process
        Else
            Get Non-Lexical-Input from visual interface
            Call Non-Lexical-Parsing-Routine(Non-Lexical-Input)
            Wait for DONE-MSG from sign interpreter
            Get Lexical-Input from sign interpreter
            Call ASL-Grammar(Lexical-Input, Grammar-Output)
            Make Grammar-Output available to evaluation procedure
        End If
        Send DONE-MSG to visual interface
    End While
End Parser
```

Figure 6.3: The Parsing Algorithm

6.2.2 The Interface between the Lexical and Non-Lexical Parsing Routines

The lexical and the non-lexical parsing routines communicate through a set of common variables:

> *Topic, Question, Negation, Loc1, Loc2, Loc3, Loc4, Loc5, Left, Right, First-Ref, Second-Ref, Third-Ref, Fourth-Ref, First-Location, Time-Frame, Old-Time-Frame, Last-Subject*[2]

Topic, Question and *Negation* are boolean variables which keep track of non-manual information in a sentence. Since at least two of the three signals can be present in the same sentence (e.g., a topicalized question), we chose to use a separate variable for each signal. *Topic, Question* and *Negation* indicate the presence or absence of the non-manual signals that accompany topicalized sentences, questions and negated sentences respectively. Each variable's domain is {PRESENT, ABSENT}, i.e., the presence of a signal is indicated by assigning the value PRESENT to the corresponding variable, while the absence of a signal is indicated by assigning the value ABSENT to the corresponding variable. Their initial value is ABSENT.

Loc1, Loc2, Loc3, Loc4 and *Loc5* record the referents assigned to the spatial reference points established by the signer (see Figure 5.1 for a possible arrangement of spatial reference points). If, for example, the signer in Figure 5.1 establishes a spatial reference point for, say Pepe, at location LOC2 and, say David, at location LOC4, the variables Loc2 and Loc4 would have the values Pepe and David respectively. One of the spatial reference variables is reserved for the signer, say *Loc1*, and must be initialized to the signer at the beginning of a conversation. (We assume that the signer introduces herself to the system as the first thing she does, e.g., by signing her name, such that *Loc1* can be initialized correctly.) *Loc2-Loc5* are initialized to ABSENT since there are no referents assigned to locations LOC2-LOC5 at the beginning of a conversation. The domain of *Loc1* is {animate noun phrases in the system's dictionary} and the domain of *Loc2-Loc5* is {ABSENT, noun phrases from the system's dictionary}.

A signer can have four to five active reference points at a time, excluding the signer and the addressee. We allow the signer four points plus one for herself, but this number can easily be changed to five if necessary. (A point for the addressee was not included since that would be the camera in our situation.) We assume that each variable (*Loc1, Loc2, Loc3,*

[2] A note on notation: *Variable* names start with an upper case letter followed by lowercase letters and/or digits and/or dashes. *Value* names or constants are in all UPPER CASE.

Loc4, Loc5) corresponds to one *fixed* physical location (or rather area) in the signing space and that the visual interface is capable of distinguishing between these locations[3]. Having fixed locations may seem like putting an unnatural restriction on the signer, but since there are only four of them, each area can be made quite large and that should provide the signer with enough flexibility.

Left and *Right* store the referents of role play, e.g., if the signer establishes a reference for Pepe to her left and a reference for David to her right, the variables *Left* and *Right* take the values Pepe and David respectively. Their domain is {ABSENT, animate noun phrases from the lexicon (except the signer)} and their initial value is ABSENT.

First-Ref, Second-Ref, Third-Ref and *Fourth-Ref* record the interpretation of spatial references and keep track of the order in which they are made. Here, the term 'spatial reference' includes indexing, role play and incorporation of spatial locations into verb movement. E.g., if LOC2 and LOC4 are assigned referents as in the above examples, and the signer indexes LOC2 first and LOC4 second or if a multi-directional verb starts at LOC2 and ends at LOC4 or if the first reference is made by turning the head left and the second by turning the head right, then *First-Ref* would have the value Pepe and *Second-Ref* would have the value David. Their domain is {ABSENT, noun phrases from the lexicon} and their initial value is ABSENT.

First-Location records the first spatial reference the signer makes (indexing or role play) and it is used by the grammar when the signer establishes new referents. Its domain is {ABSENT, LOC1-LOC5, LEFT, RIGHT}, and the initial value is ABSENT.

Time-Frame records the current time frame, e.g., if the signer is telling a story that happened yesterday, the value of *Time-Frame* is 'yesterday'. What 'yesterday' really means is up to the user to decide (could be the date, for example). Initially *Time-Frame* is assigned a value which reflects the time at which the "conversation" takes place. The domain is implementation dependent.

Old-Time-Frame records the time frame which is effective before the the signer temporarily switches to a new time frame by shifting her body backward or forward. The initial value is ABSENT and the domain is implementation dependent.

Last-Subject records the last articulated subject reference. (It is common in ASL to omit the subject of a sentence if it is the same as in the previous sentence.) The domain is {all noun phrases from the dictionary} and the initial value is the same as *Loc1*, i.e., the signer.

[3] According to B. Dorner, this is a reasonable assumption. Personal communication, Sch. Comp. Sci., SFU, 1993.

Comments

Having only four variables to record spatial references puts some restrictions on the type of sentence that a system can accept. For example,

- The subject of intransitive verbs may only include up to four referents but if the signer indexes locations in addition to the subject, the number of subject referents is reduced.
- Multi-directional transitive verbs may have up to two subject and object referents or the signer can give one location by indexing which limits the number of subject and object referents to one.
- Multi-directional di-transitive verbs may have two subject and object referents provided the direct object is signed and not indexed. If the signer wishes to index the direct object, subject and indirect object referents are limited to one.

In our prototype, four variables suffice but if the user finds it necessary, she can easily eliminate these restrictions by adding more variables (and modify the input accordingly).

Strictly speaking the variable *First-Location* is redundant since it is not necessary to interpret spatial references and role play in the non-lexical routine. One can simply record the reference locations and leave the interpretation until they are actually used, i.e., when the grammar needs referents to build a meaning representation. If one choses this approach, one has to interpret the references every time they are used. However, in a large system, we expect the sign interpreter to take longer than the non-lexical parsing routine and if this is the case, it makes sense to interpret references in the non-lexical routine since these routines can run in parallel. The grammar cannot start until the sign interpreter is finished, so postponing interpretation to the grammar would degrade the system's performance.

6.2.3 Initialization

Every time the system is started up, the parser executes the initialization routine in Figure 6.4. This routine summarizes the initialization information given in Section 6.2.2.

6.3 The Non-Lexical Parsing Routine

We describe the input to the non-lexical-parsing routine and illustrate it with some examples. Further, we describe how the input is interpreted by the non-lexical parsing routine.

```
Subroutine : Initialize(IN: Signer)
    Let First-Ref = ABSENT
    Let Second-Ref = ABSENT
    Let Third-Ref = ABSENT
    Let Fourth-Ref = ABSENT
    Let Last-Subject = Signer
    Let Left = ABSENT
    Let Right = ABSENT
    Let First-Location = ABSENT
    Let Loc1 = Signer
    Let Loc2 = ABSENT
    Let Loc3 = ABSENT
    Let Loc4 = ABSENT
    Let Loc5 = ABSENT
    Let Time-Frame = "now"
    Let Old-Time-Frame = ABSENT
End Initialize
```

Figure 6.4: The initialization routine

6.3.1 Input to the Non-Lexical Parsing Routine

The non-lexical parsing routine assumes that its input is ordered and that there are no "holes" in the input data. The order is arbitrary but it has to agree with the order in which the non-lexical parsing routine reads it. To avoid "holes" in the data, we introduced the special value NULL to mean "not present" (We use different names for values in the non-lexical input and values in the interface even if the semantics are the same. (For example, the input values for topicalized sentences are TOP and NULL (explained below), while the corresponding interface variable *Topic* may take on the values ABSENT or PRESENT (see 6.2.2).) We made this distinction strictly to enhance readability and the user may of course choose names as she pleases.), e.g., if a sentence does not have a third spatial reference, the visual interface assigns NULL to the sixth entry (see Table 6.1). At the conceptual level, the input can be described as an array of non-empty strings, but it is up to the user to find the best actual data structure for a particular implementation. An overview of the input format is presented in Table 6.1 and the individual entries are described below. We introduce the following notation: *Non-lexical-Input*—the name of the array, *Non-lexical-Input[index]*—an individual entry *or* value (should be clear from context, e.g., *Input[1]* may mean the first entry or the *value* of the first entry depending on the context), *Non-Lexical-Input[range]*—consecutive entries (e.g., *Non-Lexical-Input[1-3]* means the first three entries

Entry #	Conceptual Name	Values
1	Topic	TOP, NULL
2	Question	QUE, NULL
3	Negated	NEG, NULL
4	1^{st} Spatial Reference	LOC1, LOC2, LOC3, LOC4, LOC5, NULL, LEFT, RIGHT
5	2^{nd} Spatial Reference	LOC1, LOC2, LOC3, LOC4, LOC5, NULL, LEFT, RIGHT
6	3^{rd} Spatial Reference	LOC1, LOC2, LOC3, LOC4, LOC5, NULL, LEFT, RIGHT
7	4^{th} Spatial Reference	LOC1, LOC2, LOC3, LOC4, LOC5, NULL, LEFT, RIGHT
8	Body Position	FORWARD, BACKWARD, NULL

Table 6.1: Input to the non-lexical parsing routine.

or the *values* of the first three entries).

Non-lexical-Input[1-3] indicate the presence or absence of the non-manual signals that accompany questions, topicalized and negated sentences. The absence of a signal is indicated by the value NULL, while the values TOP, QUE and NEG indicate the presence of a signal.

Non-lexical-Input[4-7] indicate the presence or absence of a first, second, third and fourth spatial reference (as before, the term 'spatial reference' includes indexing, role play and incorporation of spatial locations into verb movement). The absence of a spatial reference is indicated by the value NULL, while the presence of a spatial reference is indicated by the values: LOC1, LOC2, LOC3, LOC4, LOC5, LEFT and RIGHT, where LOC1, LOC2, LOC3, LOC4 and LOC5 denote the physical spatial locations in the signing space, and LEFT and RIGHT denote the left and right side of the body (see Figure 5.1).

Non-lexical-Input[8] indicate body position. The value NULL indicates a neutral body position, while the values FORWARD and BACKWARD indicate forward and backward shifts.

Some Examples

Table 6.2 illustrates the non-lexical input expected from the visual interface for some sample ASL sentences.

6.3.2 The Routine

The interpretation of the non-lexical input through the non-lexical parsing routine is straight forward (the complete non-lexical parsing routine and its subroutines are illustrated in Figure 6.5):

```
Subroutine : Non-Lexical-Parsing-Routine(IN: Non-Lexical-Input)
    Call Non-Manual-Signal(Topic, Non-Lexical-Input[1])
    Call Non-Manual-Signal(Question, Non-Lexical-Input[2])
    Call Non-manual-Signal(Negation, Non-Lexical-Input[3])
    Call Spatial-Ref(First-Ref, Non-Lexical-Input[4])
    Call Spatial-Ref(Second-Ref, Non-Lexical-Input[5])
    Call Spatial-Ref(Third-Ref, Non-Lexical-Input[6])
    Call Spatial-Ref(Fourth-Ref, Non-Lexical-Input[7])
    Call Temp-Time-Frame(Non-Lexical-Input[8])
End Non-Lexical-Parsing-Routine

Subroutine : Non-Manual-Signal(Non-Manual-Signal, Value)
    If Value = NULL then
        Let Non-Manual-Signal = ABSENT
    Else
        Let Non-Manual-Signal = PRESENT
End Non-Manual-Signal

Subroutine : Spatial-Ref(IN: Reference, Location)
    First-Location = Location
    Case Location:
        LEFT: Let Reference = Left
        RIGHT: Let Reference = Right
        LOC1: Let Reference = Loc1
        LOC2: Let Reference = Loc2
        LOC3: Let Reference = Loc3
        LOC4: Let Reference = Loc4
        LOC5: Let Reference = Loc5
    End Case
End Spatial-Ref

Subroutine : Temp-Time-Frame(IN: Non-Manual)
    If Non-Manual = FORWARD then
        Let Old-Time-Frame = Time-Frame
        Let Time-Frame = "after"-Old-Time-Frame
    Else If Non-Manual = BACKWARD then
        Let Old-Time-Frame = Time-Frame
        Let Time-Frame = "before"-Old-Time-Frame
    Else If Non-Manual = NEUTRAL and Old-Time-Frame != NULL then
        Let Time-Frame = Old-Time-Frame
        Let Old-Time-Frame = NULL
    End If
End Temp-Time-Frame
```

Figure 6.5: The non-lexical parsing routine and subroutines

ASL sentence	Input to Non-Manual Routine
CAROL SWIM	$[NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL]$
PEPE (establish ref.)	$[NULL, NULL, NULL, LOC2, NULL, NULL, NULL, NULL]$
DAVID (establish ref.)	$[NULL, NULL, NULL, LOC4, NULL, NULL, NULL, NULL]$
${}_{LOC2}\text{TEASE}_{LOC4}$	$[NULL, NULL, NULL, LOC2, LOC4, NULL, NULL, NULL]$
HAIRCUT, ${}_{LOC2}\text{GIVE}_{LOC4}$	$[TOP, NULL, NULL, LOC2, LOC4, NULL, NULL, NULL]$
$\overline{\text{INDEX(LOC2) LIKE INDEX(LOC4)}}^{q}$	$[NULL, QUE, NULL, LOC2, LOC4, NULL, NULL, NULL]$
${}_{LOC1}\text{TEASE}_{LOC2}$ ${}_{LOC1}\text{TEASE}_{LOC4}$	$[NULL, NULL, NULL, LOC1, LOC2, LOC1, LOC4, NULL]$

Table 6.2: Some sample sentences and their corresponding input data

Non-Manual Signals

The non-manual information in *Non-Lexical-Input* (*Non-Lexical-Input[1-3]*) is interpreted as follows:

$$\begin{aligned}
&\forall X \text{ such that } 1 \leq X \leq 3 \wedge \\
&\qquad (\text{if } \textit{Non-Lexical-Input}[X] = \text{TOP, then } \textit{Non-manual}_X = \text{PRESENT} \vee \\
&\qquad \text{if } \textit{Non-Lexical-Input}[X] = \text{QUE, then } \textit{Non-manual}_X = \text{PRESENT} \vee \\
&\qquad \text{if } \textit{Non-Lexical-Input}[X] = \text{NEG, then } \textit{Non-manual}_X = \text{PRESENT} \vee \\
&\qquad \text{if } \textit{Non-Lexical-Input}[X] = \text{NULL, then } \textit{Non-manual}_X = \text{ABSENT}). \\
&\textit{Topic} = \textit{Non-manual}_1,\ \textit{Question} = \textit{Non-manual}_2,\ \textit{Negation} = \textit{Non-manual}_3.
\end{aligned}$$

Subroutine *Non-Manual-Signal()* in Figure 6.5 illustrates this procedure.

Indexing and Role Play

Recall: A spatial reference *point* is a physical location in the signing space that the signer uses to represent somebody or something, while spatial *referencing* is the act of making an index motion in the direction of a previously established reference point.

Spatial reference points and spatial references provide manual languages with a form of "memory" that spoken languages don't possess. The closest equivalent to a spatial reference in spoken languages is their use of pronouns but, while spatial references uniquely define their referents, this is often not the case with pronominal references. From a computational point of view, this aspect of manual languages is an asset; the discourse situation is greatly simplified since spatial referencing make manual languages far less ambiguous. When the signer establishes a new reference, i.e., the signer takes on the role of some other referent or assigns a new referent to one of the reference points, the lexical items (a noun phrase) are sent via the sign interpreter to the grammar, while the spatial information is processed by the parser.

The non-lexical parsing routine is only concerned with *referencing* of spatial locations and body directions, while the ASL-grammar is concerned with *establishing* the references. The routine for establishing references is discussed in Section 6.4.3 and for the rest of this discussion, we assume that the references have already been established. The spatial information in *Non-Lexical-Input* (*Non-Lexical-Input[4–7]*) is interpreted as follows:

$\forall X$ such that $4 \leq X \leq 7 \wedge$

(if *Non-Lexical-Input[X]* = LOCY, where $1 \leq Y \leq 5$, then $Ref_X = LocY \vee$
if *Non-Lexical-Input[X]* = LEFT, then $Ref_X = Left \vee$
if *Non-Lexical-Input[X]* = RIGHT, then $Ref_X = Right \vee$
if *Non-Lexical-Input[X]* = NULL, then $Ref_X = NULL$).

First-Ref = $Ref_4 \wedge$ *Second-Ref* = $Ref_5 \wedge$ *Third-Ref* = $Ref_6 \wedge$ *Fourth-Ref* = $Ref_7 \wedge$
First-Location = *Non-Lexical-Input[4]*.

Subroutine *Spatial-Ref()* in Figure 6.5 illustrates this procedure.

Time

The signer can temporarily change the time frame by leaning her body slightly forward or backward. *Non-Lexical-Input[8]* is interpreted as follows:

```
if Non-Lexical-Input[8] = FORWARD then
    Old-Time-Frame = Time-Frame ∧
    Time-Frame = "after"-Old-Time-Frame
if Non-Lexical-Input[8] = BACKWARD then
    Old-Time-Frame = Time-Frame ∧
    Time-Frame = "before"-Old-Time-Frame
if Non-Lexical-Input[8] = NEUTRAL ∧ Old-Time-Frame ≠ ABSENT then
    Time-Frame = Old-Time-Frame ∧
    Old-Time-Frame = ABSENT
```

Subroutine *Temp-Time-Frame()* in Figure 6.5 illustrates this procedure.

6.4 ASL-Grammar

We describe how our prototype grammar interacts with the non-lexical parsing routine through the interface variables decribed in the previous sections. We are the first to admit that our linguistic competence in ASL is incomplete and we do not claim that this grammar is complete. We only want to show how our approach works and leave it to the linguists to develop a complete and correct ASL-grammar. Hopefully the tools invented in this thesis will be of help to them in their work.

The prototype was implemented in a Definite Clause Grammar (DCG), therefore we chose to present the ASL-grammar as DCG rewrite rules. Before discussing details of the grammar, we describe its input format and illustrate it with some examples. Then we proceed to the grammar and discuss the various topic in this order: basic sentences, establishment of references, indexing, role play, non-manual signals, time, and finally, subject deletion. The grammar rules, the dictionary and the grammar subroutines presented in this section are summarized in appendices A, B, and C.

(Note to DCG hackers: The grammar presented here is strictly a prototype, only aimed at illustrating the concepts, and we did not put emphasis on fancy coding. In a real system, one could for example, include verb type information (multi- vs. non-directional, transitivness, etc.) in argument to avoid the long and cumbersome functor names. It would make the grammar shorter and cleaner. Some meta programming can simplify the dictionary and make it more modular.)

6.4.1 Input to the ASL-Grammar

The input format was dictated by the DCG formalism; DCG grammars expect their input to be *a list* of lexical items. (Technically speaking a *parser* used for *analysis* of natural language takes as input a *grammar*, written in, say, DCG and a list of lexical items.) Following standard DCG notation, a list is delimited by '[' and ']' and list items are separated by commas. Some examples,

ASL-sentence	Input to ASL-grammar
CAROL SWIM	[carol, swim]
PEPE (establish ref. LOC2)	[pepe]
DAVID (establish ref. LOC4)	[david]
$_{LOC2}\text{TEASE}_{LOC4}$	[tease]
HAIRCUT, $_{LOC2}\text{GIVE}_{LOC4}$	[haircut, give]
$\overline{\text{INDEX(LOC2) LIKE INDEX(LOC4)}}^{q}$	[like]
$_{LOC1}\text{TEASE}_{LOC2}$ $_{LOC1}\text{TEASE}_{LOC4}$	[tease, tease]

6.4.2 Basic Sentences

We consider as basic sentences, those that do not include non-manual signals, role play or indexing, i.e., ASL sentences as they appear in isolation outside a discourse. Since there is no non-lexical information present, a meaning representation of these sentences is obtained in the same way as for spoken languages; simple parsing of lexical items by a grammar. Together with the sample dictionary in Figure 6.6, the grammar fragment below can parse basic ASL sentences.

```
sentence(S) --> sent(S).
sent(S) --> verb_less_sentence(S).
sent(S) --> np(Subj),
            non_dir_vp(Subj, S).
sent(S) --> multi_dir_vp(Subj, Vp).
non_dir_vp(Subj, Vp) --> intrans_non_dir_verb(Subj, Vp).
non_dir_vp(Subj, Vp) --> trans_non_dir_verb(Subj, Obj, Vp),
                         np(Obj).
multi_dir_vp(Vp) --> trans_multi_dir_vp(Vp).
multi_dir_vp(Vp) --> ditrans_multi_dir_vp(Dobj, Vp),
                     np(Dobj).
trans_multi_dir_vp(Vp) --> np(Subj),
                           trans_multi_dir_verb(Subj, Dobj, Vp),
                           np(Dobj).
ditrans_multi_dir_vp(Dobj, Vp) --> np(Subj),
                                   ditrans_multi_dir_verb(Subj, Iobj, Dobj, Vp)
                                   np(Iobj).
verb_less_sentence(is_has(Subj, Comp)) --> np(Np1),
                                           np(Np2).
verb_less_sentence(is_has(Np1, Ap)) --> np(Np1),
                                        ap(Ap).
np(NP) --> noun(Noun).
ap(Ap) --> adjective(Adj).
```

Some examples

In all these examples, the non-manual input would be an array of NULL's since there is no non-manual and spatial information.

Grammar input	Grammar output
[diana, student]	is_has(diana, student)
[sergio, love, diana]	love(sergio, diana)
[sergio, swim]	swim(sergio)

```
% adjectives
adjective —> [nice].
% nouns
noun(diana) —> [diana].
noun(sergio) —> [sergio].
noun(book) —> [book].
% verbs
intrans_non_dir_verb(Subj, swim(Subj)) —> [swim].
trans_non_dir_verb(Subj, Obj, love(Subj, Obj)) —> [love].
trans_multi_dir_verb(Subj, Obj, meet(Subj, Obj)) —> [meet].
ditrans_multi_dir_verb(Subj, Iobj, Dobj, give(Subj, Iobj, Dobj)) —> [give].
reverse_ditrans_multi_dir_verb(Iobj, Subj, Dobj, take(Subj, Iobj, Dobj)) —> [take].
```

Figure 6.6: A sample dictionary.

6.4.3 Establishment of Spatial References and Roles

There are two types of references: *spatial references*, which are dereferenced by indexing and *roles*, which are dereferenced by turning the head left or right. We assume that establishment of both spatial references and roles are processed sequentially. When the signer establishes a new reference, she must supply both a spatial location and a referent. The referent is naturally lexical, while the location is spatial, so the two pieces of information are separated in the visual interface and must "meet up" again in the parser. The referent must be a single noun phrase, and since no other ASL "sentence" consists of a single noun phrase it is easy for the grammar to recognize when the signer establishes a new referent. Once the grammar has parsed the referent, it retrieves the spatial location from *First-Location* and interprets the two pieces of information as follows.

```
Assume Referent = the noun phrase processed by the grammar.
If First-Ref = LEFT then Let Left = Referent.
If First-Ref = RIGHT then Let Right = Referent.
If First-Ref = LOCY, where $1 \leq Y \leq 5$, then Let LocY = Referent.
```

Pseudo code versions of the grammar fragment and the extra subroutines used by the grammar to process establishment of new references are presented below (subroutines are enclosed in curly braces). Notice that the grammar fragment does not produce any output, since establishment of new referents is not a query and therefore should not be evaluated.

```
sent(_) —> establish-reference(_).
establish-reference(_) —> np(Np)
```

```
                                        {Get-first-location(Location)},
                                        {Establish-reference(NP, Location)}.

    Subroutine : Get-first-location(out: Location)
        Location = First-Location
    End Get-first-location

    Subroutine : Establish-reference(in: Referent, Location)
        Case Location:
            LEFT:  Let Left = Referent
            RIGHT:  Let Right = Referent
            LOC1:  Let Loc1 = Referent
            LOC2:  Let Loc2 = Referent
            LOC3:  Let Loc3 = Referent
            LOC4:  Let Loc4 = Referent
            LOC5:  Let Loc5 = Referent
        End Case
    End Establish-reference
```

An example

The signer establishes a spatial reference for Diana in LOC2. The old referent in LOC2 was Jörg.

Non-manual input: [NULL, NULL, NULL, LOC2, NULL, NULL, NULL, NULL]

Grammar input	Interface Before	Interface After
[diana]	Topic = ABSENT	Topic = ABSENT
	Question = ABSENT	Question = ABSENT
Grammar output	Negation = ABSENT	Negation = ABSENT
	Loc1 = eli	Loc1 = eli
	Loc2 = jörg	Loc2 = diana
	Loc3 = ABSENT	Loc3 = ABSENT
	Loc4 = ABSENT	Loc4 = ABSENT
	Loc5 = ABSENT	Loc5 = ABSENT
	Left = ABSENT	Left = ABSENT
	Right = ABSENT	Right = ABSENT
	First-Ref = jörg	First-Ref = jörg
	Second-Ref = ABSENT	Second-Ref = ABSENT
	Third-Ref = ABSENT	Third-Ref = ABSENT
	Fourth-Ref = ABSENT	Fourth-Ref = ABSENT
	First-Location = LOC2	First-Location = LOC2
	Time-Frame = today	Time-Frame = today
	Old-Time-frame = ABSENT	Old-Time-frame = ABSENT
	last-Subject = ABSENT	last-Subject = ABSENT

6.4.4 Indexing and Role Play

When a signer uses indexing or role play, the grammar must access the previously established spatial information in order to build a semantic representation of the sentence. Recall from section 6.3.2 that the references have already been interpreted, so the grammar obtains the necessary referents by simply reading off the value of one of the variables in the interface (*First-Ref*, *Second-Ref*, *Third-Ref*, or *Fourth-Ref*). Below, we show how the different components of the parser interact to produce a semantic representation of an ASL sentence that includes indexing or role play.

```
non_dir_vp --> {Get_first_ref(Subj)},
               trans_non_dir_verb(Subj, Obj, Vp)
               {Get_second_ref(Obj)}.
multi_dir_vp(Vp) --> trans_multi_dir_vp(Vp).
multi_dir_vp(Vp) --> ditrans_multi_dir_vp(Dobj, Vp),
                     np(Dobj).
multi_dir_vp(Vp) --> reverse_ditrans_multi_dir_vp2(Dobj, Vp),
                     np(Dobj).
trans_multi_dir_vp(Vp) --> {get_first_ref(Subj)},
                           trans_multi_dir_verb(Subj, Obj, Vp),
                           {get_second_ref(Obj)}.
ditrans_multi_dir_vp(Dobj, Vp) --> {get_first_ref(Subj)},
                                   trans_multi_dir_verb(Subj, Iobj, Dobj, Vp),
                                   {get_second_ref(Iobj)}.
reverse_ditrans_multi_dir_vp2(Dobj, Vp), -->
                           {get_first_ref(Iobj)},
                           reverse_multi_dir_verb(Iobj, Subj, Dobj, Vp),
                           {get_second_ref(Subj)}.
```

```
Subroutine : Get-first-ref(out: Referent)
    Let Referent = First-Ref
End Get-first-ref

Subroutine : Get-second-ref(out: Referent)
    Let Referent = Second-Ref
End Get-second-ref
```

An example

Assume spatial references for Diana and Sergio have been established in LOC2 and LOC4 respectively. Then consider the sentence $_{LOC2}$GIVE$_{LOC4}$ BOOK (Diana gave Sergio a book).

Non-manual input: [NULL, NULL, NULL, LOC2, LOC4, NULL, NULL, NULL]

Grammar input

[give, book]

Grammar output

give(diana, sergio, book)

Interface

Topic = ABSENT
Question = ABSENT
Negation = ABSENT
Loc1 = eli
Loc2 = diana
Loc3 = ABSENT
Loc4 = sergio
Loc5 = ABSENT
Left = ABSENT
Right = ABSENT
First-Ref = diana
Second-Ref = sergio
Third-Ref = ABSENT
Fourth-Ref = ABSENT
First-Location = LOC2
Time-Frame = today
Old-Time-frame = ABSENT
Last-Subject = eli

Dual and Trial Inflection

By adding routines for accessing the variables *Third-Ref* and *Fourth-Ref* the above grammar can easily be extended to parse dual inflection. The grammar fragment below gives some examples.

```
trans_multi_dir_vp(Vp) —> {get_first-ref(Subj)},
                          trans_multi_dir_verb(Subj, Dobj1, Vp),
                          {get_second-ref(Dobj1)},
                          {get_third-ref(Subj)},
                          trans_multi_dir_verb(Subj, [Dobj1, Dobj2], Vp),
                          {get_fourth-ref(Iobj2)}.
```

```
ditrans_multi_dir_vp(Dobj, Vp) --> {get_first-ref(Subj)},
                                    ditrans_multi_dir_verb(Subj, [Iobj1, Iobj2], Dobj, Vp),
                                    {get_second-ref(Iobj1)},
                                    {get_third-ref(Subj)},
                                    ditrans_multi_dir_verb(Subj, [Iobj1, Iobj2], Dobj, Vp),
                                    {get_fourth-ref(Iobj2)}.
reverse_ditrans_multi_dir_vp(Dobj, Vp) -->
                               {get_first-ref(Iobj1)},
                               ditrans_multi_dir_verb(Subj, [Iobj1, Iobj2], Dobj, Vp),
                               {get_second-ref(Subj)},
                               {get_third-ref(Iobj2)},
                               ditrans_multi_dir_verb(Subj, [Iobj1, Iobj2], Dobj, Vp),
                               {get_fourth-ref(Subj)}.
```

```
Subroutine : Get-third-ref(out: Referent)
    Let Referent = Third-Ref
End Get-third-ref

Subroutine : Get-fourth-ref(out: Referent)
    Let Referent = Fourth-Ref
End Get-fourth-ref
```

Similarly, we can extend the grammar to parse trial inflection by adding two more variables for spatial references and corresponding routines for accessing these variables.

An example

Assume spatial references for Eli, Diana and Sergio have been established in LOC1, LOC2 and LOC3 respectively. Then consider the sentence $_{LOC1}$GIVE$_{LOC2}$ $_{LOC1}$GIVE$_{LOC3}$ BOOK (Eli gave Sergio and Diana a book).

Non-manual input: [NULL, NULL, NULL, LOC1, LOC2, LOC1, LOC3, NULL]

Grammar input	**Interface**
[give, give, book]	Topic = ABSENT
	Question = ABSENT
Grammar output	Negation = ABSENT
	Loc1 = eli
	Loc2 = sergio
give(eli, {sergio,diana}, book)	Loc3 = diana
	Loc4 = ABSENT
	Loc5 = ABSENT
	Left = ABSENT
	Right = ABSENT
	First-Ref = eli
	Second-Ref = sergio
	Third-Ref = eli
	Fourth-Ref = diana
	First-Location = LOC1
	Time-Frame = today
	Old-Time-frame = ABSENT
	Last-Subject = eli

6.4.5 Question, Negation, and Topicalization

Questions, negated sentences, and topicalized sentences are all accompanied by non-manual signals so in order to parse these, the grammar must gain access to the non-manual information stored in the interface variables. These routines are similar to the ones we saw above and we list them below (*Non-manual-topic, Non-manual-question, Non-manual-negated*). Yes-no questions and negated sentences are produced by giving a non-manual signal together with any of the declarative sentence patterns (topicalization included) so no new sentence patterns are introduced and only two new clauses where one checks for the presence of the appropriate non-manual signal are needed to parse yes-no questions and negated sentences. An interrogative question is posed by including an interrogative sign at the end of a sentence. Also, some part of speech is deleted from the surface structure of an interrogative question so some new rewrite rules are needed. Topicalization affects the order in which signs are produced and since both objects and verb phrases can be topicalized the complexity of the grammar will increase considerably. Below we show some of the rewrite rules necessary for parsing topicalized sentences, negated sentences and questions. We do not include any examples of interrogative sentences since they do not illustrate any new points.

```
sentence(S) —> {Non-manual-question(true)},
                sent(S).
sentence(S) —> {Non-manual-negation(true)},
                sent(S).
sent(S) —> {Non-manual-topic(true)},
           topicalized_sent(S).
topicalized_sent(Vp) —> np(Dobj),
           non_dir_vp_topicalized_dobj(Dobj, Vp).
topicalized_sent(Vp) —> non_dir_vp_topicalized_vp(Subj, Vp),
           {get_second_ref(Subj)}.
topicalized_sent(Vp) —> np(Dobj),
                        multi_dir_vp_topicalized_dobj(Dobj, Vp).
non_dir_vp_topicalized_dobj(Dobj, Vp) —> {get_first_ref(Subj)},
                                         non_dir_verb(Subj, Dobj, Vp).
non_dir_vp_topicalized_vp(Subj, Vp) —> non_dir_verb(Subj, Obj, Vp),
                                         {get_first_ref(Obj)}.
multi_dir_vp_topicalized_dobj(Dobj, Vp) —> ditrans_multi_dir_verb(Subj, Iobj, Dobj, Vp)
                                           {get_first_ref(Subj)},
                                           {get_second_ref(Iobj)}.
```

```
Subroutine : Non-manual-topic(out: Non-manual-flag)
    If Topic = PRESENT then
        Let Non-manual-flag = TRUE
    Else
        Let Non-manual-flag = FALSE
    End If
End Non-manual-topic

Subroutine : Non-manual-question(out: Non-manual-flag)
    If Question = PRESENT then
        Let Non-manual-flag = TRUE
    Else
        Let Non-manual-flag = FALSE
    End If
End Non-manual-question

Subroutine : Non-manual-negation(out: Non-manual-flag)
    If Negation = PRESENT then
        Let Non-manual-flag = TRUE
    Else
        Let Non-manual-flag = FALSE
    End If
End Non-manual-negation
```

Note on the semantics of TRUE and FALSE: In logic programming terms, the semantics of

TRUE is that the goal succeeds, while the semantics of FALSE is that the goal fails.

An example

Assume spatial references for Diana and Sergio have been established in LEFT and RIGHT respectively. Then consider the sentence BOOK, $_{RIGHT}$GIVE$_{LEFT}$ (Sergio gave Diana a book).

Non-manual input: [TOP, NULL, NULL, RIGHT, LEFT, NULL, NULL, NULL]

Grammar input

[book, give]

Grammar output

give(sergio, diana, book)

Interface

Topic = PRESENT
Question = ABSENT
Negation = ABSENT
Loc1 = eli
Loc2 = ABSENT
Loc3 = ABSENT
Loc4 = ABSENT
Loc5 = ABSENT
Left = diana
Right = sergio
First-Ref = sergio
Second-Ref = diana
Third-Ref = ABSENT
Fourth-Ref = ABSENT
First-Location = LOC5
Time-Frame = yesterday
Old-Time-frame = ABSENT
Last-Subject = eli

6.4.6 Time

In section 6.3.2, we discussed how the signer can temporarilly switch the time frame non-manually by shifting her body slightly backward or forward. Here, we will concentrate on how the signer establishes a new time frame by using lexical items. When the signer establishes a new time frame via a lexical item, it is considered a "permanent" change in the sense that it is assumed that the signer will not talk more about what happened in the previous time frame and therefore, there is no need to remember it. If the signer should wish to go back to the previous time frame, she would have to do this by explicitly giving the sign.

The signer uses a time-marker to establish a new time frame and since time markers are lexical items, they are processed by the ASL-grammar, i.e., when the signer uses a time marker to change the time frame, the grammar must record the new time frame in the interface variable . Time markers can be specific or general; if the signer uses a specific time marker, it is stored in *Time-Frame* as is. If the signer uses a general time marker (PAST, FUTURE), the parser must modify the *Time-frame* to "before" or "after" current time frame (how to implement "before" and "after" is up to the user). The grammar activates the subroutine *New-Time-Frame()* every time it encounters a lexical item that it recognises as a time-marker. We illustrate this below.

```
sent(S) —> spec_time_marker(Tm),
           {New-time-frame(Tm)},
           multi_dir_vp(Vp).
trans_multi_dir_vp(Vp) —> {get_first_ref(Subj)},
                          trans_multi_dir_verb(Subj, Obj, Vp),
                          {get_second_ref(Obj)},
                          general_time_marker(Tm),
                          {new-time-frame(Tm).}
```

```
Subroutine : New-Time-Frame(IN: Time-Marker)
   If Time-Marker = PAST then
      Let Time-Frame = "before"-Time-Frame
   Else If Time-Marker = FUTURE then
      Let Time-Frame = "after"-Time-Frame
   Else
      Let Time-Frame = Time-Marker
   End If
End New-Time-Frame
```

The user (e.g., database designer) decides where to make use of the temporal information and this decision must be reflected in the system's dictionary, so when the grammar encounters a sign (mostly verbs) which demands temporal information for its semantic representation, the grammar calls the routine *Get-time()* which gets the current time frame from the *Time-frame* variable in the interface. We give a couple of examples of dictionary entries below.

```
trans_multi_dir_verb(Subj, Obj, meet(Subj, Obj, Time)) —> [meet],
                                                          {Get-time(Time)}.
intrans_non_dir_verb(Subj, swim(Subj, Time)) —> [swim],
                                                {Get-time(Time)}.
```

```
Subroutine : Get-time(out: Time)
    Let Time = Time-frame
End Get-time
```

An example

Assume spatial references for Eli and Diana have been established in LOC1 and LOC2 respectively. Then consider the sentence $_{LOC1}$MEET$_{LOC2}$ PAST (Eli and Diana met), i.e., the signer establishes a new time frame.

Non-manual input: [NULL, NULL, NULL, LOC1, LOC2, NULL, NULL, NULL]

Grammar input	**Interface Before**	**Interface After**
[meet, past]	Topic = ABSENT	Topic = ABSENT
	Question = ABSENT	Question = ABSENT
Grammar output	Negation = ABSENT	Negation = ABSENT
	Loc1 = eli	Loc1 = eli
	Loc2 = diana	Loc2 = diana
meet(eli, diana, 'before today')	Loc3 = ABSENT	Loc3 = ABSENT
	Loc4 = ABSENT	Loc4 = ABSENT
	Loc5 = ABSENT	Loc5 = ABSENT
	Left = ABSENT	Left = ABSENT
	Right = ABSENT	Right = ABSENT
	First-Ref = eli	First-Ref = eli
	Second-Ref = diana	Second-Ref = diana
	Third-Ref = ABSENT	Third-Ref = ABSENT
	Fourth-Ref = ABSENT	Fourth-Ref = ABSENT
	First-Location = LOC2	First-Location = LOC2
	Time-Frame = today	Time-Frame = before today
	Old-Time-frame = ABSENT	Old-Time-frame = ABSENT
	Last-Subject = eli	Last-Subject = eli

6.4.7 Subject Deletion

It is common in ASL to omit the subject of a sentence if it is the same as the subject of the previous sentence and if the verb of the current sentence is non-directional. In order to obtain a subject referent when the signer chooses to use this technique, the grammar must keep track of every articulated subject it encounters. In our prototype this is accomplished by executing the routine *Store-last-subject()* which records the current subject in the interface variable *Last-subject* after every sentence with an articulated subject. Below we give some examples of how to include the routine.

```
trans_multi_dir_vp(Vp) —> {get_first-ref(Subj)},
                          trans_multi_dir_verb(Subj, Dobj1, Vp),
                          {get_second-ref(Dobj1)},
                          {get_third-ref(Subj)},
                          trans_multi_dir_verb(Subj, [Dobj1, Dobj2], Vp),
                          {get_fourth-ref(Iobj2)},
                          {store-last-subject}.
non_dir_vp(Subj, Vp) —> intrans_non_dir_verb(Subj, Vp),
                        {store-last-subject}.
```

```
Subroutine : Store-last-subject(in: Subject)
    Let Last-subject = Subject
End Store-last-subject
```

An example

Assume that the old subject is Eli and that the signer changes the subject to Diana. Assume that the spatial location for Diana is LOC2. Then consider the sentence DIANA (INDEX) SWIM (Diana is swimming).

Non-manual input: [NULL, NULL, NULL, LOC2, NULL, NULL, NULL, NULL]

Grammar input	**Interface Before**	**Interface After**
[swim]	Topic = ABSENT	Topic = ABSENT
	Question = ABSENT	Question = ABSENT
Grammar output	Negation = ABSENT	Negation = ABSENT
	Loc1 = eli	Loc1 = eli
	Loc2 = diana	Loc2 = diana
swim(diana)	Loc3 = ABSENT	Loc3 = ABSENT
	Loc4 = ABSENT	Loc4 = ABSENT
	Loc5 = ABSENT	Loc5 = ABSENT
	Left = ABSENT	Left = ABSENT
	Right = ABSENT	Right = ABSENT
	First-Ref = diana	First-Ref = diana
	Second-Ref = ABSENT	Second-Ref = ABSENT
	Third-Ref = ABSENT	Third-Ref = ABSENT
	Fourth-Ref = ABSENT	Fourth-Ref = ABSENT
	First-Location = LOC2	First-Location = LOC2
	Time-Frame = today	Time-Frame = today
	Old-Time-frame = ABSENT	Old-Time-frame = ABSENT
	Last-Subject = eli	Last-Subject = diana

Later, when the grammar parses a subjectless sentence, it reads off the value of *Last-subject* and uses it as the subject referent. The code below illustrates how to parse subjectless sentences.

```
sent(S) —> missing _subject(Vp).
missing_subject_vp(Vp) —> intrans_non_dir_verb(Subj, Vp),
                          {get_last_subject(Subj)}.
missing_subject_vp(Vp) —> trans_non_dir_verb(Subj, Obj, Vp),
                          {get_first_ref(Obj)}
                          {get_last_subject(Subj)}.
```

```
Subroutine : Get-last-subject(in-out: Subject)
    Let Subject = Last-Subject
End Get-last-subject
```

An example

Assume that the last subject mentioned was Diana. Then consider the sentence SWIM (Diana is swimming).

Non-manual input: [NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL]

Grammar input	**Interface Before**	**Interface After**
[swim]	Topic = ABSENT	Topic = ABSENT
	Question = ABSENT	Question = ABSENT
Grammar output	Negation = ABSENT	Negation = ABSENT
	Loc1 = eli	Loc1 = eli
	Loc2 = diana	Loc2 = diana
swim(diana)	Loc3 = ABSENT	Loc3 = ABSENT
	Loc4 = ABSENT	Loc4 = ABSENT
	Loc5 = ABSENT	Loc5 = ABSENT
	Left = ABSENT	Left = ABSENT
	Right = ABSENT	Right = ABSENT
	First-Ref = ABSENT	First-Ref = ABSENT
	Second-Ref = ABSENT	Second-Ref = ABSENT
	Third-Ref = ABSENT	Third-Ref = ABSENT
	Fourth-Ref = ABSENT	Fourth-Ref = ABSENT
	First-Location = ABSENT	First-Location = ABSENT
	Time-Frame = today	Time-Frame = today
	Old-Time-frame = ABSENT	Old-Time-frame = ABSENT
	Last-Subject = diana	Last-Subject = diana

The ASL grammar as presented above only *parses* ASL and does not produce a semantic representation of ASL sentences. In order to do produce a semantic represenation, the rules must be modified and below we discuss the particular representation, which we designed as part of LM and its assosiated query language.

6.5 Translating ASL into LM

Since ASL does not have explicit determiners we had to look for other clues for translating ASL sentences into LM: surface structure, type of verb, non-manual signals, etc. We suggest the following translations:

6.5.1 Elementary Statements

Elementary statements in ASL are translated directly into LM exactly as in English. Some examples:

GARFIELD CAT = iscat(Garfield)

GARFIELD STRIPED = isstriped(Garfield)

DIANA GIVE BRIGITTE GARFIELD = give(Diana, Garfield, Brigitte)

6.5.2 Determiners

It is unclear how the definite and indefinite/distinction is made in ASL. Some researchers claim that ASL does not have determiners [27, 59], while others claim that ASL does have determiners [60]. Those who claim that ASL has determiners have not yet identified how the indefinite/definite distinction is made. R. Wilbur [59, p.235] writes:

> "English uses the articles *a/an* and *the* , whereas ASL uses pointing and specific locations in space to make the distinction between definite ("the") and indefinite ("a")."

Based on this quote we made the following assumptions about the indefinite/definite distinction in the input to our system: If the signer gives the sign for a noun, we assume that she is introducing a new referent to the discourse, and that this is similar to using the indefinite article in English. If she uses a previously established reference point, we assume this is similar to using the definite article in English. We emphasize that this is a simplifying assumption that may change once the ASL linguists have investigated the topic further.

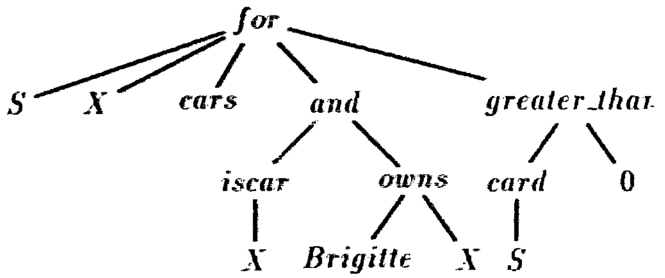
Indefinite Articles

We translate indefinite type sentences, i.e., sentences where the sign for the noun is given, into LM's *for* representation for *a*:

$$for(S,X,D,and(F1,F2),greater\_than(card(S),0)))$$

Consider the following example:

BRIGITTE OWN CAR. (Brigitte owns a car.)

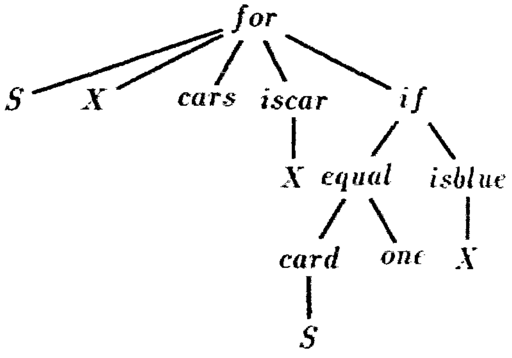


The Definite Article

In English, the singular definite article presupposes existence and uniqueness of a noun's referent, and in a database system with an English frontend, failure to meet either of these two assumptions is detected when the *for* formula for *the*: $for(S,X,D,F1,if(equal(card(S),1),F2))$ is evaluated with respect to the knowledge base. We assume that using an already established reference point in ASL, is similar to using the definite article in English. Consequently, if the signer uses a non-existing reference point, she is violating the existence assumption introduced by the definite article, and this type of violation is detected during the parse since the discourse unit cannot substitute a referent for that particular reference. The parse is immediately interrupted, no internal representation is produced and the reason for failure is communicated to the user. One could introduce a new logic value for these violations but we choose not to, since this is a grammatic and not semantic mistake on part the users part.

Since we cannot check the uniqueness assumption during the parse, a successfully parsed query is translated into the usual *for* representation for *the* and evaluated with respect to the knowledge base. For example, if the signer in our previous example intends to talk more about Brigitte's car, she establishes a reference point for the car, which she can later index. As in

CAR (index) BLUE. =The car is blue.



ASL provides us with an easy way of distinguishing between the two types of failure and we can inform the user of which assumption she violated; if the query reaches the database consultation stage, she violated the uniqueness assumption; if the query is interrupted in the parsing stage, she violated the existence assumption. In, for example, English, we cannot tell which assumption is violated and both types of violations will result in a complete semantic representation, which evaluates to *pointless* in the database.

-SOME Verbs

Statements with a "-some verb" are simply translated into the usual *for* formula for *some*, i.e., $some(X, F1, F2) = for(S, X, D, and(F1, F2), greater\_than(card(S), 1)))$

6.5.3 ASL and LM's Quantification Hierarchy

The basic sentence patterns in ASL are:

1. subject + intransitive verb

   e.g., HE SWIM. (He is swimming.)

2. subject + complement

   e.g., WEI DOCTOR. (Wei is a doctor.)

   e.g., CAR FLAT TIRE. (The car has a flat tire.)

3. subject + verb + object

   e.g., ME READ BOOK. (I read a book.)

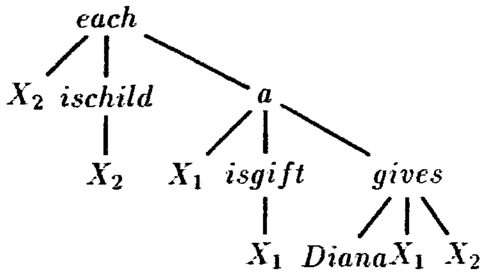
4. subject + verb + indirect object + object
   e.g., TINA GIVE ERIC HAIRCUT. (Tina gave Eric a haircut.)

In general, the basic ASL sentences conform to the quantification hierarchy proposed by Colmerauer (see 2.1.2): Quantifications introduced by the subject dominate those introduced by a complement. If a verb has two complements, the rightmost dominates the leftmost. Recall that this second hypothesis caused some problems in English and we concluded: if the quantification of the recipient is not *a* or *the*, this quantification should dominate regardless of position, i.e., "Diana gave each child a gift" and "Diana gave a gift to each child" translate into the same *for* formula. Next we discuss how ASL offers a solution to this problem.

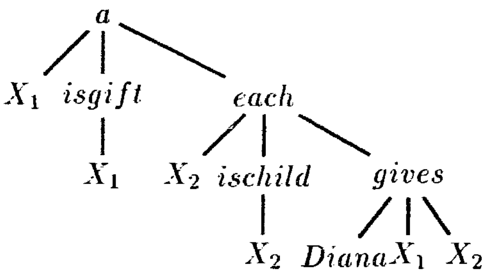
-EACH and -ALL Verbs

Recall the two possible represenations of "Diana gave a gift to each child."

(1)



(2)



The first representation implies that each child got a separate gift, while the second implies that there was only one gift and the children were sharing this gift. For English, we assume that the author intended the first interpretation, but we cannot be 100% certain. ASL can convey both these ideas unambigously by adding different affixes to the verb: the -EACH affix conveys the idea that each child got a separate gift, while the -ALL affix conveys the idea that they shared one gift. We propose that if a sentence contains an -EACH verb, the quantification of the recipient (i.e., each) dominates the quantification of the other complement. If the a sentence contains an -ALL verb, the quantification of the object dominates the quantification of the recipient. I.e., DIANA GIVE-EACH CHILD GIFT is translated into (1) above and GIFT, DIANA GIVE-ALL CHILDREN is translated into (2) above.

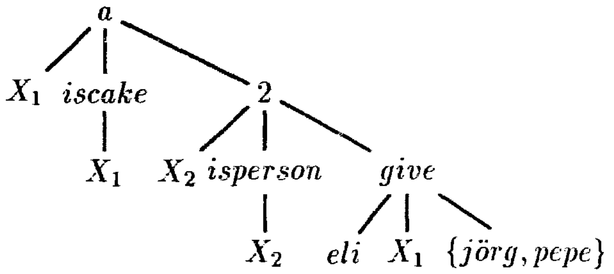
Two Objects—Dual Inflection

Multi-directional verbs can indicate more than one object (most signers only use dual inflection, but there are some which also use trial) and the implication is that the action applies to both objects collectively (similar to -ALL verbs). Recall, dual inflection can be accomplished in two ways; either by first moving to one reference point and then quickly "bouncing" to the other reference point, or the signer can repeat the verb twice, each time with a different endpoint (starting point for reverse agreement verbs). Consider the following examples:

${}_{eli}\text{GIVE}_{j\ddot{o}rg,pepe}$ CAKE or

${}_{eli}\text{GIVE}_{j\ddot{o}rg}$ ${}_{eli}\text{GIVE}_{pepe}$ CAKE

Since the action applies to both objects collectively, we propose to process the complements from right to left. We also suggest that the duality concept translates into an integer *for* formula. Hence, the above sentences translate into the following *for* formula:



$$
\begin{aligned}
= for(&S1, X1, cakes, and(iscake(X1),\\
&\quad for(S2, X2, people, and(isperson(X2),\\
&\qquad give(eli, X2, \{j\ddot{o}rg, pepe\})),\\
&\quad equal(card(S2), 2))),\\
&greater\_than(card(S1), 1))
\end{aligned}
$$

Topicalized Sentences

Recall the common topicalization patterns from section 5.3.1:

1. intransitive verb, subject
   e.g., SWIM, HE. (He is swimming.)

2. complement, subject
   e.g., DOCTOR, WEI. (Wei is a doctor.)

e.g., FLAT TIRE, CAR. (The car has a flat tire.)

3. object, subject + verb

   e.g., BOOK, ME READ. (I read a book.)

4. verb + object, subject

   e.g., READ YOU, ME. (I am reading to you.)

5. object, subject + verb + indirect object

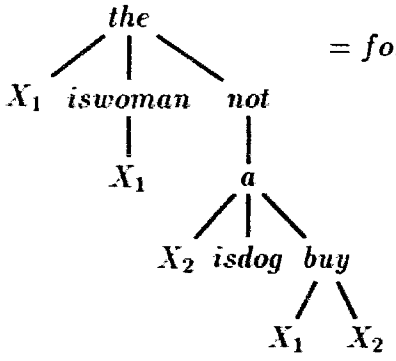
   e.g., HAIRCUT, TINA GIVE ERIC. (Tina gave Eric a haircut.)

In terms of constituents (e.g., subject, complement, recipient) the hierarchy stays the same, but the order in which the quantifications appear in a sentence has changed, so in terms of order of appearance, there is obviously a difference. Type 1 sentences can only have one quantification so there is no problem. Most type 2 sentences are elementary statements, so they translate directly into LM. Type 2 sentences that are not elementary statements, but have subject and complement, e.g., FLAT TIRE, CAR, should process the quantifications from right to left. The same is valid for sentences of type 3 and 4. Type 5 sentences are subject to our analysis of two complements of the verb, so the order in which to process the complements depends on the inflection of the verb.

6.5.4 Negation

Sentences that contain a non-manual negating signal and possibly a negating sign different from NONE, translate directly into the quantifer representation Colmerauer suggested for negation.

e.g., WOMAN (INDEX) $\overset{n}{\overline{\text{BUY DOG}}}$ (The woman didn't buy a dog.)

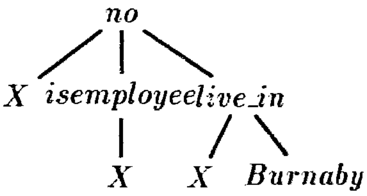


$$
\begin{aligned}
= for(&S_1, X_1, women, iswoman(X_1),\\
&if(equal(card(S_1),1),\\
&\quad not(for(S_2, X_2, dogs,\\
&\qquad and(isdog(X_2), buy(X_1, X_2)),\\
&\qquad greater\_than(card(S_2,0))\ ))))
\end{aligned}
$$

NONE

The negating sign NONE translates into the *for* for *no*. For example,

$$\overline{\text{EMPLOYEE NONE LIVE IN BURNABY}}^{n}$$



$$\begin{aligned} = for(&S, X, employees,\\ &and(isemployee(X), live\_in(X, Burnaby))\\ &equal(card(S), 0)\ ) \end{aligned}$$

Some sample ASL queries' translation into complete LM formulae are listed in appendix D.

Chapter 7

Conclusion

7.1 Contributions

We believe the main contributions of our work to be:

- a computational model of (a subset of) ASL
- a testbed of its use within ASL consultation of deductive databases
- the motivation of the use of multiple logical values, both explicit and implicit, to provide more helpful answers to natural language querying of deductive databases.
- a rigorous characterization of multi-valued deductive databases for cooperative answering
- a rigorous logical system, LM, underlying both our computational model of ASL and our database query language.

Our model of ASL accounts for its parallel nature and represents, to the best of our knowledge, the first model for parallel human languages. The possibility of interesting and mutual feedback between parallel and sequential human languages has been one of the by-products of our research. The main strength in our approach, from our point of view, is that our solutions are at the same time integrated in a cohesive formal logical system with rigorously defined syntax and semantics, but also have the flexibility to be adapted to different contexts. For instance, our cooperative answering mechanisms can be introduced to systems with other kinds of front end and likewise, our model for ASL may be used for other applications than database consultation.

7.2 Limitations and Future Work

At present, our ASL interface cannot distinguish between the two following two sentences:

$\text{WOMAN } \overset{n}{\overline{\text{BUY DOG.}}}$ (The woman didn't buy the dog.)

$\overset{n}{\overline{\text{WOMAN}}} \text{ BUY DOG.}$ (It wasn't the woman who bought the dog.)

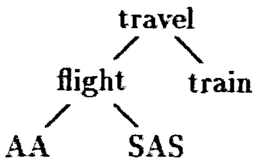
But, for an interface to computer applications, it is not necessary to distinguish between them, since they are basically syntactic variants of each other with a shift in emphasis, while the semantic content is the same. In a translation system, however, it would be interesting to distinguish between the two sentences. In such a system one would have to synchronice the non-manual and the lexical information such that the analyser would know which part of the sentence the non-manual signal accompanies. This requires a more sophisticated model of ASL and we would have to develop a much more sophisticated semantic representation of ASL than what we have presented here.

Future work in cooperative answering will concentrate on including the other problem areas (generalizations, user's beliefs and expectations) into our formalism.

We expect it to be relatively easy to account for generallizations through our type hierachy. Recall the query

AmericanAirlines_flight(burbank, dulles, at_10am)

from the introduction (section 1.3.5). If we have a type hierarchy like this



and similar hierarchies for the arguments and if the inital query fails, we traverse the hierarchies sideways to offer alternative solutions or upwards to make the query more general.

Another area of cooperative answering that we have not dealt with, is inclusion of information on related topics. For example, if somebody asks for a flight between Oslo and Vancouver, one can include, for example, the price and departure time (1.3.5). If we replace our simple types with *Typed Feature Structures* (TFS) (see [3, 50, 54]), where each constant's associated type has a list of feature/value pairs. If we let the features of our Oslo–Vancouver flight be COST and DEP_TIME they will surface through the unification process, and magic, we have more information in our answer.

Appendix A

ASL Grammar

```
% Basic sentence patterns

sentence(S) —> sent(S).
sent(S) —> verb_less_sentence(S).
sent(S) —> np(Subj),
           non_dir_vp(Subj, S).
sent(S) —> multi_dir_vp(Subj, Vp).
non_dir_vp(Subj, Vp) —> intrans_non_dir_verb(Subj, Vp).
non_dir_vp(Subj, Vp) —> trans_non_dir_verb(Subj, Obj, Vp),
                        np(Obj).
multi_dir_vp(Vp) —> trans_multi_dir_vp(Vp).
multi_dir_vp(Vp) —> ditrans_multi_dir_vp(Dobj, Vp),
                    np(Dobj).
trans_multi_dir_vp(Vp) —> np(Subj),
                          trans_multi_dir_verb(Subj, Dobj, Vp),
                          np(Dobj).
ditrans_multi_dir_vp(Dobj, Vp) —> np(Subj),
                                  ditrans_multi_dir_verb(Subj, Iobj, Dobj, Vp)
                                  np(Iobj).
verb_less_sentence(is_has(Subj, Comp)) —> np(Np1),
                                          np(Np2).
verb_less_sentence(is_has(Np1, Ap)) —> np(Np1),
                                       ap(Ap).
np(NP) —> noun(Noun).
ap(Ap) —> adjective(Adj).
```

```
% establish a spatial reference or role

sent(_) —> establish-reference(_).
establish-reference(_) —> np(Np)
                          {Get-first-location(Location)},
                          {Establish-reference(NP, Location)}.


% indexing and role play

non_dir_vp —> {Get_first_ref(Subj)},
               trans_non_dir_verb(Subj, Obj, Vp)
               {Get_second_ref(Obj)}.
multi_dir_vp(Vp) —> trans_multi_dir_vp(Vp).
multi_dir_vp(Vp) —> ditrans_multi_dir_vp(Dobj, Vp),
                    np(Dobj).
multi_dir_vp(Vp) —> reverse_ditrans_multi_dir_vp2(Dobj, Vp),
                    np(Dobj).
trans_multi_dir_vp(Vp) —> {Get_first_ref(Subj)},
                          trans_multi_dir_verb(Subj, Obj, Vp),
                          {Get_second_ref(Obj)}.
ditrans_multi_dir_vp(Dobj, Vp) —> {Get_first_ref(Subj)},
                                  trans_multi_dir_verb(Subj, Iobj, Dobj, Vp),
                                  {Get_second_ref(Iobj)}.
reverse_ditrans_multi_dir_vp2(Dobj, Vp), —>
                           {Get_first_ref(Iobj)},
                           reverse_multi_dir_verb(Iobj, Subj, Dobj, Vp),
                           {Get_second_ref(Subj)}.


% dual and trial inflection

trans_multi_dir_vp(Vp) —> {Get_first-ref(Subj)},
                          trans_multi_dir_verb(Subj, Dobj1, Vp),
                          {Get_second-ref(Dobj1)},
                          {Get_third-ref(Subj)},
                          trans_multi_dir_verb(Subj, [Dobj1, Dobj2], Vp),
                          {Get_fourth-ref(Iobj2)}.


ditrans_multi_dir_vp(Dobj, Vp) —> {get_first-ref(Subj)},
                                  ditrans_multi_dir_verb(Subj, [Iobj1, Iobj2], Dobj, Vp),
                                  {Get_second-ref(Iobj1)},
                                  {Get_third-ref(Subj)},
                                  ditrans_multi_dir_verb(Subj, [Iobj1, Iobj2], Dobj, Vp),
                                  {Get_fourth-ref(Iobj2)}.
```

```
reverse_ditrans_multi_dir_vp(Dobj, Vp) —>
                                {get_first-ref(Iobj1)},
                                ditrans_multi_dir_verb(Subj, [Iobj1, Iobj2], Dobj, Vp),
                                {Get_second-ref(Subj)},
                                {Get_third-ref(Iobj2)},
                                ditrans_multi_dir_verb(Subj, [Iobj1, Iobj2], Dobj, Vp),
                                {Get_fourth-ref(Subj)}.


% Yes-no question

sentence(S) —> {Non-manual-question(true)},
               sent(S).


% Negated sentence

sentence(S) —> {Non-manual-negation(true)},
               sent(S).


% Topicalization sent(S) —> {Non-manual-topic(true)},
                            topicalized_sent(S).
topicalized_sent(Vp) —> np(Dobj),
                            non_dir_vp_topicalized_dobj(Dobj, Vp).
topicalized_sent(Vp) —> non_dir_vp_topicalized_vp(Subj, Vp),
                            {get_second_ref(Subj)}.
topicalized_sent(Vp) —> np(Dobj),
                        multi_dir_vp_topicalized_dobj(Dobj, Vp).
non_dir_vp_topicalized_dobj(Dobj, Vp) —> {get_first_ref(Subj)},
                                          non_dir_verb(Subj, Dobj, Vp).
non_dir_vp_topicalized_vp(Subj, Vp) —> non_dir_verb(Subj, Obj, Vp),
                                        {get_first_ref(Obj)}.
multi_dir_vp_topicalized_dobj(Dobj, Vp) —> ditrans_multi_dir_verb(Subj, Iobj, Dobj, Vp)
                                            {get_first_ref(Subj)},
                                            {get_second_ref(Iobj)}.


% Set new time frame

sent(S) —> spec_time_marker(Tm),
           {New-time-frame(Tm)},
           multi_dir_vp(Vp).
trans_multi_dir_vp(Vp) —> {Get_first_ref(Subj)},
                          trans_multi_dir_verb(Subj, Obj, Vp),
                          {Get_second_ref(Obj)},
                          general_time_marker(Tm),
                          {New-time-frame(Tm).}
```

```
% Subject deletion; Store last subject

trans_multi_dir_vp(Vp) —> {Get_first-ref(Subj)},
                          trans_multi_dir_verb(Subj, Dobj1, Vp),
                          {Get_second-ref(Dobj1)},
                          {Get_third-ref(Subj)},
                          trans_multi_dir_verb(Subj, [Dobj1, Dobj2], Vp),
                          {Get_fourth-ref(Iobj2)},
                          {Store-last-subject}.
non_dir_vp(Subj, Vp) —> intrans_non_dir_verb(Subj, Vp),
                        {Store-last-subject}.


% Subject deletion; Retrieve last subject

sent(S) —> missing _subject(Vp).
missing_subject_vp(Vp) —> intrans_non_dir_verb(Subj, Vp),
                          {Get_last_subject(Subj)}.
missing_subject_vp(Vp) —> trans_non_dir_verb(Subj, Obj, Vp),
                          {Get-first-ref(Obj)}
                          {Get_last_subject(Subj)}.
```

Appendix B

ASL Dictionary

```
% adjectives

adjective —> [nice].


% nouns

noun(diana) —> [diana].
noun(sergio) —> [sergio].
noun(book) —> [book].


% verbs

intrans_non_dir_verb(Subj, swim(Subj)) —> [swim].
trans_non_dir_verb(Subj, Obj, love(Subj, Obj)) —> [love].
trans_multi_dir_verb(Subj, Obj, meet(Subj, Obj)) —> [meet].
ditrans_multi_dir_verb(Subj, Iobj, Dobj, give(Subj, Iobj, Dobj)) —> [give].
reverse_ditrans_multi_dir_verb(Iobj, Subj, Dobj, take(Subj, Iobj, Dobj)) —> [take].


% Retrieve time frame

trans_multi_dir_verb(Subj, Obj, meet(Subj, Obj, Time)) —> [meet],
                                                          {Get-time(Time)}.
intrans_non_dir_verb(Subj, swim(Subj, Time)) —> [swim],
                                                 {Get-time(Time)}.
```

Appendix C

Grammar Subroutines

```
Subroutine : Get-first-location(out: Location)
    Location = First-location
End Get-first-location

Subroutine : Establish-reference(in: Referent, Location)
    Case Location:
        LEFT: Let Left = Referent
        RIGHT: Let Right = Referent
        LOC1: Let Loc1 = Referent
        LOC2: Let Loc2 = Referent
        LOC3: Let Loc3 = Referent
        LOC4: Let Loc4 = Referent
        LOC5: Let Loc5 = Referent
    End Case
End Establish-reference

Subroutine : Get-first-ref(out: Referent)
    Let Referent = First-Ref
End Get-first-ref

Subroutine : Get-second-ref(out: Referent)
    Let Referent = Second-Ref
End Get-second-ref

Subroutine : Get-third-ref(out: Referent)
    Let Referent = Third-Ref
End Get-third-ref
```

```
Subroutine : Get-fourth-ref(out: Referent)
    Let Referent = Fourth-Ref
End Get-fourth-ref


Subroutine : Non-manual-topic(out: Non-manual-flag)
    If Topic = PRESENT then
        Let Non-manual-flag = TRUE
    Else
        Let Non-manual-flag = FALSE
    End If
End Non-manual-topic


Subroutine : Non-manual-question(out: Non-manual-flag)
    If Question = PRESENT then
        Let Non-manual-flag = TRUE
    Else
        Let Non-manual-flag = FALSE
    End If
End Non-manual-question


Subroutine : Non-manual-negation(out: Non-manual-flag)
    If Negation = PRESENT then
        Let Non-manual-flag = TRUE
    Else
        Let Non-manual-flag = FALSE
    End If
End Non-manual-negation


Subroutine : New-Time-Frame(IN: Time-Marker)
    If Time-Marker = PAST then
        Let Time-Frame = “before”-Time-Frame
    Else If Time-Marker = FUTURE then
        Let Time-Frame = “after”-Time-Frame
    Else
        Let Time-Frame = Time-Marker
    End If
End New-Time-Frame
```

```
Subroutine : Get-time(out: Time)
    Let Time = Time-frame
End Get-time
```

```
Subroutine : Store-last-subject(in: Subject)
    Let Last-subject = Subject
End Store-last-subject
```

```
Subroutine : Get-last-subject(in-out: Subject)
    Let Subject = Last-Subject
End Get-last-subject
```

Appendix D

Some Sample ASL Queries in LM

1. $\overset{q}{\overline{\text{JÖRG BORN WHERE?}}}$ ("Where was Jörg born?")
   $= those(X, country, born\_in(X, jörg))$

2. $\overset{q}{\overline{\text{BRIGITTE OWN CAR (ESTAB.)?}}}$ ("Does Brigitte own a car?")
   $= for(S, X, cars, and(iscar(X), own(brigitte, X)), greater\_than(cardinality(S), 0)))$

3. $\overset{q}{\overline{\text{CAR (INDEX) RED?}}}$ ("Is the car red?")
   $= for(S, X, cars, iscar(X), if(equal(cardinality(S), 1), isred(X)))$.

4. $\overset{q}{\overline{\text{ALICJA, DIANA WORK WITH WHO?}}}$ ("Who do Alicja and Diana work with?"
   $= those(X, people, work\_with([alicja, diana, X))$

5. $\overset{q}{\overline{\text{ADMINISTRATIVE EMPLOYEE LIVE\_IN BURNABY?}}}$ ("Does any administrative employee live in Burnaby?")
   $= for(S, X, employee, and(admin(X), employee(X), live\_in(burnaby, X)),$
   $greater\_than(cardinality(S), 0))$

6. $\overset{q}{\overline{{}_{eli}\text{GIVE}_{jörg,pepe}\ \text{CAKE?}}}$ ("Did Eli give Jörg and Pepe a cake?")
   $= for(S1, X1, cakes, and(iscake(X1),$
   $for(S2, X2, people, and(isperson(X2),$
   $give(eli, X2, \{jörg, pepe\})),$
   $equal(card(S2), 2))),$
   $greater\_than(card(S1), 1))$

7. $\overset{t}{\overline{\text{GIFT,}}}\ \overset{q}{\overline{\text{DIANA GIVE WHO?}}}$ ("To whom did Diana give a gift?")
   $those(X, people, for(S, X, gift, give(diana, X, gift),$
   $greater\_than(cardinality(S), 0)))$

Bibliography

[1] H. Abramson and V. Dahl. *Logic Grammars.* Springer-Verlag, 1989.

[2] Ivan Bratko. *Prolog; Programming for Artificial Intelligence.* Addison-Wesley Publishers Ltd., second edition, 1990.

[3] B. Carpenter. *The Logic of Typed feature Structures: with applications to unification grammars, logic programs, and constraint resolution.* Cambridge University Press, 1992.

[4] L. Cholvy. Answering Queries Addressed to a Rule Base. *Revue d'intelligence artificielle*, 4(1):79–98, 1990.

[5] L. Cholvy and R. Demolombe. Querying a Rule Base. In L. Kershberg, editor, *Expert Database Systems.* Tysons Corner, 1987.

[6] N. Chomsky. Three Models for the Description of Language. *IRE Trans. on Information Theory*, 2(3):113–124, 1956.

[7] W. W. Chu, Q. Chen, and R. Lee. Cooperative Answering via Type Abstraction Hierarchy. In S. M. Deen, editor, *Cooperating Knowledge Based Systems 1990*, pages 271–290. Springer Verlag, 1991.

[8] W. W. Chu, Q. Chen, and R. Lee. A Structured Approach to Cooperative Answering. *IEEE Transactions on Knowledge and data Engineering*, In press.

[9] A. Colmerauer. Metamorphosis Grammars. In Bolc L., editor, *Natural Language Computer Systems*, pages 133–188. Springer-Verlag, 1980.

[10] A. Colmerauer. An Interesting Subset of Natural Language. In K. L. Clark and S.-A. Tärnlund, editors, *Logic Programming*, pages 45–66. Academic Press, New York, 1982. This article is a translation of "Un sous-ensemble intéressant du francais", R.A.I.R.O, vol. 13, No.4, 1979.

[11] R. Cooper. *Quantification and Syntactic Theory.* D. Reidel Publishing Company, 1983.

[12] F. Cuppens and R. Demolombe. Cooperative Answering: a Methodology to Provide Intelligent Access to Databases. In *Proc. of 2nd Int'l. Conf. on Expert Database Systems*, pages 333–353, 1988.

[13] V. Dahl. *Un système déductif d'interrogation de banques de données en espagnol.* PhD thesis, University of Aix-Marseille II, Marseille, France, 1977.

[14] V. Dahl. Translating Spanish into Logic through Logic. *American Journal of Computational Linguistics*, 7(3):149–164, 1981.

[15] V. Dahl. On Database Systems Development Through Logic. *ACM Transactions on Database Systems*, 7(1):102–123, 1982.

[16] V. Dahl. Incomplete Types for Logic Databases. *Applied Math. Letters*, 4(3):25–28, 1991.

[17] B. Dorner. Master's thesis, Simon Fraser University, Burnaby, Canada, in preparation.

[18] D. C. Fass, N. Cercone, G. Hall, C. Groeneboer, P. McFetridge, and F. Popowich. A Classification of User-System Interactions in Natural Language, with Special Reference to "Ill-Formed Input". In *Proceedings of the 5th Rocky Mountain Conference on Artificial Intelligence*, pages 143–148, 1990.

[19] D. C. Fass and G. Hall. A Belief-Based View of Ill-Formed Input. In N. Cercone, F. Gardin, and G. Valle, editors, *Computational Intelligence III (Proceedings of the International Symposium, Milan, Italy, September 24-28 1990)*, pages 143–148, 1991. Also as Technical Report CSS/LCCR TR 90-18, Centre for Systems Science, Simon Fraser University, Burnaby, BC, Canada.

[20] S. S. Fels and G. E. Hinton. Building Adaptive Interfaces with Neural Networks: The Glove-Talk Pilot Study. In D.Daiper, D.Gilmore, G.Cockton, and B.Shakel, editors, *Proceedings of the 3rd International Conference on Human-Computer Inteaction*, pages 683–688. North-Holland.

[21] S. Fisher. Influences on Word Order Change in American Sign Language. In C. N. Li, editor, *Word Order and Word Order Change*, pages 1–25. University of Texas Press, 1975.

[22] L. A. Friedman. Space, Time, and Person Reference in American Sign Language. *Language*, 51(4):940–961, 1975.

[23] L. A. Friedman. The Manifestation of Subject, Object, and Topic in the American Sign Language. In C. N. Li, editor, *Subject and Topic*, pages 125–148. Academic Press, 1976.

[24] T. Gaasterland, P. Godfrey, and J. Minker. Relaxation as a Platform of Cooperative Answering. In R. Demolombe, L. F. del Cerro, and T. Imielinski, editors, *Proc. of the 1st Int'l. Wshop. on Non-standard Queries and Answers, Volume 2*, pages 101–120, 1991.

[25] T. Gaasterland, P. Godfrey, and J. Minker. An Overview of Cooperative Answering. *J. of Intelligent Systems*, 1(2), 1992.

[26] T. Imielinski. Intelligent Query Answering in Rule Based Systems. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming.* Morgan.Kaufman Publishers, 1984.

[27] J. O. Isenhath. *The Linguistics of American Sign Language.* McFarland & Company, Inc., Publishers, 1990.

[28] J. M. Janas. On Feasibility of Informative Answers. In H. Gallaire and J. Minker and J.-M. Nicolas, editor, *Advances in Database Theory, Volume 1*, pages 397–414. Plenum Press, 1981.

[29] A. Joshi. Mutual Beliefs in Question Answering Systems. In N. Smith, editor, *Mutual Knowledge.* Academic Press, 1982.

[30] A. K. Joshi, B. Webber, and I. Sag. *Elemements of Discourse Understanding.* Cambridge University Press, 1981.

[31] A. K. Joshi, B. L. Webber, and R. M. Weischedel. Living up to Expectations: Computing Expert Responses. In *Proceedings of the Nat'l. Conf. on Artificial Intelligence*, pages 169–175. American Association for Artificial Intelligence, 1984.

[32] S. J. Kaplan. Appropriate Responses to Inappropriate Questions. In A. K. Joshi and B. Webber and I. Sag, editor, *Elemements of Discourse Understanding*, pages 127–144. Cambridge University Press, 1981.

[33] S. J. Kaplan. Cooperative Responses from a Portable Natural Language Query System. *Artificial Intelligence*, 19(2):165–187, 1982.

[34] E. S. Klima and U. Bellugi. *The Signs of Language.* Harvard University Press, 1979.

[35] W. Lehnert. A Computational Theory of Human Question Answering. In A. K. Joshi and B. Webber and I. Sag, editor, *Elemements of Discourse Understanding*, pages 145–176. Cambridge University Press, 1981.

[36] S. Liddell. *American Sign Language Syntax.* Mouton, 1980.

[37] C. Lucas, editor. *Sign Language Research: Theoretical Issues.* Gallaudet Press, 1990.

[38] G. F. Luger and W.A. Stubblefield. *Artificial Intelligence; Structures and Strategies for Complex Problem Solving.* Benjamin Cummings Publishing Company, Inc., second edition, 1993.

[39] E. Mays. Correcting Misconceptions about Database Structure. In *Proc. of CSCSI '80*, 1980.

[40] J. D. McCawley. *Everything that Linguists have Always Wanted to Know about Logic, but were ashamed to ask.* The University of Chicago Press, 1981.

[41] K. McCoy. Correcting Object-Related Misconceptions. In *Proc. of COLING10*, 1984.

[42] M. Moser. Incrementing Discourse with the Negation Relation. In Ch. Boitet, editor, *Proceedings of the 14th International Conference on Computational Linguitics, Vol. 1*, pages 317–323, 1992.

[43] A. Motro. Extending the Relational Model to Support Goal Queries. In *Proc. from the 1st Int'l. Wshop. on Expert Database Systems*, pages 129–150. The Benjamin/Cummings Publishing Company Inc., 1986.

[44] A. Motro. Using Constraints to Provide Intensional Answers to Relational Queries. In *Proc. of the 15th Int'l. Conf. on Very Large Data Bases*, 1989.

[45] F. C. N Pereira and S. M. Shieber. *Prolog and Natural-Language Analysis.* CSLI Lecture Notes, No. 10. Center for the Study of Language and Information (CSLI), Stanford University, 1987.

[46] F. C. N Pereira and D. H. D Warren. Definite Clause Grammars for Language Analysis - A survey of the Formalism and a Comparison with Transition Networks. *Artificial Intelligence*, 13:231–278, 1980.

[47] A. Pirotte and D. Roelants. Constraints for Improving the Generation of Intensional Answers in Deductive Databases. In *Proc. of the 5th Int'l. Conf. on Data Engineering*, 1989.

[48] A. Pirotte, D. Roelants, and E. Zimanyi. Controlled Generation of Intensional Answers. *IEEE Transactions on Knowledge and Data Engineering*, 1990.

[49] M. E. Pollack, J. Hirschberg, and B. Webber. User Participation in the Reasoning Process of Expert Systems. In *Proc. of the Nat'l. Conf. on Artificial Intelligence.* American Association of Artificial Intelligence, 1982.

[50] C. Pollard and I. A. Sag. *Information-Based Syntax and Semantics*, volume 1 of *CSLI Lecture Notes, No. 13*, chapter 1 and 2. Center for the study of language and information (CSLI), Stanford University, 1987.

[51] R. Reiter. On Closed World Databases. In J. Mylopoulos and M.L. Brodie, editors, *Readings in Artificial Intelligence and Databases*, pages 248–258. Morgan Kaufmann, 1989.

[52] D. Sahlin. The Mixtus Approach to Automatic Partial Evaluation of Full Prolog. In *Proceedings of the 1990 North American Conference on Logic Programming*, pages 377–398. MIT Press, 1990.

[53] P. Sgall, E. Hajičová, and J. Panevová. *The Meaning of the Sentence in Its Semantic and Pragmatic Aspects.* Academia: Prague and Reidel: Dordrecht, 1986.

[54] S. M. Shieber. *An Introduction to Unification-Based Approaches to Grammar.* CSLI Lecture Notes, No.4. Center for the study of language and information (CSLI), Stanford University, 1986.

[55] C. Shum and R. Muntz. Implicit Representation for Extensional Answers. In L. Kershberg, editor, *Expert Database Systems.* Tysons Corner, 1987.

[56] L. Sterling and E. Shapiro. *The Art of Prolog; Advanced Programming Techniques.* The MIT Press, 1986.

[57] W. C. Stokoe, D. Casterline, and C. Croneberg. *A Dictionary of American Sign Language on Linguistic Principles.* Linstok Press, 1976.

[58] W. Wahlster, H. Marburger, A. Jameson, and S. Busemann. Over-Answering Yes-No Questions: Extended Responses in a NL Interface to a Vision System. In *Proc. of IJCAI 1983*, 1983.

[59] R. B. Wilbur. *American Sign Language: Linguistic and Applied Dimensions.* College Hill Press, A division of Little, Brown and company (Inc.), second edition, 1987.

[60] J. Zimmer and C. Patschke. A Class of Determiners in ASL. In C. Lucas, editor, *Sign Language Research: Theoretical Issues*, pages 201–210. Gallaudet University Press, 1990. Also in: Linguistics of American Sign Language: a resource text for ASL, by C. Valli and C. Lucas, 1992, p.249–255.