EVOLUTIONARY ABSTRACTION OF SEMANTICS FOR DOMAIN SPECIFIC COMPUTING WITH WORDS

Paulus Nadi Wahjudi
University of Southern Mississippi

Follow this and additional works at: https://aquila.usm.edu/dissertations

Part of the Databases and Information Systems Commons

Recommended Citation
https://aquila.usm.edu/dissertations/1342

This Dissertation is brought to you for free and open access by The Aquila Digital Community. It has been accepted for inclusion in Dissertations by an authorized administrator of The Aquila Digital Community. For more information, please contact Joshua.Cromwell@usm.edu.
EVOLUTIONARY ABSTRACTION OF SEMANTICS
FOR DOMAIN SPECIFIC COMPUTING WITH WORDS

by

Paulus Nadi Wahjudi

Abstract of a Dissertation
Submitted to the Graduate Studies Office
of The University of Southern Mississippi
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

December 2007
ABSTRACT

EVOLUTIONARY ABSTRACTION OF SEMANTICS
FOR DOMAIN SPECIFIC COMPUTING WITH WORDS

by Paulus Nadi Wahjudi

December 2007

The World Wide Web contains numerous information that are scattered all over the world. Even though they are physically located in different geographical area, any persons with access to the Internet could have access to it at any given time from any locations. Along with the growth of Internet access, the amount of information on the World Wide Web will continue to grow. Therefore, it is no longer feasible to search the Internet for specific information without the help of a search engine. Currently, search engines utilize techniques that look for particular keywords on web pages rather than deducing an answer based on the question posed. The search engine would generate list of web pages containing the given keywords and forced users to manually sort to find the desired page. It is believed that search engines must incorporate natural language processing in order to become a question answer system that no longer list results but capable to give answer to a particular question posed by the user. In order to apply the above processing, computers must have Computing with Words (CW) capabilities where elements of computation are derived from natural language. Unfortunately, building machine knowledge of sentences and its meaning is a computationally intensive process with high turnaround time, thus making it unfeasible with current technology. It is proposed that to reduce the computational power and turnaround time needed to do such processing by incorporating genetic programming into the construction of the machine knowledge.
Genetic programming will help to reduce the amount of nodes in the Treebank to be processed in order to obtain a result. In addition, domain specific Treebank will be utilized to reduce the size of the sentence tree. Employing these methods, we plan to develop a tree that allow computers to process and build a knowledge base as a primary step to achieve computing with words.
TO MY MOTHER AND FATHER
ACKNOWLEDGEMENTS

I would like to thank you Dr. Dia Ali and the other committee members, Dr. Adel Ali, Dr. Shahram Rahimi, Dr. Khaled El-Sawi, Dr. Cliff Burgess and Dr. Ras Pandey for their advice and support throughout the process. I would especially like to thank my advisor, Dr. Dia Ali, for his generous time and commitment. Throughout my doctoral work he encouraged me to develop independent thinking and research skills. He continually stimulated my analytical thinking and greatly assisted me with scientific writing. I am extremely grateful for the assistance, generosity, and advice I received from Dr. Shahram Rahimi.

Nothing can express my gratitude to my family for their support and encouragements during my studies. My father extended his passion for learning to me while my mother was a constant source of support. I am grateful to my brother and sister for their encouragement and enthusiasm. My thanks to Wyatt and Joan Brumfield for their friendship and confidence in my study.
# TABLE OF CONTENTS

ABSTRACT ................................................................. ii  
DEDICATION .......................................................... iv  
ACKNOWLEDGEMENTS ............................................... v  
LIST OF ILLUSTRATIONS ........................................ viii  
LIST OF TABLES ....................................................... ix  

CHAPTER  

I. INTRODUCTION .................................................. 1  
   Question Answer System  
   Language Domain  

II. NATURAL LANGUAGE PROCESSING ....................... 7  
   Formal Grammar Rules  
   Language Treebank  

III. COMPUTING WITH WORDS ............................... 23  
   Generalized Constraint Language  
   Precisiated Natural Language  
   Prototypical Form  

IV. ABSTRACTION OF SEMANTICS ......................... 49  
   Constraint Driven Grammar  
   Domain Properties List  
   Sentence Tags  

V. DYNAMICALLY EVOLVING DECISION TREE .......... 59  
   Binary Decision Tree  
   Decision Tree Evolution  
   Testing and Analysis  

VI. CONCLUSIONS AND FUTURE WORK .................. 70  
   Summary of Work  
   Future Directions
APPENDIX ................................................................. 73
REFERENCES .......................................................... 76
LIST OF ILLUSTRATIONS

Figure

1.1 Traditional Web Search Engine ....................................................... 3
1.2 Question Answer System ............................................................... 5
2.1 Grammar Rules Example ............................................................. 10
2.2 Production Rules Example ........................................................... 12
2.3 Probabilistic Context Free Grammar Example .............................. 18
2.4 Treebank labeled brackets .......................................................... 20
2.5 Swedish Treebank example in XML .......................................... 21
4.1 Example of Properties List .......................................................... 52
4.2 Person Properties List ................................................................. 53
5.1 Example of an Attribute Relation File Format ............................. 66
5.2 Evolved Decision Tree ............................................................... 67
5.3 If-Then-Else form of the Decision Tree ..................................... 68
LIST OF TABLES

Table

4.1 Principal Modalities of Generalized Constraint ............... 53
CHAPTER I
INTRODUCTION

The amount of information on the Internet is rapidly increasing, meanwhile Human-Computer interaction are still done through the use of a “primitive” programming language that are incapable of complete natural language processing. With the abundant information on the World Wide Web (WWW), it is impossible to manually find specific information. This lends itself to the rise of web search engines such as Yahoo and Google that collects information on the Internet and organize it for data mining. Current search engine utilize methods that weigh a webpage relevance to particular keywords and provide the links to the webpage from the ones that are deemed most accurate to the least accurate. The user will have to manually search the list of information to find the information that he/she seeks. In an ideal condition the information will be available in the first or second page of results, if there are no correct results are found in the early pages then most likely the user must revise their search query and resubmit it. Most users have found out that the style “reduce query to keywords” is most effective and accurate.

However, this is not always the case for every query and every user, because in order to provide an accurate respond, the computer must analyze the question posed as a complete sentence rather than a collection of keywords and determine the answer to the question. Thus there is a need for natural language processing to be integrated into Internet search engines to improve user friendliness and accuracy of results.

Scientists have tried to tackle the problem of natural language processing and so far have been unsuccessful to provide adequate language processing system that will hold
an interactive conversation between human and machine. This is the primary drawback of natural language processing that does not go hand in hand with natural language processing search engines.

**Question Answer System**

The next stage in the transition from Computing with Numbers to Computing with Words is the question answer system where users will interact with computers in natural language. Users will pose a question in natural language and the computer will deduce the correct answer to it. As shown in figure 1, traditional search engine does not have deduction capability. Keywords from the search query are matched to a list of indexed documents and a resulting list will be compiled for the user. Currently, search engines are capable of natural language processing in a limited capacity. Google for example is capable in solving simple mathematical task such as addition, division, conversion, etc. Although this may be mistaken as deduction capability, it is actually a primitive natural language processing invoked when a search query conforms to a particular structure. Mathematical formulas are solved by parsing the query as an equation which is commonly done. Conversion sentence on the other hand is slightly different and contains two corresponding unit system and the keyword “in”. For example: the query “cm in inches” by default will give the length of 1 centimeter in inches. On the other hand, the query “cubic in feet” does not have a solution and Google will respond to it in the standard manner of documents list.
There are several problems that must be addressed in creating a question answer system. Each problem is closely related to each other and must be addressed in order to create a functional question answer system.

Language domain

A language domain determines the various sentences and meanings that can be built with the vocabulary associated with that domain. A person accumulates different domain in a lifetime and utilize them interchangeably as needed.

Although humans can subconsciously utilize different domain, it is different for computers. Each domain vocabulary tends to overlapped and contains different meaning for different domains, such complexity is hard to distinguish by computers. In certain case, a single domain could have identical sentence that have ambiguous meaning. This
idea reverts back to the individual’s world knowledge. World knowledge varies for each individual and affects how users perceive their knowledge and most importantly determine the creation of their query. A user perception is heavily influenced by their world knowledge. In order to assimilate world knowledge, the computer must possess the ability to process natural language domains and build a knowledge base. In our case, a question answer system must be able to process web documents and store it in a knowledge base.

Perception capability is defined as: “The computer must be able to accept a user question in natural language, perceive the question, consult with its knowledge base and give the answer in natural language.” Zadeh defined the fuzzy logic based precisiated natural language (PNL) as the tool to represent constraints. Figure 2 depicts the processing components in a question answer system. The natural language processing is in charge of processing the query into a format that can be processed by the perception and knowledge base component. The perception and knowledge base component is also in charge of acquiring the computer world knowledge. Due to the amount of computational power needed to process such knowledge, we must limit the system to process within a specific domain. Even in a specific domain, a Treebank holds an unprecedented amount of information that is far beyond the reach of current computational resources. Most of the scenarios used in Computing with Words (CW) publications incorporate such a narrow domain that it is only capable of answering a specific type of question.
Unfortunately, all of the researches done in CW have focused primarily on proving that it is feasible to generate an answer query given a valid question. Most of them have either omitted or skipped the explanation on how to acquire the machine world knowledge. In general, researches assume the existence of a relational database containing the individual information. For example, the question “What is the distance from Los Angeles to Miami?” can be answered if there exist a relational database table of the distance between major cities in the USA. This is not always the case, as most information does not transcribe naturally into tables. The correct way of building machine world knowledge is to process the Treebank. Some researchers have proposed different web semantics that will facilitate the creation of machine world knowledge along with a structured query. Genetic programming is an effective and efficient approach to replace the standard exhaustive search within a language domain. To
facilitate this, there is a need for an evolutionary Treebank algorithm that utilize genetic programming techniques in a domain specific Treebank. The power of genetic programming is the ability to do direct searches that are will cut the amount of tree needed. Shown in figure 4, crossover between two nodes at different level will result in a new sub-tree where the target leaf nodes exist.

The main drawback of natural language question answer system is the need of deduction capability. Zadeh’s vision of the question answer system and its supporting components of PNL is a promising look into the future of Computing with Words. The first step to creating Computing with Words is to build computer world knowledge. The world knowledge determined the ability of computers to reason and generates the answer. As described above, building the computer world knowledge is a computationally heavy task. The evolutionary Treebank algorithm will aid in the acquisition of the machine world knowledge. This important factor will provide computers the capability to process domain specific natural language that can be further processed into useable form such as knowledge base driven expert system.

The approach proposed is highly feasible and compatible with the objectives of Computing with Words and provide a much needed revelation to the “magic black-box” attitude that have plagued the advancements of CW in the field of computational intelligence. The hope is as that as new hardware provides stronger computational power, computing trend will also shift from computing with numbers to computing with words.
CHAPTER II

NATURAL LANGUAGE PROCESSING

Natural language processing covers two main studies, the study of semantics and syntax. Semantics refers to the aspects of meaning that are expressed in a language, code, or other form of representation while syntax is the study of the rules that govern the structure of sentences, and which determine their relative grammaticality.

Semantics is contrasted with two other aspects of meaningful expression, namely, syntax, the construction of complex signs from simpler signs, and pragmatics, the practical use of signs by agents or communities of interpretation in particular circumstances and contexts. By the usual convention that calls a study or a theory by the name of its subject matter, semantics may also denote the theoretical study of meaning in systems of signs. Though terminology varies, writers on the subject of meaning generally recognize two sorts of meaning that a significant expression may have the relation that a sign has to objects and objective situations, actual or possible, and the relation that a sign has to other signs, most especially the sorts of mental signs that are conceived of as concepts. Most theorists refer to the relation between a sign and its objects, as always including any manner of objective reference, as its denotation. Some theorists refer to the relation between a sign and the signs that serve in its practical interpretation as its connotation, but there are many more differences of opinion and distinctions of theory that are made in this case. Many theorists, especially in the formal semantic, pragmatic, and semiotic traditions, restrict the application of semantics to the denotative aspect, using other terms or completely ignoring the connotative aspect.
In linguistics, semantics is the subfield that is devoted to the study of meaning, as borne on the syntactic levels of words, phrases, sentences, and sometimes larger units of discourse, generically referred to as texts. As with any empirical science, semantics involves the interplay of concrete data with theoretical concepts. Traditionally, semantics has included the study of connotative sense and denotative reference, truth conditions, argument structure, thematic roles, discourse analysis, and the linkage of all of these to syntax. The decompositional perspective towards meaning holds that the meaning of words can be analyzed by defining meaning atoms or primitives, which establish a language of thought.

The term syntax can also be used to refer to these rules themselves, as in the syntax of a language. Modern research in syntax attempts to describe languages in terms of such rules, and, for many practitioners, to find general rules that apply to all languages. Since the field of syntax attempts to explain grammaticality judgments, and not provide them, it is unconcerned with linguistic prescription. Though all theories of syntax take human language as their object of study, there are some significant differences in outlook.

A set of concepts related to fuzziness in semantics is based on prototypes. Natural categories are not characterizable in terms of necessary and sufficient conditions, but are graded fuzzy at their boundaries and inconsistent as to the status of their constituent members. Systems of categories does not objectively exist in the world but are rooted in people's experience. These categories evolve as learned concepts of the world —meaning is not an objective truth, but a subjective construct, learned from experience, and language arises out of the grounding of our conceptual systems in shared embodiment.
and bodily experience. A corollary of this is that the conceptual categories will not be identical for different cultures, or indeed, for every individual in the same culture.

**Formal Grammar Rules**

There are two features shared by most theories of formal syntax. First, they hierarchically group subunits into constituent units. Second, they provide a system of rules to explain why certain utterances seem more acceptable or grammatical than others. Most formal theories of syntax also offer explanations of the systematic relationships between syntax and semantics, in other words, between form and meaning. There are various theories for designing the best grammars such that by systematic application of the rules, one can arrive at every phrase marker in a language and hence every sentence in the language. The most common are Phrase structure grammars and immediate dominance (ID) constraints from linear precedence (LP) constraints grammars, the latter of which some argue has an explanatory advantage. Dependency grammar is a class of syntactic theories separate from generative grammar in which structure is determined by the relation between a word and its dependents. One difference from phrase structure grammar is that dependency grammar does not have phrasal categories. Algebraic syntax is a type of dependency grammar. Tree-adjoining grammar is a grammar formalism with interesting mathematical properties which has sometimes been used as the basis for the syntactic description of natural language. In monotonic and monostratal frameworks, variants of unification grammar are often preferred formalisms.

A grammar is a description of the syntax of a language. Theoretical models rarely consider the language in use, as revealed by corpus linguistics, but focus on a mental
language. In contrast, the empirically responsible approach to syntax seeks to construct
grammars that will explain language in use. A key class of grammars in the latter
tradition is the stochastic context-free grammars.

In computer science and linguistics, a formal grammar is a precise description of a
formal language — that is, of a set of strings. The two main categories of formal
grammar are that of generative grammars, which are sets of rules for how strings in a
language can be generated, and that of analytic grammars, which are sets of rules for how
a string can be analyzed to determine whether it is a member of the language. In short, an
analytic grammar describes how to recognize when strings are members in the set,
whereas a generative grammar describes how to write only those strings in the set.

A generative grammar consists of a set of rules for transforming strings. To
generate a string in the language, one begins with a string consisting of only a single start
symbol, and then successively applies the rules (any number of times, in any order) to
rewrite this string. The language consists of all the strings that can be generated in this
manner. Any particular sequence of legal choices taken during this rewriting process
yields one particular string in the language. If there are multiple different ways of
generating a single string, then the grammar is said to be ambiguous. For example,
assume the alphabet consists of a and b, the start symbol is S and we have the following
rules shown in figure 2.1.

\[ R1. \ S \rightarrow aSb \]

\[ R2. \ S \rightarrow ba \]

Figure 2.1 Grammar rules example
Based on the rules in figure 2.1, a sentence start with $S$, and a rule can be apply to it. If we choose rule 1, we replace $S$ with $aSb$ and obtain the string $aSb$. If we choose rule 1 again, we replace $S$ with $aSb$ and obtain the string $aaSbb$. This process is repeated until we only have symbols from the alphabet ($a$ and $b$). If we now choose rule 2, we replace $S$ with $ba$ and obtain the string $aababb$, and are done. We can write this series of choices more briefly, using symbols: $S \rightarrow aSb \rightarrow aaaSbb \rightarrow aababb$. The language of the grammar is the set of all the strings that can be generated using this process: \{ba, abab, aababb, aaababbb, \ldots\}.

In the classic formalization of generative grammars a grammar $G$ consists of the following components: a finite set of terminal symbols, a finite set of nonterminal symbols, a finite set of production rules with a left- and a right-hand side consisting of a sequence of these symbols, a start symbol. A formal grammar defines (or generates) a formal language, which is a (possibly infinite) set of sequences of symbols that may be constructed by applying production rules to a sequence of symbols which initially contains just the start symbol. A rule may be applied to a sequence of symbols by replacing an occurrence of the symbols on the left-hand side of the rule with those that appear on the right-hand side. A sequence of rule applications is called a derivation. Such a grammar defines the formal language of all words consisting solely of terminal symbols that can be reached by a derivation from the start symbol. Nonterminals are usually represented by uppercase letters, terminals by lowercase letters, and the start symbol by $S$. For example, a production rules for the grammar with terminals \{a,b\}, nonterminals \{S,A,B\} is shown in figure 2.2.
$S \rightarrow \text{ABS}$

$S \rightarrow \varepsilon$ (e is the empty string)

$BA \rightarrow AB$

$BS \rightarrow b$

$Bb \rightarrow bb$

$Ab \rightarrow ab$

$Aa \rightarrow aa$

Figure 2.2 Production Rules example

Within the field of computer science, specifically in the area of programming languages, the Chomsky hierarchy is a containment hierarchy of classes of formal grammars that generate formal languages. Type-0 grammars (unrestricted grammars) include all formal grammars. They generate exactly all languages that can be recognized by a Turing machine. These languages are also known as the recursively enumerable languages. Note that this is different from the recursive languages which can be decided by an always-halting Turing machine. Type-1 grammars (context-sensitive grammars) generate the context-sensitive languages. These grammars have rules of the form $\alpha A\beta \rightarrow \alpha\gamma\beta$ with $A$ a nonterminal and $\alpha$, $\beta$ and $\gamma$ strings of terminals and nonterminals. The strings $\alpha$ and $\beta$ may be empty, but $\gamma$ must be nonempty. The rule $S \rightarrow \varepsilon$ is allowed if $S$ does not appear on the right side of any rule. The languages described by these grammars are exactly all languages that can be recognized by a linear bounded automaton (a nondeterministic Turing machine whose tape is bounded by a constant times the length of the input.) Type-2 grammars (context-free grammars) generate the context-free languages.
languages. These are defined by rules of the form $A \rightarrow \gamma$ with $A$ a nonterminal and $\gamma$ a string of terminals and nonterminals. These languages are exactly all languages that can be recognized by a non-deterministic pushdown automaton. Context free languages are the theoretical basis for the syntax of most programming languages. Type-3 grammars (regular grammars) generate the regular languages. Such a grammar restricts its rules to a single nonterminal on the left-hand side and a right-hand side consisting of a single terminal, possibly followed (or preceded, but not both in the same grammar) by a single nonterminal. The rule $S \rightarrow \varepsilon$ is also here allowed if $S$ does not appear on the right side of any rule. These languages are exactly all languages that can be decided by a finite state automaton. Additionally, this family of formal languages can be obtained by regular expressions. Regular languages are commonly used to define search patterns and the lexical structure of programming languages. Note that the set of grammars corresponding to recursive languages is not a member of this hierarchy.

Every regular language is context-free, every context-free language is context-sensitive and every context-sensitive language is recursive and every recursive language is recursively enumerable. These are all proper inclusions, meaning that there exist recursively enumerable languages which are not recursive, recursive languages that are not context-sensitive, context-sensitive languages which are not context-free and context-free languages which are not regular.

**Context Free and Context Sensitive Grammar**

A Context Free Grammar (CFG) is a formal grammar in which every production rule is of the form $V \rightarrow w$ where $V$ is a nonterminal symbol and $w$ is a string consisting of
terminals and/or non-terminals. The term "context-free" expresses the fact that the non-terminal $V$ can always be replaced by $w$, regardless of the context in which it occurs. A formal language is context-free if there is a context-free grammar that generates it.

Context-free grammars are powerful enough to describe the syntax of most programming languages; in fact, the syntax of most programming languages is specified using context-free grammars. On the other hand, context-free grammars are simple enough to allow the construction of efficient parsing algorithms which, for a given string, determine whether and how it can be generated from the grammar. An Earley parser is an example of such an algorithm, while a Left to Right – Rightmost derivation parser (LR) and Left to Right – Leftmost derivation parser (LL) only deal with more restrictive subsets of context-free grammars. Backus-Naur Form (BNF) is the most common notation used to express context-free grammars. However, not all formal languages are context-free.

A context-free grammar $G$ can be defined as:

$$G = (V_t, V_n, P, S)$$

$V_t$: A finite set of terminals

$V_n$: A finite set of non-terminals

$P$: A finite set of production rules

$S$: An element of $V_n$, the distinguished starting non-terminal

Elements of $P$ are of the form:

$$V_n \rightarrow (V_t \cup V_n)^* V_n \rightarrow (V_t \cup V_n)^*$$
A language $L$ is said to be a Context-Free-Language (CFL) if its grammar is Context-Free. More precisely, it is a language whose words, sentences and phrases are made of symbols and words from a Context-Free-Grammar. Usually, CFL is of the form $L=L(G)$. Although some operations on context-free grammars are decidable due to their limited power, CFGs do have interesting undecidable problems. One of the simplest and most cited is the problem of deciding whether a CFG accepts the language of all strings. A reduction can be demonstrated to this problem from the well-known undecidable problem of determining whether a Turing machine accepts a particular input. The reduction uses the concept of a computation history, a string describing an entire computation of a Turing machine. We can construct a CFG that generates all strings that are not accepting computation histories for a particular Turing machine on a particular input, and thus it will accept all strings only if the machine does not accept that input. As a consequence of this, it is also undecidable whether two CFGs describe the same language, since we cannot even decide whether a CFG is equivalent to the trivial CFG deciding the language of all strings.

A context-sensitive grammar (CSG) is a formal grammar in which the left-hand sides and right-hand sides of any production rules may be surrounded by a context of terminal and nonterminal symbols. Context-sensitive grammars are more general than context-free grammars but still regular enough to be parsed by a linear bounded automaton. The concept of context-sensitive grammar is a way to describe the syntax of natural language where it is indeed often the case that a word may or may not be appropriate in a certain place depending upon the context. A formal language that can be described by a context-sensitive grammar is called a context-sensitive language (CSL).
A formal grammar \( G = (N, \Sigma, P, S) \) is context-sensitive if all rules in \( P \) are of the form 
\[ \alpha A \beta \rightarrow \alpha \gamma \beta \]
where \( A \in N \) (\( A \) is a single nonterminal), \( \alpha, \beta \in (N \cup \Sigma)^* \) (\( \alpha \) and \( \beta \) are strings of nonterminals and terminals) and \( \gamma \in (N \cup \Sigma)^+ \) (\( \gamma \) is a nonempty string of nonterminals and terminals). In addition, a rule of the form \( S \rightarrow \epsilon \) provided \( S \) does not appear on the right side of any rule where \( \epsilon \) represents the empty string is permitted. The addition of the empty string allows the statement that the context sensitive languages are a proper superset of the context free languages, rather than having to make the weaker statement that all context free grammars with no \( \rightarrow \epsilon \) productions are also context sensitive grammars. The name context-sensitive is explained by the \( \alpha \) and \( \beta \) that form the context of \( A \) and determine whether \( A \) can be replaced with \( \gamma \) or not. This is different from a context-free grammar where the context of a non-terminal is not taken into consideration.

Another definition of context-sensitive grammars defines them as formal grammars where all productions are of the form \( \alpha \rightarrow \beta \) where \( |\alpha| \leq |\beta| \) where \( |\alpha| \) is the length of \( \alpha \). Such a grammar is also called a monotonic or non-contracting grammar because none of the rules decreases the size of the string that is being rewritten. If the possibility of adding the empty string to a language is added to the strings recognized by the non-contracting grammars (which can never include the empty string) then the languages in these two definitions are identical.

**Probabilistic Context Free Grammar**

A problem faced in any formal syntax is that often more than one production rule may apply to a structure, thus resulting in a conflict. The greater the coverage, the higher
this conflict, and all grammarians have spent considerable effort devising a prioritization for the rules, which usually turn out to be defeasible. Another difficulty is overgeneration, where unlicensed structures are also generated. Probabilistic grammars circumvent these problems by using the frequency of various productions to order them, resulting in a winner-take-all interpretation, which by definition, is defeasible given additional data. As usage patterns are altered in diachronic shifts, these probabilistic rules can be relearned, thus upgrading the grammar.

One may construct a probabilistic grammar from a traditional formal syntax by assigning each non-terminal a probability taken from some distribution, to be eventually estimated from usage data. On most samples of broad language, probabilistic grammars that tune these probabilities from data typically outperform hand-crafted grammars although some rule-based grammars are now approaching the accuracies of Probabilistic Context Free Grammar (PCFG). Recently, probabilistic grammars appear to have gained some cognitive plausibility. It is well known that there are degrees of difficulty in accessing different syntactic structures. Probabilistic versions of minimalist grammars have been used to compute information-theoretic entropy values which appear to correlate well with psycholinguistic data on understandability and production difficulty.

A stochastic context-free grammar (SCFG) also known as probabilistic context-free grammar (PCFG) is a context-free grammar in which each production is augmented with a probability. The probability of a derivation is the product of the probabilities of the productions used in that derivation, thus some derivations are more consistent with the stochastic grammar than others. PCFGs extend context-free grammars in the same way that hidden Markov models extend regular grammars. PCFGs have application in two
areas: Natural language processing and the study of RNA molecules within the field of Bioinformatics. PCFGs are a specialized form of weighted context-free grammars. Context-free grammars were originally conceived in an attempt to model natural languages, i.e. those normally spoken by humans. Some research has extended this idea with SCFGs.

$$0.7 \text{ VP } \rightarrow \text{ V NP}$$
$$0.3 \text{ VP } \rightarrow \text{ V NP NP}$$

Figure 2.3 Probabilistic Context Free Grammar example

Each rule is preceded by a probability that reflects the relative frequency with which it occurs as shown in figure 2.3. Given this grammar, we can now say that the number of NPs expected while deriving VPs is $0.7 \times 1 + 0.3 \times 2 = 1.3$. In particular, some speech recognition systems use PCFGs to improve their probability estimate and thereby their performance. Recently, PCFG's have played a role in explaining the Accessibility Hierarchy, which seeks to explain why certain structures are more difficult to understand than others, those with relative clauses like "they had forgotten that the box which Pat brought with apples in was lost". It turns out that if there is a probabilistic account of more likely constructions, then one can compute an information theoretic measure or entropy for the constructs. If the cognitive apparatus for syntax is based on information theoretic considerations, then it may very well employ something similar to PCFG.
Language Treebank

Treebank is a large and structured set of texts electronically stored and processed. They are used to do statistical analysis, checking occurrences or validating linguistic rules on specific universe. In order to make the corpora more useful for doing linguistic research, they are often subjected to a process known as annotation. Treebank utilize part-of-speech tagging along with additional information such as semantics. Treebanks can be created completely manually, where linguists annotate each sentence with syntactic structure, or semi-automatically, where a parser assigns some syntactic structure which linguists then check and, if necessary, correct.

The Treebank provides natural language processing (NLP) systems to evaluate and identify elements of a sentence and associate it as specific components in the natural language such as verbs, nouns, and adjective. The goal of NLP evaluation is to measure one or more qualities of an algorithm or a system, in order to determine if (or to what extent) the system answers the goals of its designers, or the needs of its users. Research in NLP evaluation has received considerable attention, since the definition of proper evaluation criteria is one way to precisely specify an NLP problem thus, going beyond the vagueness of tasks defined only as language understanding or language generation. A precise set of evaluation criteria, which includes mainly evaluation data and evaluation metrics, enables several teams to compare their solutions to a given NLP problem.

Treebank is a text corpus in which each sentence has been annotated with syntactic structure. Syntactic structure is commonly represented as a tree structure, hence the name treebank. Treebanks can be used in corpus linguistics for studying syntactic phenomena or in computational linguistics for training or testing parsers. Treebanks are often created
on top of a corpus that has already been annotated with part-of-speech tags. In turn, treebanks are sometimes enhanced with semantic or other linguistic information.

Treebanks can be created completely manually, where linguists annotate each sentence with syntactic structure, or semi-automatically, where a parser assigns some syntactic structure which linguists then check and, if necessary, correct. Some treebanks follow a specific linguistic theory in their syntactic annotation. However, two main groups can be distinguished: treebanks that annotate phrase structure (for example the Penn Treebank) and those that annotate dependency structure (for example the Prague Dependency Treebank). The syntactic structure in a treebank can be represented in many different ways, for example using simple labelled brackets in a text file or using Extensible Markup Language (XML) as shown in figure 2.4.

(S (NP (NNP John))
  (VP (VBZ loves)
    (NP (NNP Mary)))
  ( . ))

Figure 2.4 Treebank labeled brackets

There are numerous Treebank projects done in language other than English. For example, the Swedish language can be defined as the following: a treebank is a sequence of sentences. a sentence is a sequence of word. a word is an element with five attributes: id = Unique id within the sentence, form = Word form (string), pos = Part-of-speech tag,
**head** = Syntactic head (word id), **deprel** = Dependency relation to head. The resulting Treebank is shown in figure 2.5.

```xml
<sentence id="24">
  <word id="1" form="Dessutom" postag="ab" head="2" deprel="ADV"/>
  <word id="2" form="höjs" postag="vb.prs.sfo" head="0" deprel=""/>
  <word id="3" form="åldergränsen" postag="nn.utr.sin.def.nom" head="2" deprel="SUB"/>
  <word id="4" form="till" postag="pp" head="2" deprel="ADV"/>
  <word id="5" form="18" postag="rg.nom" head="6" deprel="DET"/>
  <word id="6" form="år" postag="nn.neu.plu.ind.nom" head="4" deprel="PR"/>
  <word id="7" form="." postag="mad" head="2" deprel="IP"/>
</sentence>
```

Figure 2.5 Swedish Treebank example in XML

**Natural Language Processing Evaluations**

Intrinsic evaluation considers an isolated NLP system and characterizes its performance mainly with respect to a gold standard result, pre-defined by the evaluators. Extrinsic evaluation, also called evaluation in use considers the NLP system in a more complex setting, either as an embedded system or serving a precise function for a human user. The extrinsic performance of the system is then characterized in terms of its utility with respect to the overall task of the complex system or the human user.

Black-box evaluation requires one to run an NLP system on a given data set and to measure a number of parameters related to the quality of the process (speed, reliability, resource consumption) and, most importantly, to the quality of the result (e.g. the accuracy of data annotation or the fidelity of a translation). Glass-box evaluation looks at the design of the system, the algorithms that are implemented, the linguistic resources it uses (e.g. vocabulary size), etc. Given the complexity of NLP problems, it is often difficult to predict performance only on the basis of glass-box evaluation, but this type of
evaluation is more informative with respect to error analysis or future developments of a system.

In many cases, automatic procedures can be defined to evaluate an NLP system by comparing its output with the gold standard (or desired) one. Although the cost of producing the gold standard can be quite high, automatic evaluation can be repeated as often as needed without much additional costs (on the same input data). However, for many NLP problems, the definition of a gold standard is a complex task, and can prove impossible when inter-annotator agreement is insufficient. Manual evaluation is performed by human judges, which are instructed to estimate the quality of a system, or most often of a sample of its output, based on a number of criteria. Although, thanks to their linguistic competence, human judges can be considered as the reference for a number of language processing tasks, there is also considerable variation across their ratings. This is why automatic evaluation is sometimes referred to as objective evaluation, while human one appears to be more subjective.
CHAPTER III
COMPUTING WITH WORDS

Zadeh [35] defined Computing with Words (CW) as a methodology to derive computational information such as facts, propositions and from natural language. CW is inspired by the human ability to perform both physical and mental activities with limited computations. The work on CW started in the creation of Fuzzy Logic to analyze vague concepts mathematically. However, Fuzzy Logics are more often used by engineers to represent imprecise concepts. Unfortunately, although most linguists have had little trouble accepting the idea of vague concepts, its impact on semantic theory has remained only of secondary interest to linguists.

After the early days of applying Fuzzy Logic to each and every conceivable problem that inspired the artificial intelligence community back in the 1970s, the field of Fuzzy Logic experienced some decline in popularity in the decades that followed. Fuzzy Logic matured to become the subject of major work on the foundations of mathematics, mainly carried out in eastern Europe. Most notable study in the field is the discovery of Fuzzy Logic as a generalization of classic logic that preserves its property of Hilbert completeness. It has also become an operative technology that enabled many remarkable technical achievements, celebrated mostly by Japanese engineers. Despite these successes, it may be fair to say that to this day, Fuzzy Logic has failed to live up to the high expectations artificial intelligence enthusiasts once had, that is to deploy a technology that can make machines understand the categories of reasoning that humans use to successfully communicate vague ideas and concepts.
Only recently, Zadeh took up renewed interest in this line of research, addressing the main shortcoming when he observes that progress has been, and continues to be, slow in those areas where a methodology is needed in which the objects of computation are perceptions – perceptions of time, distance, form, direction, color, shape, truth, likelihood, intent, and other attributes of physical and mental objects”.

The key point Zadeh has to make about perceptions is that they are inherently fuzzy, and that humans use natural language representations, instead of numeric measurements that is used by machines. Thus, the paradigm shift that takes Zadeh into his “new direction of artificial intelligence” is one that takes us “from computing with numbers to computing with words” [32]. The representations of fuzzy concepts employed in his computational theory of perceptions are linguistic in nature. They are expressions of a language he refers to as precisiated natural language [36]. Such a language would have to be natural, in the sense that it is a formal language, weakly equivalent to a subset of a natural language, and precisiated, in the sense that every such expression can automatically be translated to a form suitable for approximate reasoning.

There are two major imperatives for computing with words. First, computing with words is a necessity when the available information is too imprecise to justify the use of numbers; and second, when there is a tolerance for imprecision which can be exploited to achieve tractability, robustness, low solution cost and better rapport with reality. Exploitation of the tolerance for imprecision is an issue of central importance in CW and CTP. At this juncture, the computational theory of perceptions – which is based on CW – is in its initial stages of development. In time, it may come to play an important role in
the conception, design and utilization of information/intelligent systems. The role model for CW and CTP is the human mind.

In our quest for machines which have a high degree of machine intelligence (high MIQ), we are developing a better understanding of the fundamental importance in the remarkable human capacity to perform a wide variety of physical and mental tasks without any measurements and any computations. Underlying this remarkable capability is the brain’s crucial ability to manipulate perceptions – perceptions of distance, size, weight, force, color, numbers, likelihood, truth and other characteristics of physical and mental objects. A basic difference between perceptions and measurements is that, in general, measurements are crisp whereas perceptions are fuzzy. In a fundamental way this is the reason why in dealing with perceptions, it is necessary to employ a logical system that is fuzzy rather than crisp. Humans employ words to describe perceptions. This obvious observation is the point of departure for the theory outlined in the preceding sections. When perceptions are described in words, manipulation of perceptions is reduced to computing with words (CW). In CW, the objects of computation are words or, more generally, propositions drawn from a natural language. A basic premise in CW is that the meaning of a proposition, p, may be expressed as a generalized constraint in which the constrained variable and the constraining relation are, in general, implicit in p.

In coming years, computing with words and perceptions is likely to emerge as an important direction in science and technology. In a reversal of long-standing attitudes, manipulation of perceptions and words which describe them is destined to gain in respectability. This is a certain event in the future, as there are much to be gained by exploiting the tolerance for imprecision, uncertainty and partial truth in dealing with real-
world problems, and is the primary motivation for the methodology of computing with
words (CW) and the computational theory of perceptions (CTP).

Generalized Constraint Language

A generalized constraint is a constraint of the form \( X \) isr \( R \), where \( X \) is the
constrained variable, \( R \) is a constraining relation, generally non-bivalent, and \( r \) is an
indexing variable which identifies the modality of the constraint, that is, its semantics.
The principal constraints are: possibilistic (\( r = \text{blank} \)); probabilistic (\( r = p \)); veristic (\( r = v \));
usuality (\( r = u \)); random set (\( r = rs \)); fuzzy graph (\( r = fg \)); bimodal (\( r = bm \)); and group (\( r = g \)).
Generalized constraints may be qualified, combined and propagated. The set of all
generalized constraints together with rules governing qualification, combination and
propagation constitutes the Generalized Constraint Language (GCL).

Informally, a granule of a variable \( X \) is a clump of values of \( X \) which are drawn
together by definition, fuzziness, equivalence, similarity, proximity or functionality. For
example, an interval is a granule. So is a fuzzy interval. And so is a probability
distribution. Granulation is pervasive in human cognition. For example, the granules of
Age are fuzzy sets labeled young, middle-aged and old. The granules of Height may be
very short, short, medium, tall, and very tall. And the granules of Truth may be not true,
quite true, not very true, very true, etc. There are four basic rationales which underlie
granulation of attributes and the concomitant use of linguistic variables. First, the
inability of the human brain and senses to resolve the information in detail. For example,
looking at a person, we can determine their age bracket but unable to precisely determine
their age. Second, when numerical information may not be available. For example, we
may not know the exact population size a specific age group in a city but we can roughly determine it. Third, when an attribute is not quantifiable. For example, we describe degrees of reliability as: low, not high, high, very high, etc because we do not have a numerical scale. And fourth, when there is a tolerance for imprecision which can be exploited through granulation to achieve tractability, robustness and economy of communication. For example, it may be sufficient to know that there is a significant number of elderly in a particular city without getting the exact amount. What should be noted is that this is the principal rationale which underlies the extensive use of granulation, in the form of linguistic variables, in consumer products.

There is a close connection between granularity and uncertainty. Assume that $X$ is a variable and I am asked, "What is the value of $X$?" If my answer is "$X$ is $a$," where $a$ is a singleton, then there is no uncertainty in the information which I am providing about $X$. In this instance, information is singular. But if the answer is "$X$ is approximately $a$," or "$X$ is $\ast a$," for short, then there is some uncertainty in my answer. In this instance, information and its uncertainty will be described as granular. Closely, but not exactly, granularity may be equated to non-singularity. In the instance of "$X$ is $\ast a$," information is non-singular. A basic question which arises is: How can the meaning of $\ast a$ be preciated? In the context of standard probability theory, call it PT, $\ast a$ would normally be interpreted as a probability distribution centering on $a$.

Constraints are ubiquitous. A typical constraint is an expression of the form $X \in C$, where $X$ is the constrained variable, and $C$ is the set of values which $X$ is allowed to take. A typical constraint is hard (inelastic) in the sense that if $u$ is a value of $X$ then $u$ satisfies the constraint if and only if $u \in C$. The problem with hard constraints is that most real-
world constraints are not hard, that is, have some degree of elasticity. Real-world constraints may assume a variety of forms. They may be simple in appearance and yet have a complex structure.

There are many ways in which generalized constraints may be operated on. The basic operations expressed in symbolic form are the following:

Conjunction

\[
X \text{ is } R
\]
\[
Y \text{ is } S
\]

\[
(X, Y) \text{ ist } T
\]

Example (possibilistic constraints)

\[
X \text{ is } R
\]
\[
Y \text{ is } S
\]

\[
(X, Y) \text{ is } R \times S
\]

where \( \times \) is the Cartesian product.

Example (probabilistic/possibilistic)

\[
X \text{ isp } R
\]

\[
(X, Y) \text{ is } S
\]

\[
(X, Y) \text{ isrs } T
\]

In this example, if \( S \) is a fuzzy relation then \( T \) is a fuzzy random set. What is involved in this example is a conjunction of a probabilistic constraint and a possibilistic constraint. This type of probabilistic/possibilistic constraint plays a key role in the Dempster-Shafer theory of evidence, and in its extension to fuzzy sets and fuzzy probabilities (Zadeh [43]).
Example (possibilistic/probabilistic)

\[
X \text{ is } R
\]

\[
(X,Y) \text{ is } S
\]

\[
Y/X \text{ is } T
\]

This example, which is a dual of the proceeding example, is an instance of conditioning.

Projection (possibilistic)

\[
(X,Y) \text{ is } R
\]

\[
X \text{ is } S
\]

where \( X \) takes values in \( U=\{u\} \); \( Y \) takes values in \( V=\{v\} \); and the projection

\[
S=\text{Proj}_x R,
\]

is defined as

\[
\mu_S(v) = \mu_{R_{|x=R}}(v) = \max_u \mu_R(u,v),
\]

where \( \mu_R \) and \( \mu_S \) are the membership functions of \( R \) and \( S \), respectively.

Projection (probabilistic)

\[
(X,Y) \text{ is } R
\]

\[
X \text{ is } S
\]

where \( X \) and \( Y \) are real-valued random variables, and \( R \) and \( S \) are the probability distributions of \((X,Y)\) and \( X \), respectively. The probability density function of \( S, p_S \), is related to that of \( R, p_R \), by the familiar equation

\[
p_S(u) = \int p_R(u,v)dv
\]

with the integral taken over the real line.
Propagation

\[
\begin{align*}
\mathcal{f}(X) \text{ is } R \\
g(X) \text{ is } S
\end{align*}
\]

where \( f \) and \( g \) are functions or functionals.

Example (possibilistic constraints)

\[
\begin{align*}
\mathcal{f}(X) \text{ is } R \\
g(X) \text{ is } S
\end{align*}
\]

where \( R \) and \( S \) are fuzzy sets. In terms of the membership function of \( R \), the membership function of \( S \) is given by the solution of the variational problem

\[
\mu_S(v) = \sup_{\mu_R(f(u))}
\]

subject to

\[v = g(u).\]

Among the principal generalized constraints there are three that play the role of primary generalized constraints. They are:

- Possibilistic constraint: \( X \text{ is } R \)
- Probabilistic constraint: \( X \text{ is } \mathcal{P} R \)
- Veristic constraint: \( X \text{ is } \mathcal{V} R \)

A generalized constraint (GC), is composite if it can be generated from other generalized constraints through conjunction, and/or projection and/or constraint propagation and/or qualification and/or possibly other operations. For example, a random-set constraint may be viewed as a conjunction of a probabilistic constraint and either a possibilistic or veristic constraint. The Dempster-Shafer theory of evidence is, in effect, a theory of
possibilistic random-set constraints. The derivation graph of a composite constraint
defines how it can be derived from primary constraints.

The three primary constraints: possibilistic, probabilistic and veristic are closely
related to a concept which has a position of centrality in human cognition—the concept
of partiality. In the sense used here, partial means: a matter of degree or, more or less
equivalently, fuzzy. In this sense, almost all human concepts are partial (fuzzy). Familiar
examples of fuzzy concepts are: knowledge, understanding, friendship, love, beauty,
intelligence, belief, causality, relevance, honesty, mountain and, most important, truth,
likelihood and possibility. Is a specified concept, \( C \), fuzzy? A simple test is: If \( C \) can be
hedged, then it is fuzzy. For example, in the case of relevance, we can say: very relevant,
quite relevant, slightly relevant, etc. Consequently, relevance is a fuzzy concept.

The three primary constraints may be likened to the three primary colors: red,
blue and green. In terms of this analogy, existing themes of uncertainty may be viewed as
theories of different mixtures of primary constraints. For example, the Dempster-Shafer
theory of evidence is a theory of a mixture of probabilistic and possibilistic constraints.
The Generalized Theory of Uncertainty embraces all possible mixtures, and in this sense
the conceptual structure of GTU accommodates most, and perhaps all, of the existing
theories of uncertainty.

A concept which plays an important role in GTU is that of Generalized Constraint
Language (GCL). Informally, GCL is the set of all generalized constraints together with
the rules governing syntax, semantics and generation. Simple examples of elements of
GCL are:
$((X,Y) \text{ isp } A) \land (X \text{ is } B)$

$(X \text{ isp } A) \land ((X,Y) \text{ isv } B)$

$\text{Proj}_Y((X \text{ is } A) \land (X,Y) \text{ isp } B)$

where $\land$ is conjunction.

A very simple example of a semantic rule is:

$$(X \text{ is } A) \land (Y \text{ is } B) \rightarrow \text{Poss}(X=\mu_A, Y=\mu_B) = \mu_A(u) \land \mu_B(v),$$

where $u$ and $v$ are generic values of $X$, $Y$, and $\mu_A$ and $\mu_B$ are the membership functions of $A$ and $B$, respectively.

In principle, GCL is an infinite set. However, in most applications only a small subset of GCL is likely to be needed.

Precisiated Natural Language

How can precise meaning be assigned to a proposition, $p$, drawn from a natural language? The problem is that natural languages are intrinsically imprecise. Imprecision of natural languages is a consequence of the fact that (a) a natural language is, basically, a system for describing perceptions; and (b) perceptions are intrinsically imprecise as a consequence of (a) the bounded ability of sensory organs, and ultimately the brain, to resolve detail and store information; and (b) incompleteness of information. Given these facts, how can we precisiate the meaning $p$? A key idea which underlies the concept of Precisiated Natural Language (PNL), Zadeh [55] is to represent the meaning of $p$ as a generalized constraint, in symbols.

$$p \rightarrow X \text{ isr } R.$$
This idea is consistent with the fundamental premise of GTU, namely, that information is representable as a generalized constraint. The basis for the consistency is that a proposition, viewed as an answer to a question, is a carrier of information. In this sense, the premise “Information is representable as a generalized constraint,” is equivalent to the premise “A proposition is representable as a generalized constraint.” A forerunner of PNL is PRUF (Zadeh[48]).

Given that the Generalized Constraint Language, GCL, is the set of all generalized constraints, representing \( p \) as a generalized constraint is equivalent to translating \( p \) into an element, \( p^* \), of GCL. Thus, precisiation of a natural language, NL, may be viewed as translation of NL into GCL. Equivalently, translation of \( p \) into GCL may be viewed as explicitation of \( X, R \) and \( r \).

A proposition, \( p \), is precisiable if it is translatable into GCL. Not every proposition in NL is precisiable. But since GCL includes every possible constraint, it is more expressive in relation to NL than any existing synthetic language, among them the languages associated with first order logic, modal logic, Prolog and LISP.

Translation of \( p \) into GCL is made more transparent though annotation. To illustrate,

(a) \( p: \) Monika is young \( \rightarrow X/\text{Age(Monika)} \) is \( R/\text{young} \)

(b) \( p: \) It is likely that Monika is young \( \rightarrow \text{Prob}(X/\text{Age(Monika)} \) is \( R/\text{young}) \) is \( S/\text{likely} \)

Example b is an instance of probability qualification.
More concretely, let \( g(u) \) be the probability density function of the random variable, \( \text{Age(Monika)} \). Then, with reference to our earlier discussion of probability qualification, we have

\[
\text{Prob(\text{Age(Monika) is young}) is likely} \to \int_0^{\infty} g(u)\mu_{\text{young}}(u)\,du \text{ is likely}
\]
or, in annotated form,

\[
\text{GC}(g) = X / \int_0^{\infty} g(u)\mu_{\text{young}}(u)\,du \text{ is R/likely.}
\]

The test-score of this constraint on \( g \) is given by

\[
\text{ts}(g) = \mu_{\text{likely}} \left( \int_0^{\infty} g(u)\mu_{\text{young}}(u)\,du \right).
\]

(c) \( p \): Most Swedes are tall.

Following (b), let \( h(u) \) be the count density function of Swedes, meaning that \( h(u)\,du = \) fraction of Swedes whose height lies in the interval \([u, u+du]\). Assume that height of Swedes lies in the interval \([a, b]\). Then, fraction of tall Swedes: \( \int_a^b h(u)\mu_{\text{tall}}(u)\,du \) is most. Interpreting this relation as a generalized constraint on \( h \), the test-score may be expressed as

\[
\text{ts}(h) = \mu_{\text{most}} \left( \int_0^{b} h(u)\mu_{\text{tall}}(u)\,du \right).
\]

In summary, precisiation of “Most Swedes are tall” may be expressed as the generalized constraint.

Most Swedes are tall \( \rightarrow \) \( \text{GC}(h) = \mu_{\text{likely}} \left( \int_a^b h(u)\mu_{\text{tall}}(u)\,du \right) \).

An important application of the concept of precisiation relates to precisiation of propositions of the form “\( X \) is approximately \( a \),” where \( a \) is a real number. How can “approximately \( a \)”, or \( \ast a \) for short, be precisiated? In other words, how can the uncertainty associated with the value of \( X \) which is described as \( \ast a \), be defined precisely?
There is a hierarchy of ways in which this can be done. The simplest is to define \( *a \) as \( a \). This mode of precisiation will be referred to as singular precisiation, or \( s- \) precisiation, for short. \( s- \)-precisiation is employed very widely, especially in probabilistic computations, in which an imprecise probability, \( *a \), is computed with as if it were an exact number, \( a \).

The other ways will be referred to as granular precisiation, or \( g- \) precisiation, for short. In \( g- \) precisiation, \( *a \) is treated as a granule. What we see is that various modes of precisiating \( *a \) are instances of the generalized constraint. The concept of precisiation has an inverse—the concept of imprecisiation, which involves replacing \( a \) with \( *a \), with the understanding that \( *a \) is not unique. A basic problem which relates to imprecisiation is the following. Assume for simplicity that we have two linear equations involving real-valued coefficients and real-valued variables:

\[
\begin{align*}
    a_{11}X + a_{12}Y &= b_1 \\
    a_{21}X + a_{22}Y &= b_2.
\end{align*}
\]

Solutions of these equations read,

\[
\begin{align*}
    X &= \frac{a_{22}b_1 - a_{12}b_2}{a_{11}a_{22} - a_{12}a_{31}} \\
    Y &= \frac{a_{11}b_2 - a_{21}b_1}{a_{11}a_{22} - a_{12}a_{31}}.
\end{align*}
\]

Now suppose that we imprecisiate the coefficients, replacing, \( a_{ij} \) with \( *a_{ij} \), \( i, j = 1, 2 \), and replacing \( b_i \) with \( *b_i \), \( i = 1, 2 \). How can we solve these equations when imprecisiated coefficients are defined as generalized constraints?
There is no general answer to this question. Assuming that all coefficients are defined in the same way, the method of solution will depend on the modality of the constraint. For example, if the coefficients are interval-valued, the problem falls within the province of interval analysis (Moore [24]). If the coefficients are fuzzy-interval-valued, the problem falls within the province of the theory of relational equations (Di Nola et al [6, 7], Mares [23]). And if the coefficients are real-valued random variables, we are dealing with the problem of solution of stochastic equations. In general, solution of a system of equation with imprecisiated coefficients may present complex problems.

One complication is the following. If (a) we solve the original equations, as we have done above; (b) imprecisiate the coefficients in the solution; and (c) employ the extension principle to complete $X$ and $Y$, will we obtain solutions of imprecisiated equations? The answer, in general, is: No. Nevertheless, when we are faced with a problem which we do not know how to solve correctly, we proceed as if the answer is: Yes. This common practice may be described as Precisiation/Imprecisiation Principle which is defined in the following.

Precisiation/Imprecisiation Principle

Informally, let $f$ be a function or a functional. $Y=f(X)$, where $X$ and $Y$ are assumed to be imprecise, $Pr(X)$ and $Pr(Y)$ are precisiations of $X$ and $Y$, and $Pr(X)$ and $Pr(Y)$ are imprecisiations of $Pr(X)$ and $Pr(Y)$, respectively. In symbolic form, the P/I Principle may be expressed as:

$$f(X)\ast=f(Pr(X))$$
where \( * = \) denotes "approximately equal," and \(*f\) is imprecisiation of \( f \). In words, to compute \( f(X) \) when \( X \) is imprecise, (a) precisiate \( X \), (b) compute \( f(Pr(X)) \); and (c) imprecisiation \( f(Pr(X)) \). Then, usually, \(*f(Pr(X))\) will be approximately equal to \( f(X) \). An underlying assumption is that approximation are commensurate in the sense that the closer \( Pr(X) \) is to \( X \), the closer \( f(Pr(X)) \) is to \( f(X) \). This assumption is related to the concept of gradual rules of Dubois and Prade [9].

As an illustration, suppose that \( X \) is a real-valued function; \( f \) is the operation of differentiation, and \(*X\) is the fuzzy graph of \( X \). It should be underscored that imprecisiation is an imprecise concept. Use of the P/I Principle underlies many computations in science, engineering, economics and other fields. In particular, as was alluded to earlier, this applies to many computation in probability theory which involve imprecise probabilities. It should be emphasized that the P/I Principle is neither normative (prescriptive) nor precise; it merely describes imprecisely what is common practice—without suggesting that common practice is correct.

Prototypical Form

Informally, a protoform of an object is its abstracted summary. More specifically, a protoform is a symbolic expression which defines the deep semantic structure of an object such as a proposition, question, command, concept, scenario, case or a system of such objects. In the following, our attention which will be focused on protoforms of propositions, with \( PF(p) \) denoting a protoform of \( p \). Abstraction has levels, just as summarization does. For this reason, an object may have a multiplicity of protoforms. Conversely, many objects may have the same protoform. Such objects are said to be
protoform-equivalent, or PF-equivalent, for short. The set of protoforms of all precisiable propositions in NL, together with rules which govern propagation of generalized constraints, constitute what is called the Protoform Language (PFL).

Examples:

- Monika is young \( \rightarrow \) Age(Monika) is young \( \rightarrow \) \( A(B) \) is \( C \)
- Monika is much younger than Pat \( \rightarrow \) \( (A(B), A(C)) \) is \( R \)
- Distance between New York and Boston is about 200 mi \( \rightarrow \) \( A(B, C) \) is \( R \)
- Usually Robert returns from work at about 6pm \( \rightarrow \) Prob\( \{A \text{ is } B\} \) is \( C \)
- Carol lives in a small city near San Francisco \( \rightarrow \) \( A(B(C)) \) is \( (D \text{ and } E) \)
- Most Swedes are tall \( \rightarrow \) \( 1/n \ \Sigma \text{Count}(G[A] \text{ is } R) \) is \( Q \)

Protoformal deduction

The rules of deduction in GTU are, basically, the rules which govern constraint propagation. In GTU, such rules reside in the Deduction Database (DDB). The Deduction Database comprises a collection of agent-controlled modules and submodules, each of which contains rules drawn from various fields and various modalities of generalized constraints. A typical rule has a symbolic part, which is expressed in terms of protoforms; and a computational part which defines the computation that has to be carried out to arrive at a conclusion. In what follows, we describe briefly some of the basic rules, and list a number of other rules without describing their computational parts. The motivation for doing so is to point to the necessity of developing a set of rules which is much more complete than the few rules which are used as examples in this section.
Computational rule of inference

Symbolic part  Computational part

\( X \) is \( A \)
\( (X, Y) \) is \( B \)
\( Y \) is \( C \)

\( \mu_C(v) = \max_u (\mu_A(u) \land \mu_B(u, v)) \)

\( A, B \) and \( C \) are fuzzy sets with respective membership functions \( \mu_A, \mu_B, \mu_C \) \( \land \) is min or t-norm.

Intersection / product syllogism (Zadeh [46])

Symbolic part  Computational part

\( Q_1A's \) are \( B's \)
\( Q_2(A \text{and} B)'s \) are \( C's \)
\( Q_3A's \) are \( (B \text{and} C)'s \)

\( Q_1, Q_2 \) are fuzzy quantifiers; \( A, B, C \) are fuzzy sets; \( * \) is product in fuzzy arithmetic. [14]

Basic extension principle (Zadeh [39])

Symbolic part  Computational part

\( X \) is \( A \)
\( f(X) \) is \( B \)

\( \mu_B(v) = \sup_u (\mu_A(u)) \)

subject to
\( v = g(u) \)

\( g \) is a given function or functional; \( A \) and \( B \) are fuzzy sets.

The extension principle (Zadeh [55]) is the principal rule governing possibilistic constraint propagation.
The extension principle is a primary deduction rule in the sense that many other
deduction rules are derivable from the extension principle. An example is the following
rule.

**Basic probability rule**

<table>
<thead>
<tr>
<th>Symbolic part</th>
<th>Computational part</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(X) ) is ( A )</td>
<td>( \mu_B(v) = \sup_u (\mu_B(f(u))) )</td>
</tr>
<tr>
<td>( g(X) ) is ( B )</td>
<td>subject to ( v = g(u) )</td>
</tr>
</tbody>
</table>

The extension principle is a primary deduction rule in the sense that many other
deduction rules are derivable from the extension principle. An example is the following
rule.

**Basic probability rule**

<table>
<thead>
<tr>
<th>Symbolic part</th>
<th>Computational part</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Prob}(X \text{ is } A) ) is ( B )</td>
<td>( \mu_D(v) = \sup_u (\mu_D(\int_u \mu_A(u) g(u) du)) )</td>
</tr>
<tr>
<td>( \text{Prob}(X \text{ is } C) ) is ( D )</td>
<td>subject to ( v = \int_u \mu_C(u) r(u) du )</td>
</tr>
</tbody>
</table>

\( \int_u r(u) du = 1. \)

\( X \) is a real-valued random variable; \( A, B, C \) and \( D \) are fuzzy sets: \( r \) is the probability
density of \( X \); and \( U=\{u\} \). To derive this rule, we note that

\[
\text{Prob}(X \text{ is } A) \text{ is } B \quad \Rightarrow \quad \int_u r(u) \mu_A(u) du \text{ is } B
\]

\[
\text{Prob}(X \text{ is } C) \text{ is } D \quad \Rightarrow \quad \int_u r(u) \mu_C(u) du \text{ is } D
\]

which are generalized constraints of the form

\( f(r) \) is \( B \)

\( g(r) \) is \( D \).

Applying the extension principle to these expressions, we obtain the expression for \( D \)
which appears in the basic probability rule.
The bimodal interpolation rule is a rule which resides in the Probability module of DDB. The symbolic and computational parts of this rule are:

**Symbolic**

\[
\text{Prob}(X \text{ is } A_i) = P_i, \quad i=1, \ldots, n
\]

\[
\text{Prob}(X \text{ is } A) = Q
\]

**Computational**

\[
\mu_Q(v) = \sup_{\mu_A(\int u r(u)du) \wedge \mu_{A_n}(\int u r(u)du)} \mu_{A_1}(\int u r(u)du) \wedge \ldots \wedge \mu_{A_n}(\int u r(u)du)
\]

subject to

\[
v = \int u r(u)du
\]

\[
\int r(u)du = 1
\]

In this rule, \(X\) is a real-valued random variable; \(r\) is the probability density of \(X\); and \(U\) is the domain of \(X\). The probability rule is a special case of the bimodal interpolation rule.

What is the expected value, \(E(X)\), of a bimodal distribution? The answer follows through application of the extension principle:

\[
\mu_{E(X)}(v) = \sup_{\mu_A(\int u r(u)du) \wedge \mu_{A_n}(\int u r(u)du)} \mu_{A_1}(\int u r(u)du) \wedge \ldots \wedge \mu_{A_n}(\int u r(u)du)
\]

subject to

\[
v = \int u r(u)du
\]

\[
\int r(u)du = 1
\]

**Fuzzy-graph interpolation rule**
This rule is the most widely used rule in applications of fuzzy logic (Zadeh [51]).

We have a function, \( Y = f(X) \), which is represented as a fuzzy graph. The question is:

What is the value of \( Y \) when \( X \) is \( A \)? The \( A_i, B_i \) and \( A \) are fuzzy sets.

Symbolic part

\[
\begin{align*}
X \text{ is } A \\
Y = f(X) \\
f(X) \text{ isf} \sum_i A_i \times B_i \\
Y \text{ is } C
\end{align*}
\]

Computational part

\[
C = \sum_i m_i \wedge B_i,
\]

where \( m_i \) is the degree to which \( A \) matches \( A_i \)

\[
m_i = \sup_u (\mu_A(u) \wedge \mu_{A_i}(u)) , \quad i=1, \ldots, n.
\]

When \( A \) is a singleton, this rule reduces to

\[
\begin{align*}
X &= a \\
Y &= f(X) \\
f(X) \text{ isf} \sum_i A_i \times B_i , \quad i=1, \ldots, n. \\
Y &= \sum_i \mu_{A_i}(a) \wedge B;
\end{align*}
\]

In this form, the fuzzy-graph interpolation rule coincides with the Mamdani rule—a rule which is widely used in control and related applications. (Mamdani and Assilian [22]). In the foregoing, we have summarized some of the basic rules in DDB which govern generalized constraint propagation. Many more rules will have to be developed and added to DDB. A few examples of such rules are the following.

Probabilistic extension principle
\[ f(X) \text{ is } A \]
\[ g(X) \text{ is } ?B \]

Usuality-qualified extension principle

\[ f(X) \text{ is } A \]
\[ g(X) \text{ is } ?B \]

Usuality-qualified fuzzy-graph interpolation rule

\[ X \text{ is } A \]
\[ Y = f(X) \]

\[ f(X) \text{ isf} \sum_i \text{ if } X \text{ is } A_i \text{ then } Y \text{ is } B_i \]

\[ Y \text{ is } ?B \]

Bimodal extension principle

\[ X \text{ isbm} \sum_i P_i \setminus A_i \]
\[ Y = f(X) \]

\[ Y \text{ is } ?B \]

Bimodal, binary extension principle

\[ X \text{ is } R \]
\[ Y \text{ is } S \]

\[ Z = f(X,Y) \]

\[ Z \text{ is } T \]

In the instance, bimodality means that \( X \) and \( Y \) have different modalities, and binary means that \( f \) is a function of two variables. An interesting special case is one in which \( X \) is \( R \) and \( Y \) is \( S \). The deduction rules which were briefly described in the foregoing are intended to serve as examples. How can these rules be applied to reasoning.
under uncertainty? To illustrate, it will be convenient to return to the examples given in section 1.

$p$: Usually Robert returns from work at about 6:00 pm. What is the probability that Robert is home at 6:15 pm?

First, we find the protoforms of the data and the query.

Usually Robert returns from work at about 6:00 pm

\[ \text{Prob}(\text{Time(Return(Robert)) is *6:00 pm}) \text{ is usually} \]

which in annotated form reads

\[ \text{Prob}(X/\text{Time(Return(Robert)) is A/*6:00pm}) \text{ is B/usually} \]

Likewise, for the query, we have

\[ \text{Prob}(\text{Time(Return(Robert)) is } \leq \circ \, *6:15pm}) \text{ is ? D} \]

which in annotated form reads

\[ \text{Prob}(X/\text{Time(Return(Robert)) is } C/ \leq \circ \, *6:15pm}) \text{ is D/usually} \]

Searching the Deduction Database, we find that the basic probability rule matches the protoforms of the data and the query

\[ \text{Prob}(X \text{ is } A) \text{ is } B \]
\[ \text{Prob}(X \text{ is } C) \text{ is } D \]

where

\[ \mu_D(v) = \sup_{\mu_B}(\mu_A(\int u \mu_A(u)g(u)du)) \]

subject to

\[ v = \int u \mu_C(u)g(u)du \]
\[ \int g(u)du = 1 \]
Instantiating $A$, $B$, $C$ and $D$, we obtain the answer to the query:

Probability that Robert is home at about 6:15pm is $D$,

where

$$\mu_D(v) = \sup_g \left( \mu_{\text{usually}} \left( \int_{6.00 \, \text{pm}} g(u) \, du \right) \right)$$

subject to

$$v = \int_{6.15 \, \text{pm}} g(u) \, du$$

and

$$\int g(u) \, du = 1$$

For the tall Swedes problem, we start with the data

$p$: Most Swedes are tall.

 Assumes that the queries are:

$q_1$: How many Swedes are not tall

$q_2$: How many are short

$q_3$: What is the average height of Swedes

In our earlier discussion of this example, we found that $p$ translates into a generalized constraint on the count density function, $h$.

Thus

$$p \rightarrow \int_a^b h(u) \, du \mu_{\text{tall}}(u) \, du \text{ is most}$$

Precisations of $q_1$, $q_2$ and $q_3$ may be expressed as

$q_1$: $$\int_a^b h(u) \, du \mu_{\text{not\,tall}}(u) \, du$$

$q_2$: $$\int_a^b h(u) \, du \mu_{\text{short}}(u) \, du$$

$q_3$: $$\int_0^b h(u) \, du$$
Considering $q_1$, we note that

$$\mu_{\text{not tall}}(u) = 1 - \mu_{\text{tall}}(u).$$

Consequently

$$q_1 \rightarrow 1 - \int_a^b h(u) \mu_{\text{tall}}(u) du$$

which may be rewritten as

$$q_2 \rightarrow 1\text{-most}$$

where 1-most plays the role of the antonym of most.

Considering $q_2$, we have to compute

$$A: \int_a^b h(u) \mu_{\text{short}}(u) du$$

given that $\int_a^b h(u) \mu_{\text{tall}}(u) du$ is most

Applying the extension principle, we arrive at the desired answer to the query:

$$\mu_A(v) = \sup(\mu_{\text{most}}(\int_a^b h(u) \mu_{\text{tall}}(u) du))$$

subject to

$$v = \int_a^b h(u) \mu_{\text{short}}(u) du$$

and

$$\int_a^b h(u) du = 1.$$

Likewise, for $q_3$ we have as the answer

$$\mu_A(v) = \sup(\mu_{\text{most}}(\int_a^b h(u) \mu_{\text{tall}}(u) du))$$

subject to

$$v = \int_a^b uh(u) du$$

and

$$\int_a^b h(u) du = 1.$$
As an illustration of application of protoformal deduction to an instance of this example, consider

\( p: \) Most Swedes are tall
\( q: \) How many Swedes are short?

We start with the protoforms of \( p \) and \( q \) (see earlier example):

- Most Swedes are tall \( \rightarrow \) \( \frac{1}{n} \sum \text{Count}(G[A \text{ is } R]) \) is \( Q \)
- ?T Swedes are short \( \rightarrow \) \( \frac{1}{n} \sum \text{Count}(G[A \text{ is } S]) \) is \( T \)

where

\( G[A] = \sum_i \text{Name}_i / A_i, \quad i=1, \ldots, n. \)

An applicable deduction rule in symbolic form is:

\[ \frac{1}{n} \sum \text{Count}(G[A \text{ is } R]) \] is \( Q \)
\[ \frac{1}{n} \sum \text{Count}(G[A \text{ is } S]) \] is \( T \)

The computational part of the rule is expressed as

\[ \frac{1}{n} \sum_i \mu_R(A_i) \] is \( Q \)
\[ \frac{1}{n} \sum_i \mu_S(A_i) \] is \( T \)

where

\[ \mu_T(v) = \sup_{A_i, \ldots, A_n} \mu_Q(\sum_i \mu_R(A_i)) \]

subject to

\[ v = \sum_i \mu_S(A_i) \]

What we see is that computation of the answer to the query, \( q \), reduces to the solution of a variational problem, as it does in the earlier discussion of this example in which protoformal deduction was not employed.
An example on Vera’s age problem. Vera has a son who is in mid-twenties, and a daughter, who is in mid-thirties. What is Vera’s age?

In dealing with this problem, we will proceed to solution directly, bypassing protoformal deduction. Precisations of the query and given information may be expressed as

\[ q: \text{What is Vera’s age?} \rightarrow \text{Age}(\text{Vera}) \text{ is ?A} \]

\[ P_1: \text{Vera has a son who is in mid-twenties} \rightarrow \text{Age}(\text{Son}(\text{Vera})) \text{ is *20.} \]

\[ P_2: \text{Vera has a daughter who is in mid-thirties} \rightarrow \text{Age}(\text{Daughter}(\text{Vera})) \text{ is *30.} \]

Let \( X \) be Vera’s age when her son was born, and let \( Y \) be Vera’s age when her daughter was born. From World Knowledge Database, we draw the information

\[ \text{wk}_1: \text{Child-bearing age ranges from *16 to *42.} \]

\[ \text{wk}_2: \text{Age of mother is the sum of the age of child and the age of mother when the child was born.} \]

Combining the given information with that drawn from the World Knowledge Database, we led to an estimate of Vera’s age which may be expressed as

\[ \text{Age}(\text{Vera}) \text{ is } ((*25+[*16, *42]) \land (*35+[*16, *42])) \]

The point of this example is that it underscores that, in general, computation of an estimate depends on the interpretation of “approximately \( a \),” when \( a \) is a real number. In particular, computation of Vera’s age is straightforward if \( *a \) is interpreted as a possibility distribution. It is less straightforward when \( a \) is interpreted as a probability distribution. And it is much less straightforward when \( *25, \) for example, is interpreted as a possibility distribution, and \( [*16, *42] \) is interpreted as a probability distribution or, more realistically, as a bimodal distribution.
CHAPTER IV
ABSTRACTION OF SEMANTICS

The aim for abstraction of semantics is to decipher a proposition in natural language containing implicit fuzzy constraints into a canonical form with explicit fuzzy constraints. This is crucial to derive propositions in natural language for both Initial Data Set (IDS) and Terminal Data Set (TDS). The key in understanding natural language is by understanding the context. However, Chomsky's approach in Context Sensitive Grammar (CSG) is incapable to reason with natural languages because it is working at a much finer granularity while the human mind construct words at a coarser granularity. Many perceive that Context Sensitive Language (CSL) is capable in defining Natural Language (NL) and that the difficulty in automating NL processing is the lack of formal definition. In short, CSG is defined at a word level while human perception of language is in a sentence level. For example, we cannot derive the context of a conversation given only three words at a time.

Which drives the question on “How to derive the semantic meaning of a sentence before determining its context?” Zadeh’s approach in this is to convert natural language proposition into a generalized constraint language. A generalized constraint is a constraint of the form $X$ isr $R$, where $X$ is the constrained variable, $R$ is a constraining relation, generally non-bivalent, and $r$ is an indexing variable which identifies the modality of the constraint, that is, its semantics. The principal constraints are: possibilistic ($r=$blank), probabilistic ($r=p$), veristic ($r=v$), subsethood constraint ($r=c$), usuality ($r=u$); random set ($r=rs$), fuzzy graph ($r=fg$), bimodal ($r=bm$) and group ($r=g$).
Generalized constraints may be qualified, combined and propagated. However, there has not been any significant work done in processing sentences into its principal constraints.

One of the key problems when using CwW is translating perceptions given in natural language into a standard format that can be manipulated with computation. Therefore, if a set of documents is available in natural language, it is necessary to translate these documents into a standard format before we can perform any deduction based on the implicit perceptions in these documents. A context free grammar (CFG) describes a language by providing a set of production rules which govern how a non-terminal symbol of the language can be expanded into a set of terminal and non-terminal symbols. The CFG forms part of the phase structure (PS) grammars that were introduced by N. Chomsky in 1957 when he applied Post production rules to natural languages.

Due to the complexity and difference between languages, each grammar must be associated with a principal constraint. As discussed above, grammars alone are incapable to perceive the meaning of a sentence. However, a grammar can be built that are only capable of constructing sentence within the principal constraint and failed on different constraint. A grammar contains two components, structural and terminal rules. Structural rules define the components of the sentence.

To be effective, each grammar must be deterministic and perceivable, a grammar must be associated with a particular constraint. The changes must be done in the both terminal rules and structural rules. In terminal rules we limit the type of terminal words that can be built by the grammar. In structural rules, we will remove phrases that do not conform to the principal constraints within the Adjective Phrases (AP), Noun Phrases (NP) and Determiner Phrases (DP). However, this approach has its drawbacks where it
spawns additional grammars from a comprehensive grammar that can process different constraints but the resulting grammars are small and accurate in determining the meaning of the sentence that it outweighs the processing overhead.

We will analyze each constraint as shown in table 1 and determine the key factors in natural language that are commonly used to build such sentences. The constraints are represented in an $X \text{ is } r R$ form where "r" represents the different type of constraints. Each constraint address a particular concept commonly represented in natural language.

<table>
<thead>
<tr>
<th>$X \text{ is } r R$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$X \text{ is } R$</td>
<td>Possibilistic</td>
</tr>
<tr>
<td>$X \text{ isp } R$</td>
<td>Probabilistic</td>
</tr>
<tr>
<td>$X \text{ isv } R$</td>
<td>Veristic</td>
</tr>
<tr>
<td>$X \text{ isu } R$</td>
<td>Usuality</td>
</tr>
<tr>
<td>$X \text{ isrs } R$</td>
<td>Random Set</td>
</tr>
<tr>
<td>$X \text{ isfg } R$</td>
<td>Fuzzy Graph</td>
</tr>
<tr>
<td>$X \text{ isbm } R$</td>
<td>Bimodal</td>
</tr>
<tr>
<td>$X \text{ isg } R$</td>
<td>Group</td>
</tr>
</tbody>
</table>

Table 4.1 Principal modalities of generalized constraints

**Constraint Driven Grammar**

We propose the creation of a Constraint Defined Grammar (CDG) that augments any existing grammar (CFG, CSG, PCFG) within a specific domain. CDG allows the ability to determine the components of each sentence made by that grammar according to computing with words. CDG contains two primary components, Properties List (PL) and Sentence Tags (ST). PL describes the components within a domain that can be either a
subject or object in a sentence along with information and values associated with it. For example, within a domain that describes physical characteristics of a person there are several properties described in it as shown in Figure 4.1.

Person → Name, Body, Arms, Legs, Head, Height, Weight, Color

Body → Size, Shape

Arms → Fingers, Elbow

Legs → Ankle, Thighs, Knee

Head → Eyes, Ears, Mouth, Nose, Hair, Facial Hair

Fingers → Thumb, Index, Middle, Ring, Little

Hair → Color, Length

Facial Hair → Moustache, Beard

Figure 4.1 Example of properties list

In a different domain, a person can have an entirely different set of properties list depending on the type of information and values that can be extracted about a person for that particular domain. For example, in a domain of transportation and traveling such as the Robert example in chapter III, a person properties list is shown in figure 4.2. It significantly differs from figure 4.1 as both domains require different information from the component of a person. This approach allows the ability to avoid confusion in answering an important question: Is it the value of Robert in departure time or is it the value of departure time in Robert? Such question are simple for the human mind, however the computer is unaware of the difference thus it requires a definition and
directions on where to extract the necessary values. In general, any nouns in a grammar will have a properties list associated with it to aid in the defining the components of computing with words.

Person → Departure Time, Arrival Time, Travel Time

Departure Time → Year, Month, Days, Hours, Minutes, Seconds
Arrival Time → Year, Month, Days, Hours, Minutes, Seconds
Travel Time → Years, Months, Days, Hours, Minutes, Seconds

Figure 4.2. Person Properties List

Domain Properties List

An important factor that must be addressed in computing with words is the ability to determine the quality of a domain. More often than none, a domain is considered complete when there is a grammar that can parse sentences within that domain. However, the quality of information within that domain may be inadequate to support computing with words and a question answer system. We have identified six types of information properties that a domain must contain to facilitate a question answer system: Discrete, Relative, Uncertainty, Sequential, Group, Sequential and Query information properties. These information properties represent the different type of information that a human being associate with in natural language.

The first information property is Distinct Information (DI), this property involves unique and discrete information that represent particular information and can represented with a value. For example, the distinct information property of a geographical domain
consists of coordinate information such as latitude and longitude, elevation, and climate. The distinct information property is crucial to provide an accurate respond within a question answer system. In a different domain that covers traveling information, although this domain may share some DI properties it also have additional DIs that might not be available in geographical information such as address information, street name, apartment number, zipcode and additional information pertinent to traveling such as road condition, traffic report and even weather forecast. The second information property is Relative Information (RI), this property covers information that can be represented with different values of similar type within a boundary. For a geographical domain, tall and big are considered relative information. What constitutes tall in a geographical domain? Any natural and manmade landmarks that are over a certain height defined by the domain expert can be categorized as tall and any area spanning over a certain distance as defined can be consider as big. In a traveling domain, RI involves information such as fast or slow and short or long. Any traveling time within specific time period can be considered as fast while anything over that time period is considered slow, additionally any stops made during the travel can be considered as short stop or long stop depending on the amount of time spent.

Discrete and Relative information covers information that can be processed computationally. The next three property deals with a more complex type of information that is often used in natural language but harder to process computationally if not identified. Uncertainty Information (UI) property refers to general assumptions made within the domain that a human being perceives naturally. In geographical domain, we assume that most large body of water can be defined as a sea, this is correct in most cases
but not in all situations. Similarly, while traveling we have discussed that a travel time is considered fast if it is achieved within a period of time, however we also often define very fast travel or very slow travel which are derived from slow/fast travel times. Most often, words associated with UI can be associated with certainty values that modify existing information property. The Group Information (GI) property refers to a collection of information that combines as one. The information is created by combining different type of information properties. For example in a geographical domain a user might request information regarding a country. This will require a collection of answers to be presented to the user. In a traveling domain however, a query for directions will require the utilization of a Sequential Information (SI) where a collection of information must be presented in particular sequence to give accurate turn-by-turn directions from point A to point B.

The Query Information (QI) property provides the different type of query that can be used to extract information from the domain. We followed the basic Who, What, Where, When, Why and How type of question with the addition of a Boolean type question for Yes/No answers. Who, Where and When primarily requires concrete type information as a response, while Why and How will require either a GI or SI type information. What is the most complex query as it answers can be in any type of information property.

A domain expert will be able to identify the components of each information property and determine the quality of the domain based on it domain properties list. A domain that predominantly consist of one or two information properties are more compatible for computational processing however it lacks the ability to represent the
variety within a complex natural language. A good domain will have an even distribution of different information properties.

Sentence Tags

Sentence Tags are any components of a sentence within the domain and grammar that can be used to identify a sentence’s relation to the different components of computing with words. There are three types of tags: Keyword Tag (KT), Element Tag (ET), Context Tag (CT). A Keyword Tag is any word that can be used to identify or perceive the content of the sentence within the context of computing with words. Some examples of a keyword tag are the word “not” that indicates negation, “half” that indicates partiality and “somewhat” that indicates fuzziness. An Element Tag is certain affix that indicates a particular type of sentence. Some sentence utilize affix to replace particular word and such approach allows faster identification and processing of sentence in computing with words. Affixes encompass both prefix such as “anti” that indicates negation and “hyper” that indicates very or exaggeration and suffixes such as “est” in tallest, smartest indicating the most and “er” in taller, smarter that indicates better than. A Context Tag is a combination of keywords and affixes that determine the sentence constraint. For example, “most of the time” and “every now and then” indicates fuzzy frequency while “smarter than” and “faster than” indicates comparison between similar values.

Each Sentence Tag is associated with a role. There are three role defined from the $X isr R$ constraint: Left Hand Side Role (LHSR), Constraint Role (CR) and Right Hand Side Role (RHSR). LHSR represents the $X$ component of the constraint within a sentence.
as a random variable, fuzzy variable or other type of variable that exist in the domain, while the RHSR represents the R component such as probability distribution and fuzzy graph defined within the domain.

The Constraint Role is crucial as it serve as the component that identifies the sentence association to a particular constraint. Each constraint contains several different type of CR within the domain that identify any given sentence association with that particular constraint.

Tags can now be defined by its type and role for example a Keyword Tag with Left Hand Side Role (KT-LHSR) in the simple sentence “Robert is smart” is Robert, then we can further process that Robert is property of a person.name().

We will analyze the three primary constraints and the definition of the sentence tags associated with it. In the possibilistic constraint of \( X \ is \ R \), a CR includes the possibility of \( X \) is equal (EQ) or not equal (NEQ) to \( R \). In the example of physical characteristics, the sentence “Robert is short” and “Robert is not short” contains the LHSR of Robert which is a property of a person.name() = robert and the RHSR of short which is a property of person.height() = short or not short. The probabilistic constraint of \( X \ isp \ R \), a CR defines the LHSR as a part of certainty factor value of RHSR. The sentence “It is very likely that Robert is short” contains the CR very likely that indicates a high probability that person.height() = short for person.name() = Robert. Within the veristic constraint of \( X \ isv \ R \), a CR describe a truth distribution of the LHSR. For example, in the same example of physical characteristic the sentence “Robert’s hair is golden brown” meaning that the value for person.hair.color() = 0.8|golden + 0.2|brown IFF person.name() = Robert.
Currently, the work done on computing with words achieve the processing of such information by indicating that the information will be provided by an expert or readily available in some type of database or knowledge base that can be queried to extract the necessary information. However, to achieve a complete computing with words may define but the domain at first but further processing must be automated to ensure rapid processing as such task are tedious and time consuming to be done manually. The approach of CDG will allow experts to define the components necessary for computers to process a sentence in a given domain and associate parts and its role in the computing with words constraints.
CHAPTER V

DYNAMICALLY EVOLVING DECISION TREE

Binary Decision Tree

In operations research, specifically in decision analysis, a decision tree is a decision support tool that uses a graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. A decision tree is used to identify the strategy most likely to reach a goal. Another use of trees is as a descriptive means for calculating conditional probabilities. In data mining and machine learning, a decision tree is a predictive model; that is, a mapping from observations about an item to conclusions about its target value. More descriptive names for such tree models are classification tree (discrete outcome) or regression tree (continuous outcome). In these tree structures, leaves represent classifications and branches represent conjunctions of features that lead to those classifications. The machine learning technique for inducing a decision tree from data is called decision tree learning, or (colloquially) decision trees. In decision analysis, a decision tree and a closely related model form, an influence diagram is used as a visual and analytical decision support tool, where the expected values (or expected utility) of competing alternatives are calculated.

A binary decision diagram (BDD), like a negation normal form (NNF) or a propositional directed acyclic graph (PDAG), is a data structure that is used to represent a Boolean function. A Boolean function can be represented as a rooted, directed, acyclic graph, which consists of decision nodes and two terminal nodes called 0-terminal and 1-terminal. Each decision node is labeled by a Boolean variable and has two child nodes.
called low child and high child. The edge from a node to a low (high) child represents an assignment of the variable to 0 (1). Such a BDD is called 'ordered' if different variables appear in the same order on all paths from the root. It is called 'reduced' if the graph is reduced according to two rules:

- Merge any isomorphic subgraphs.
- Eliminate any node whose two children are isomorphic.

In popular usage, the term BDD almost always refers to Reduced Ordered Binary Decision Diagram (ROBDD in the literature, used when the ordering and reduction aspects need to be emphasized). The advantage of an ROBDD is that it is canonical(unique) for a particular functionality. This property makes it useful in functional equivalence checking and other operations like functional technology mapping. A path from the root node to the 1-terminal represents a (possibly partial) variable assignment for which the represented Boolean function is true. As the path descends to a low child (high child) from a node, then that node's variable is assigned to 0.

**Decision Tree Evolution**

A problem with genetic algorithm is the diversity of the obtained results due to factors like the initial random seed, the initial population and number of generations. The diversity may be surprisingly high for complex search spaces given that we have limited resources (limited number of genomes and generations). Instead of using a big number of generations and an equally big number of genomes, we adopted an alternative strategy that uses relatively few generations and a small number of genomes but repeats the
learner several times. For every output of the cross-validation experiments we repeated
the algorithm 10 times and then picked the highest fit genome (based on training set).

The leaves for the tree are the different grammars associated with the principal
constraints. To direct the decision tree, we developed a set of attributes that will identify
the unique components from the different principal constraints and the grammar
associated with it. The most effective method to analyze the attributes is to process
keywords that lead toward particular constraint. For example, inequality (isneq) is often
associated with the word “NOT” with a strong possibility preceded by the word “IS” and
followed by the word “EQUAL”. In short, a sentence containing “IS NOT EQUAL” has
a high possibility to be of X isneq R type and the grammar that can process the isneq
constraint will most likely be the optimal grammar.

The proposed approach of utilizing genetic programming is to derive the most
optimal perception of a given sentence within the principal constraints. We consider that
each grammar is complete and capable of processing the sentence within the domain.
Each sentence in natural language will be processed according to a set of grammar rules
associated with a principal constraint. Before deciding on which grammar to be used to
process the sentence, we will use a genetic programming technique on the decision tree
aimed towards finding the optimal solution.

We build decision trees that have one decision node that leads to two different
leaves. Every decision node has a random chosen value as its installed test. First we
choose a random attribute. Then, if that attribute is nominal we randomly choose one of
its possible values. This approach reduces the size of the search space and it is
straightforward. Leaves are populated using the same line of thought; we just pick a
random class from the ones available. The basic form of the proposed algorithm introduces minimum changes to the mutation-crossover operators. Mutation chooses a random node of a desired tree and it replaces that node's test-value with a new random chosen value. When the random node is a leaf, it replaces the installed class with a new random chosen class. The crossover operator chooses two random nodes and swaps those nodes' sub-trees. Since predicted values rest only on leaves, the crossover operator does not affect the decision tree's coherence.

Having a population of candidate solutions we need a payoff function (or objective function) to assign utility to each one of them. A natural way to assign utility to a random decision tree is by using it to classify the known instance-set. Then we grant a scaled payoff to the best candidates. Furthermore, we chose to grant higher payoffs to smaller trees (assuming that they perform almost equally with bigger ones). This is a way to avoid unnecessary test-values replications along a specific path (that can happen since we do not exclude any already used attribute-value from being used again) while at the same time we derive comprehensible decision trees. Thus, the fitness function is balanced between accuracy and size. Thus, the fitness function is balanced between accuracy and size:

\[ \text{payoff} (\text{tree} \cdot i) = \frac{\text{CorrectClassified}_i^2 \cdot x}{\text{size}_i + x} \]

The second part of the product (the size factor) includes a factor \( x \) which has to be set to an arbitrary big number. Thus, when the size of the tree is small the size factor is near one, while it decreases when the tree grows big. This way, the payoff is greater for smaller trees. The size factor can be altered to match individual needs. For example, if we had set \( x \) to 1,000,000 then the GA would search inside a bigger search space (more
trees). However, bigger search spaces inevitably mean less optimized trees for a fixed number of generations. Alternative size factors can be used that would prefer trees with sizes inside some range (assuming that we know that the target concept can be represented with a decision tree of a specific size). This could lead to more efficient search and thus less time for the genetic algorithm to converge. To speed up evolution we also implemented an altered version of Limited Error Fitness (LEF). This technique introduces an error limit. If the number of errors of an individual, during the process of evolution, is higher than the error limit, all remaining cases are treated as errors. This means that poor individuals will not be evaluated on the entire training set, saving CPU time. With moderate usage of the error limit we were able to produce significant CPU time savings and insignificant accuracy loses. We have set the basic requirements for our genetic algorithm: an appropriate representation for possible solutions combined with suitable mutation-crossover operators and a payoff function. Here we will come-up with a mathematical formula for the size of search space. This is useful since we would like to achieve a good hypothesis ensuring that we have not exhaustively searched the space.

The size of search space depends on tree size. Let $D(n)$ be the number of topologically different binary decision trees of $n$ leaves shown below.

$$D(n) = \begin{cases} 0 & n = 0 \\ \frac{2n-2}{n(n-1)} & n > 0 \end{cases}$$

The search space depends also on the amount of different attribute-values and classes of the underlying concept. Suppose that $\alpha$ is the sum of the distinct values of all features and that $c$ is the distinct problem classes. Since we use binary decision trees the number of internal nodes is $n-1$. An internal node can use any one of the $\alpha$ distinct values.
and that holds for every node. Since we allow values to be reused, a binary decision tree of n leaves has $\alpha^{n-1}$ syntactically different trees regarding the attribute values. This has to be multiplied with the $c^n$ syntactically different decision trees regarding the problem classes. The total number of syntactically different binary decision trees of $n$ leaves can be calculated using the formula below:

$$T(n, \alpha, c) = D(n) \times \alpha^{n-1} \times c^n$$

When we search for a specific tree we do not stick to trees with specific number of leaves; instead we search on a space containing a wide range of tree sizes. Assuming that the number of training instances is $k$, the maximum number of leaves is also $k$ (one instance at every leaf). The search space can be calculated with the formula:

$$S(k, \alpha, c) = \sum_{n=1}^{k} T(n, \alpha, c)$$

The cost of the proposed heuristic is based on four different factors: the number of generations ($gen$), the number of genomes that are evaluated in the population ($pop$), the number of instances ($k$) and the average path an instance has to follow from the root to a leaf ($avPath$). Then the cost of the algorithm is: $gen \times pop \times k \times avPath$. Quite safely, the $pop$ parameter can be set to a constant multiplier of the number of dataset features $a$ ($pop = cla$) with $cla << a$. Furthermore, under a very pessimistic higher boundary we can set $avPath$ to $k$. With those assumption, the complexity of the algorithm is $O(gen \times k^2 \times a)$.

We can estimate the average number of instances that have to be re-classified in a crossovered and/or mutated tree as a function of tree levels and the original number of instances. This way we can reduce the computational cost of the objective function by recalculating it only for the changed fraction of the tree. This analysis is based on the assumption that instances are equally distributed between nodes. This means that if a
father-node has $k$ instances, then $k/2$ instances arrive at each one of its two children. Another assumption is that nodes are chosen for crossover or mutation with equal probability. Therefore, if we have a tree with size $n$ then the probability of a node to be chosen is $1/n$. Our average analysis deals with the two extremes of binary decision trees: the *linear* binary decision tree and the *complete* binary decision tree. Let $l$ be the number of levels of a binary decision tree. Then, a *linear* binary tree has $l+1$ leaves and a total of $2l+1$ nodes while a *complete* binary decision tree has $2l$ leaves and a total of $2l+1-l-1$ nodes. Any other binary decision tree with $l$ levels lies somewhere between those two ends.

We cannot precisely express the generations needed for convergence since they depend on the complexity of the underlying concept. However, since the genetic algorithm evolves complete solutions, the algorithm can be terminated whenever necessary. We should also not forget that genetic algorithms are highly parallel procedures, and thus, even lower absolute time requirements are possible using a parallel evolution. Another advantage of this procedure is that the output is not just a decision tree but a collection of decision trees that can be used alternatively.

**Testing and Analysis**

We now will test the CDG evolved decision tree, given a particular domain and a set of ST-CT associated with it. The data is defined in an attributed relation file format (ARFF). The ARFF Header section of the file contains the relation declaration and attributes declarations. The relation name is defined as the first line in the ARFF file. The format is: `@relation <relation-name>` where `<relation-name>` is a string. The string must
be quoted if the name includes spaces. Attribute declarations take the form of an ordered sequence of \texttt{@attribute} statements. Each attribute in the data set has its own \texttt{@attribute} statement which uniquely defines the name of that attribute and its data type. The order the attributes are declared indicates the column position in the data section of the file. The format for the \texttt{@attribute} statement is: \texttt{@attribute <attribute-name> <datatype>} where the <attribute-name> must start with an alphabetic character. If spaces are to be included in the name then the entire name must be quoted. The <\texttt{datatype}> for our purpose are ‘Y’ or ‘N’ to indicate whether a sentence is associated with that particular attribute. For example, the class value for a specific domain can be defined as shown in figure 5.1. The example test domain contains nine type of attributes where eight of them represent the type of ST-CT within the domain with the last attribute represent the type of constraints that can be identified by the given CT-ST. The example also contains seven test data that represents sentences within that domain that have been identified to contain particular constraint and its association to the given constraint. This data can now be evolved to determine the most effective way to determine a sentence constraints based on the CT-ST available.

\begin{verbatim}
@relation TestDomain
@attribute 'not' {'y','n'}
@attribute 'probably' {'y','n'}
@attribute 'usually' {'y','n'}
@attribute 'most of the time' {'y','n'}
@attribute 'if' {'y','n'}
@attribute 'because' {'y','n'}
@attribute 'since' {'y','n'}
@attribute 'then' {'y','n'}
@attribute 'constraint' {'is','isp','isu','isfg'}
@data
'y','n','n','n','n','n','n','n','y','is'
'Y','n','n','n','n','n','n','n','n','isp'
'n','n','y','n','n','n','n','n','n','isu'
'n','n','n','y','n','n','n','n','n','isu'
'n','n','n','y','n','n','n','n','n','isfg'
\end{verbatim}

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 5.1 Example of an Attribute Relation File Format

The sample data can be evolved to reach a stable decision tree shown in figure 5.2. Due to the scale of the testing, the most accurate decision tree is achieved within several generations and tends to stabilize in subsequent generations. However, the tree can be further evolved as more data are available. The genetic evolution approach allows for the propagation of the more significant ST-CT to the top level of the tree while the less more important ST-CT are in the lower level. The importance of an ST-CT is primarily defined by the amount of sentences that can be identified correctly while the fitness function ensures that the tree prefers a shorter tree with less depth within the same level of accuracy. In addition, the binary approach allows for a translation to an if-then-else programming form that can be use to efficiently process additional sentence within the domain.

Figure 5.2 Evolved Decision Tree
The tree in figure 5.2 can be easily translated into an if-then-else shown in figure 5.3. This leads to ease of programming that can be optimized with conventional programming methods. The evolution of the decision tree provides an efficient approach to reach the most optimized decision tree as it significantly affects performance as the size of the tree grew. A manual construction of such tree is feasible though tedious and time consuming and not efficiently extensible as new addition to the tree must be made in the lowest leaf level as changes in higher level are not feasible as it could disturb the relation build in the tree while addition to the lowest leaf level are inefficient as it increase the size of the tree. Genetically evolving the decision tree is a feasible solution due to its simple approach and endless possibility to further evolve the resulting tree. An important factor in this approach is the ability to automate the process after the definition of its ARFF irregardless to the size of the attributes.

```
if 'since'='y' then
   |-'isfg'
   +-if 'probably'='n' then
      |-if 'then'='y' then
         |-'isfg'
      +-if 'because'='y' then
         |-if 'not'='n' then
            |-'isu'
         +-'isu'
      +-'isu'
   +-'isu'
   +-'isp'
```

Figure 5.3 If-Then-Else form of the decision tree

It is shown that an evolved binary decision tree provides an effective mechanism to generate a decision tree. A domain expert can create the necessary Sentence Tags and
Properties List for its particular domain of expertise. Subsequently, the expert will generate training data for the decision tree that will be evolved to an optimal form. The resulting tree can continuously be use to process sentences within that domain without further intervention of the domain expert. As the domain grew the tree can also be further improve by adding new training data and evolving a new generation of decision tree. The decision tree can continuously be refined and adjusted as more data become available and to accommodate the expansion of the domain.
CHAPTER VI
CONCLUSIONS AND FUTURE WORK

Machine learning and statistical methods have yielded impressive results in a wide variety of natural language processing tasks. These advances have generally been regarded as engineering achievements. In fact it is possible to argue that the success of machine learning methods is significant for our understanding of the cognitive basis of language acquisition and processing. Recent work in unsupervised grammar induction is particularly relevant to this issue. It suggests that knowledge of language can be achieved through general learning procedures, and that a richly articulated language faculty is not required to explain its acquisition. Fuzzy logic as a practice has proved to be extremely useful and indispensable for various practical applications. However, its best and most important contribution to technology may be in the future, in the form of widespread acceptance and application of the theory of Computing with Words. If so, CwW will change the way we interact with computers and how computers perceive human needs. We see an immediate need for its adaptation to the field of search technology because the ability to search properly through terabytes of data has become a huge problem. The traditional design of search engines leaves a lot to be desired as is evident by the irrelevant results that a user has to wade through as well as unintuitive and ‘inhumane’ nature of the interface. The promise of a Question-Answer system to provide just the right information rather than a list of documents which might contain the information is a promise, which when properly realized would tremendously raise our productivity and accuracy when searching for knowledge.
Summary of Work

In order to achieve a Question-Answer system the first step needed is the ability to augment existing grammars that are capable of parsing sentences into a constraint driven grammar that can also parse the sentence into its appropriate components for computing with words. The methodology to create a constraint driven grammar allows for a deeper understanding of semantics in computing with words. Construction of a robust grammar for a specific domain, which can handle all the possible sentence structures as well as the required protoforms, is a more straightforward though tedious process. Constraint Driven Grammar takes into account the context in which the rules of the grammar may find themselves. The Constraint Driven Grammar approach leads to a more effective model of natural language model for computing with words, in addition to improved speed of parsing and accuracy of choosing the correct parse utilizing genetically evolved decision tree.

Future Directions

The focus on our research has been on implementing the proposed system for a specific domain utilizing sentence tags. We can also expand the role of sentence tags to also provide a representative semantics into fuzzy words such as very, most likely and probably. The tags can now represent probabilistic values that represent fuzziness. Additionally, the fuzzy sentence tags can also be improved by utilizing a context sensitive fuzzy sentence tags that have different values in different context within the same domain.
There is also the possibility to improve the binary decision tree with a type of fuzzy decision tree where a node can have different branches depending on context and associated values. Such tree will be more efficient than the binary tree with yes or no answers but are more difficult to construct and might be incompatible to apply genetic operators.
APPENDIX

ATTRIBUTE RELATION FILE

In order to demonstrate the training involved in the decision tree within a specific domain, we must generate sufficient data that can address the different constraints of computing with words as shown in Chapter 5. Our testing includes possibilistic, probabilistic, usuality and fuzzy graph constraints.

@relation CwW
@attribute 'not' {'y','n'}
@attribute 'probably' {'y','n'}
@attribute 'usually' {'y','n'}
@attribute 'most of the time' {'y','n'}
@attribute 'if' {'y','n'}
@attribute 'because' {'y','n'}
@attribute 'since' {'y','n'}
@attribute 'then' {'y','n'}
@attribute 'constraint' {'is','isp','isu','isfg'}
@data
'y,n,n,n,n,n,n,n,n,isp'
y,y,n,n,n,n,n,n,n,isp'
y,y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
y,n,n,n,n,n,n,n,isp'
Possibilistic constraints ST-CT are ‘not’, probabilistic tags are ‘probably’, usuality tags are ‘usually’ and ‘most of the time’ while fuzzy graph tags are ‘if’, ‘because’, ‘since’ and ‘then’. Sentences that contains more than one tags of the same kind are only counted once such as sentence containing the tag ‘if’ and ‘then’. Next we provide the accuracy graph of the decision tree from start to finish.
REFERENCES


