ERROR TOLERANT FINITE STATE PARSING FOR A TURKISH DIALOGUE SYSTEM

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BİLKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Atacan Çonduroğlu
July, 2001

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. İlyas Çiçekli (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. David Davenport

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Bilge Say

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet Baray
Director of the Institute

ABSTRACT

ERROR TOLERANT FINITE STATE PARSING FOR A TURKISH DIALOGUE SYSTEM

Atacan Çonduroğlu
M.S. in Computer Engineering
Supervisor: Asst. Prof. Dr. İlyas Çiçekli
July, 2001

In NLP (Natural Language Processing), high level grammar formalisms are frequently employed for parsing. Since in practice no formalism can cope with the diversity and the flexibility of the human languages, such formalisms are used in closed domains, with sub-languages. Even though we believe that in an open world sophisticated analysis is required for extracting meaning from natural language texts, this does not have to be the case for the closed domains. Simpler time-efficient finite state methods can be used in closed domains. With their simplicity and time-efficiency, finite state methods are not only responsive, but also easy to augment with error tolerance which allows these methods to flexibly parse mildly ungrammatical sentences. In this thesis, we present a parser module which is based on error tolerant finite state recognition and a grammar for parsing transcribed dialogue utterances in a closed Turkish banking domain. Test results on the syntheticly created erroneous sentences indicate that the proposed system can analyze ungrammatical sentences efficiently and can scale with the growth of the grammar.

Keywords: Natural Language Processing, Finite State Parsing, Error Tolerant Recognition, Robust Parsing, Light Parsing, Dialogue Systems, Natural Language Interfaces.

ÖZET

BİR TÜRKÇE DİYALOG SİSTEMİ İÇİN HATA TOLERANSLI SONLU DURUM ÇÖZÜMLEME

Atacan Çonduroğlu
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Yard. Doç. İlyas Çiçekli
Temmuz, 2001

Doğal dil işlemede, yüksek seviyeli gramer formalizmaları cümle çözümlemesi için sıkça kullanılmaktadır. Pratikde hiç bir formalizma doğal dilin esnekliği ve çeşitliliği ile baş edemediği için, bu formalizmalar sınırlı alanlarda, alt-dillerle kullanılmaktadır. Her ne kadar sınırlandırılmamış bir dünyada, cümlelerden anlam çıkarabilmek için sofistike metodlar kullanılması gerekteğine inansak da, sınırlı alanlarda durum böyle olmak zorunda değildir. Basit, ama daha zaman-verimli olan sonlu durum makina metodları, kapalı alanlarda çözümleme için kullanılabilir. Basitlik ve zaman verimlilikleri ile bu metodlar sadece etkileşimli değil, ayrıca onların hafif gramer dışı cümleleri çözümlemelerini sağlıyan hata toleransı ile geliştirilmeleri daha kolaydır. Bu tezde, Türkçe banka diyalogları alanındaki, diyalog cümlelerini çözümlemek için, hata toleranslı sonlu durum tanımaya dayanan bir çözümleme modülü ve grameri sunmaktayız. Sentetik olarak yaratılmış hatalı cümleler üzerindeki test sonuçları, önerdiğimiz sistemin gramer dışı cümleleri verimli bir şekilde analiz edebildiğini ve büyüyen gramerle beraber ölçeklenebildiğini göstermektedir.

Anahtar sözcükler: Doğal Dil İşleme, Sonlu Durum Çözümleme, Hata Toleranslı Tanıma, Sağlam Çözümleme, Hafif Çözümleme, Diyalog Sistemleri, Doğal Dil Arayüzleri.

Sevgili aileme

Acknowledgements

I am grateful to my advisor İlyas Çicekli for his guidance, motivation, and patience during the writing of this thesis. I really enjoyed to work with him.

I would also like to thank Prof. Kemal Oflazer for giving the original idea of this thesis and providing me the necessary development environment. His teachings have been most valuable during the development of this thesis.

Thank also go to Asst. Prof. David Davenport and Asst. Prof. Bilge Say for reading and commenting on this thesis, and to my colleagues and friends for their moral and intellectual support during my studies. Among many others, I am very thankful to Eray, Arda, Umut, Barla, Mustafa, and Selim for helping me in my darkest hours. Especially Eray, has provided me the patronage I needed in the realm of computers both during my graduate and undergraduate life.

I would like to thank to my family. Without their support and love, I could never taste any achievement in my life.

Last-place in here, though not in my heart, I would like to thank Nurfer for her precious being. Beyond all others, it is her support and friendship, which made this thesis possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In NLP (natural language processing), finite state methods have been largely neglected due to the fact that natural languages are not regular. Every natural language has at least one structure which requires a higher level grammar formalism. Beyond this weakness, among the higher level grammar formalisms regular grammars are assumed to be harder to write, since they require many more rules then the other formalisms.

But yet, in practice no grammar formalism can cover the performance set of a human language speaker. For this reason robustness in natural language processing is necessary. While facilitating the coverage of complex grammatical phenomena, the machines based on higher level grammars suffer from being order of magnitudes slower than the machines based on regular grammars, and this weakness means it is harder to make them both robust and responsive.

Although in an open world higher level grammars may prove to be more practical, we believe that in a closed world, a robust finite state parser may serve sufficiently. This thesis aims to test this hypothesis through building a robust finite state parser for a dialogue system for the Turkish banking domain.

1.1 Dialogue Systems

A dialogue system[1] provides a natural language interface to its clients, so that the clients may interact with the target system in their own language. In a dialogue system, the set of possible interactions are restricted to a specified domain.

In parallel to this restriction, the language used in a dialogue system is only a subset of the daily language we use. This sub-language, being highly dependent on the dialogue system's domain, includes only a small fragment of interpretations and usages of the words with a small number of syntactical structures.

Some well known dialogue system domains are, TRAINS, ATIS (Air Travel Information Services), and OVIS (public transport information system[2]).

TRAINS is one of the oldest experimental fields for dialogue systems [4, 11]. This domain has been created by Rochester University for research on conversational and planning agents for spoken English language. In the TRAINS domain, clients schedule freight trains for transporting their goods to different locations, by cooperating with a spoken language system. Both the number of possible goods and the number of possible railway paths are limited in this domain. The only available goods are oranges or bananas and the number of rail-lines and cities does not exceed a couple of dozen.

ATIS is characterized by the conferences organized by ARPA under the same name [10]. These conferences can also be thought of as competitions among the attendees. Their systems are expected to facilitate retrieving of airline schedules, fares, and related information from a relational database, using spoken English. In ATIS, both the number of cities and flights are limited with the includes of the relational database. In contrast with the TRAINS domain, ATIS involves no planning or demanding cooperation with the client. An example dialogue from the ATIS domain can be seen below.

[1] Also called spoken dialogue system, spoken language system, natural language interface, and dialogue agent.

[2] In Dutch the original name is: Openbaar Vervoer Informatie Systems.

> ***human:*** what are the delta airline's flights from boston to denver
> ***computer:*** there are 2 delta airline's flights from boston to denver:
> delta airline's flight 7 oh 8 connecting ...

In OVIS, information on the public transformation is serviced as in ATIS domain [25]. The domain differs from the other two preceding domains in that the language involved is not English but Dutch. Similar to other domains OVIS is also highly restricted. In OVIS, only railway information about certain cities are serviced in order to ease address related problems and avoid complex sentences which may describe transportation through more than one type of vehicle.

1.1.1 Anatomy of a Dialogue System

Most of the dialogue systems share a common architecture. This similarity among dialogue systems can be observed in SRI's implementation of ATIS, Rochester university system for TRAINS, and ovis1 project for the OVIS domains.

As illustrated in Figure 1.1, these systems comprise four important common sub-parts: speech recognizer, parser[3], dialogue controller, and answer generator.

In this ideal schema, the input sentence is first processed by the speech recognition module and converted from its acoustic form into a transcribed sentence or candidate sentences. Hidden Markov models and statistical methods have been the *de facto standard* in speech recognition. These methods, through mathematical transformations, convert digital sound into forms which better reflect the relation between the sound and its transcribed form. Then the ambiguities arising after the transformation are tried to be resolved by investigating the probabilities of the candidate character sequences in the domain language. At this stage, usually a certain ambiguity remains and rather than a single sequence of words, many sequences are given as output.

After speech recognition, the resulting sequences of words are fed into parser

[3]Sometimes called the NLP module.

module. In the parser module, the input is converted into a semantic representation. The semantic representation coming from the parser module is used to generate corresponding actions, and responses to the input.

The dialogue controller[4] is another important modules in the process of generating responses and actions. The dialogue controller functions as the memory of the whole system and keeps track of the ongoing conversation. This module formalizes the high-level purposes of dialogue utterances, and augments the whole system with a sense of awareness. While a parser deciphers the syntactic structures of a dialogue utterance, a dialogue controller interprets the utterances at a *pragmatic* level and categorize them according to their speech acts, such as QUESTIONING, ACCEPTING, REJECTING. In this way, a dialogue controller may say that a *rejecting* utterance will follow the current *questioning* utterance of the user. Such pragmatic level relations among the speech acts usually are modeled with finite state machines in the dialogue controller module.

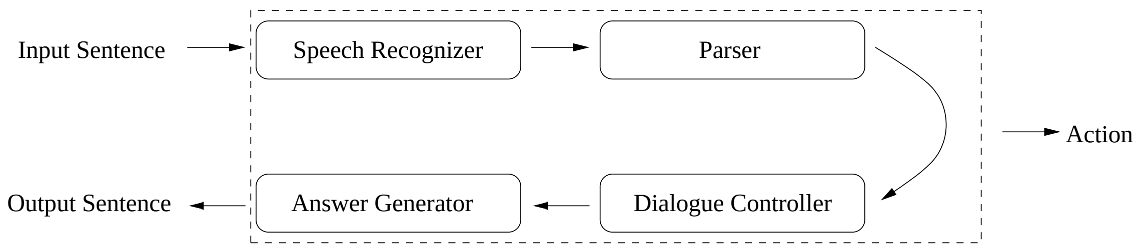


Figure 1.1: Anatomy of a basic dialogue system

Finally, the response generated by the parser and the dialogue controller module is translated back into natural language in the answer generation module. In some cases, the response can be already in natural language, but usually it is only a raw result coming from a relational database query. Generative grammars in cooperation with the speech acts are used to make necessary transformations. For applications requiring voice response, either the textual result of the answer generation module is converted back into speech in a text-to-speech module or the answer generation module is designed so that it produces acoustical forms.

[4] It is also called a dialogue manager.

There might be also other modules involved in a dialogue system such as a knowledge base module, a reasoning module, or a visualization module. But these modules are omitted in this work, since they are rather loosely bound with the parser module, which is the theme of this thesis.

1.1.2 Parsers

Parsers in dialogue systems must be robust[5](error tolerant). Spontaneous spoken language, whether transcribed or speech recognized, deviates from conventional grammatical structures. These phenomena can occur at word-level and sentence level. A word may be misspelled or some words can be repeated or forgotten or replaced with a meaningful but wrong word. The parser of a dialogue system should be able to cope with such problems.

In addition to robustness, a parser must also be time-efficient in order to be responsive. Complex analyses, which require too much time, cannot be employed in a dialogue system's parser. This requirement, forces parsers in interactive systems to balance robustness with the responsiveness [24].

In SRI's ATIS implementation [7], a layered parsing scheme is used for solving the robustness problem. SRI's context-free grammar parser GEMINI [9], builds layer by layer parsed islands of words on the input sentence. The parsing operation is stopped when these islands cannot be extended any further. In this way, even if a sentence cannot be parsed totally, the system can still produce a partial parse for it.

Semantic constraints and repair heuristics are used to augment these partial parses. By using semantic constraints, the possible readings for an unknown word sequence are limited. These semantic constraints are indeed simple domain facts such as "a flight which is available on Tuesdays can not be booked for any other day.". As repair heuristics, a list of previously chosen cue words are used either for deleting an unnecessary word, such as deletion of the word "Ehhhm", or

[5]Robust and error tolerant terms will be used interchangeably in this work.

replacing a previously uttered word with the succeeding word, such as replacing six with seven after seeing the word “no” in the utterance “six no seven”.

Similar to GEMINI, a bottom up context-free parser is used in Rochester’s TRAINS domain. The main difference in this parser is that it creates speech acts rather than logic formulas. Sentences are marked with speech acts such as, REQUEST, CONFIRM, SUGGEST. Then these speech acts are used to model the dialogue and interpret the requests of the client. Through this higher level modeling of the dialogue, an additional robustness is achieved.

In the ovis1 system, robustness is achieved by using a very simple partial parsing mechanism. Deriving from the work done in Philips [6], the parser of the ovis1 system marks information carrying units and performs stochastic processes on these units to decide their correct forms and corresponding actions.

1.2 Finite State Parsing

Besides the ovis1, simple regular grammar rules are not used for parsing in dialogue systems. Instead, finite state methods are often used for modeling the pragmatic states of the conversation. Although there exist partial finite state parsers, they are either experimental or used with higher level grammars as a pre-processing step.

In contrast to their limited usage in parsing, finite state parsing methods have been recently used and found adequate in information extraction tasks. As reported in [14, 22], these techniques have been successfully employed in the message understating conferences (MUC), where the main goal is to fill entries of the information forms from a given news text. For closed domains, it has been shown that finite state transducers may have more advantages than higher level grammar methods in extracting the semantics of the sentences.

One of the advantages of finite state machines is that they are algorithmically faster than the machines used for languages high up in the Chomsky hierarchy.

For this reason, the systems based on the finite state techniques can be tested in shorter times, and can be debugged and developed faster. Also such systems are more adequate for on-line operations, where parsing time really matters.

Another advantage of finite state machines is their ease of definition. Even though finite state grammars, particularly for approximating recursive structures of the languages, are thought hard to write and found rather less intuitive, higher level grammars, such as context-free grammars, present much greater problems. In [22], it has been told that context-free grammars are harder to modify and maintain for closed domains, since they lack the simplicity of the regular grammars used for defining finite state machines.

1.3 Our Objective

In this thesis, we try to develop a finite state parsing module for a dialogue system. The dashed area on Figure 1.2 marks the required components of this parsing module, which are an error tolerant finite state parser and its grammar.

Since there is no available corpus of task-oriented spoken dialogues in the Turkish language, we have tried to choose a rather well defined and closed domain which is both challenging and important. We thus selected the Turkish Banking Domain. Our parsing module takes punctuation-free lower case transcribed Turkish utterances as its input and generates a marking on it that describes the corresponding banking domain operation.

1.4 Outline

The next chapter presents previous work done in the error tolerant (robust) parsing area and establishes the required background on the error tolerant finite state parsing. Chapter 3 describes the linguistic properties of Turkish and gives the

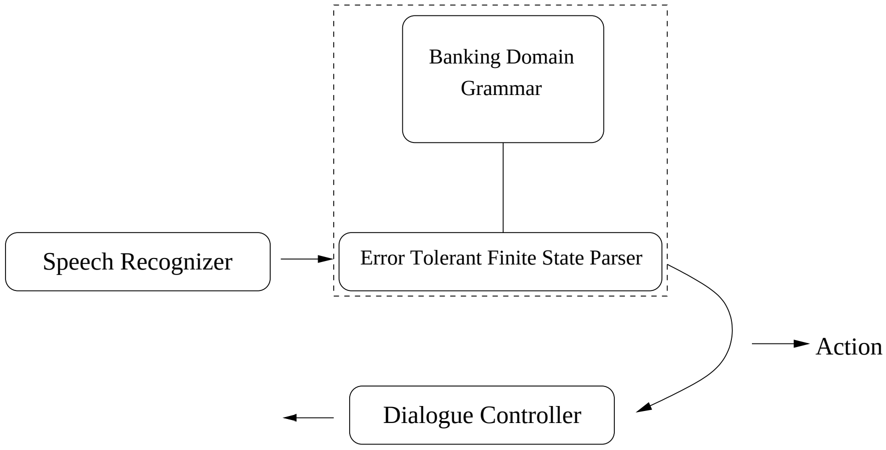


Figure 1.2: Detailed diagram of a dialogue system

specifications of the banking domain. After these preliminaries, the chapter continues with the details of the design and implementation of the banking domain grammar and finally concludes by showing how this grammar is integrated with the error tolerant parser to form the parsing module. Chapter 4 is on the implementation and the evaluation of the module. In the last Chapter we discuss our results and the possible future extensions and research.

Chapter 2

Error Tolerant Parsing

A parser is defined as an algorithm or program to assign structural descriptions to the sentences of a language according to some rules. The rules used by the parser are called a grammar and the structural description generated by the parser is called a parse. A parse serves as a proof of well-formedness of the input sentence for a grammar.

Besides this, the parse is used to assign semantics to the input string. In programming language domain, compilers (parsers) generate a two dimensional tree structure from a one dimensional symbol sequence. The compositional tree form of a parse is closer to the compositional semantics of a programming language than the former flat symbol sequence form. For this reason, the generation of a tree structure in the programming languages domain by context-free parsers has proven to be very effective.

This success of context-free parsers has greatly influenced natural language processing. In the early days, the only problem for creating a semantics for natural language sentences, and hence making a machine understand human language, seemed to be writing an appropriate context-free grammar for it.

Such a grammar could not been written. The possible sentences that a natural language does have could not be covered by any grammar. In addition to this coverage problem, there is the problem of dealing with the performance of human languages. In daily life while using the language, we repeat words, skip verbs, and insert exclamations. A parser for natural language should thus be error tolerant in order to deal with the performance aspect of the human languages. And beyond these problems, there is the problem of ambiguity. Natural languages are inherently ambiguous at both the word and sentence level. A parser which can not resolve or give metrics to resolve such ambiguities can not be competent in understanding natural language.

By restricting the language to a closed domain we can decrease both the ambiguity and the required coverage, though the error tolerance problem remains.

2.1 Prior Art in Error Tolerant Parsing

Error tolerant parsers may be divided into three categories: grammar oriented parsing methods (symbolic), statistical parsing methods and connectionist parsing methods.

Statistical and connectionist methods are both error tolerant by their nature. However, for their training these methods depend on the previously parsed corpora which is in many cases very hard to find or create. Moreover, connectionist and statistical methods lack transparency. For this reason, a system trained for a domain can not be modified to work under a new domain, but must be completely retrained with a new corpus. Because of these disadvantages, grammar oriented error tolerant parsing methods are used more often than statistical or connectionist methods.

2.1.1 CFG Partial Parsing

In contrast to conventional context-free grammar parsers, partial parsers do not try to span the whole input sentence and generate a forest of parse trees which cover the ungrammatical sentence as much as possible. These well-formed subsequences of the input sequences are referred as certainty islands which are surrounded by the noisy uncertainties.

Partial context-free grammar parsers are usually extensions of the well known full parsers. They are exemplified by GLR[1] and GLR* parsers based on the LR parser [17, 18], GEMINI from the chart parser [9], and LCFlex from the left-corner parser [24]. These parsers relax their basic parsing algorithm so that word insertions, deletions, or replacements in their (ungrammatical) input are tolerated.

Insertion is the most common form of tolerance found in these parsers. The reason for this is that word insertions correspond to frequently occurring lingual mistakes in spoken language, such as repetitions, false starts, filled pauses and exclamations. *Word skipping* is the term used for methods handling the word insertion errors.

Besides word skipping, in partial parsing other types of possible error tolerances are mostly discarded, since each new toleration schema adds additional amount of computation time to the system and hinders responsiveness.

2.1.2 CFG Minimum Distance Parsing

Minimum distance context-free parsers (MDPs) are the most disciplined approach to error tolerant parsing. MDPs are not partial parsers; like conventional parsers they span the whole input.

In minimum distance parsing, for ungrammatical inputs every possible insertion, deletion, and replacement of words are taken into account and the grammatical sentence with the minimum operations away from the original input is parsed.

Hipp's parser [13] is one of the best known context-free minimum distance

[1] GLR stands for Generalized LR.

parsers. It is a modified chart parser which searches the CFG for the parse with the minimum error penalties. In [13], it has been shown that their MDP can be successfully used as a parser module for a dialogue system in the closed domain of "electronic circuit repair assistance".

2.1.3 Efficiency of the MDPs and Partial Parsers

The time complexity of any error tolerant system depends directly on what kind of deviations are allowed from the original grammar. Since the conventional context-free parsers work in the order of $O(n^3)$, where n is the length of the input symbols, the effect of such tolerances on time complexity is striking in CFG based methods.

Even though MDPs are the most complete approach to error tolerant parsing, they are computationally infeasible. For larger domains with broader grammars, MDPs tend to be intractable [23], and thus not frequently used in practical applications.

And for the partial parsers, which are used most frequently, there always have been a dilemma between the flexibility and the time-efficiency of the parser. Partial parsers either sacrifice their robustness for being time-efficient or their responsiveness for being error tolerant.

2.2 Error Tolerant Finite State Parsing

As we have said before, there is no grammar which can cover freely occurring natural language texts. For this reason, parsers are always tested and developed for closed domains. Even though our intuition tells us that in an open world the semantics of a sentence should be built compositionally, for a closed world we do not have any proof of necessity of tree structured parses, or broad coverage grammars, or syntactical analyses of the sentences. Simple, yet time-efficient finite state methods may prove to be adequate in such closed domains.

2.2.1 Regular Grammars

While context-free parsers are the machines for accepting context-free language sentences, finite state machines[2],are the acceptors for the sentences of the regular languages.

Regular languages being a subset of context-free languages, may be represented by the context-free parsers. But still they have properties that make them more suitable to work with than the CFLs.

As pointed in [15] RGs are:

- Closed under Union;
  If L_1 and L_2 are regular, $L_1 \cup L_2$ is also regular.
- Closed under Intersection;
  If L_1 and L_2 are regular, $L_1 \cap L_2$ is also regular.
- Closed under Complementation;
  For every regular L there is a $\overline{L}$ exactly accepting the strings which are not accepted by the L.

However, CFGs are not closed under intersection and complementation. While it is possible to easily filter RGs by subtracting sub-grammars from the grammar, this is not the case for the CFGs[3].

In addition, under RG formalism, there is a one-to-one correspondence between the regular expressions, rules of the grammar, and the FSM. A unique minimum FSM can be easily built for a regular expression, and for every FSM the corresponding regular expression can easily be extracted. By building the unique minimum FSMs of the two grammars, one can easily show their equivalence, whereas this is not possible for CFGs [15].

[2]Through the chapter FSMs will be used for referring to the deterministic finite state machines.

[3]CFGs can be subtracted RG patterns. But all CFG sub-grammars are not regular, and therefore may not be subtracted.

A FSM is formally a 5-tuple $(Q, \Sigma, \delta, q_o, F)$, where Q is the finite set of the the states of the machine, Σ is the finite input alphabet, δ is the transition function mapping state and alphabet symbol pairs into states, q_0 is the start state, and F is the set of final states of the machine.

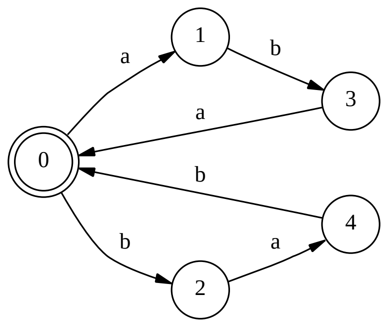


Figure 2.1: A sample FSM.

A sample FSM machine for the grammar (a b a + b a b)* is illustrated in Figure 2.1, where Q is $\{0, 1, 2, 3, 4\}$, Σ is $\{a, b\}$, q_0 is the 0, and F is $\{0\}$.

In FSMs, a directed graph is used to represent δ, the transition function. On the graph, the vertices correspond to the states of the FSM, whereas edges with labels on them represent the transitions from one state to another. Actually the graph depicted in Figure 2.1 is the sigma function of the corresponding FSM.

For a given input, FSM starting from the state q_0, *consumes* the symbols of the input to do the necessary transitions on the transition graph. When the input string is finished, the state on which the machine has stopped determines the validity of the input. If it is a final state, the input is well-formed and accepted, else it is rejected.

2.2.2 Finite State Transducers

One disadvantage of FSMs is that they don't have any output other than accept or reject. For this reason, they can not be used for parsing.

This disadvantage is overwhelmed by the FSMs with outputs, called finite state transducers(FST), or Mealy machines. Different from a FSM, a FST has an additional alphabet Δ for the outputs, and an additional λ function mapping the input symbol and state pairs into members of the Δ.

The λ function of FSTs are represented as additional symbols on the transition arcs of the FSMs, corresponding to the output symbols for transitions. The input and output symbols on the arcs are separated either with "/" or ":". The symbol on the left of the separator is called an input symbol(or a left symbol, or an upper symbol), whereas the symbol on the right is called an output symbol(or a right symbol, or a lower symbol). Since there might be transitions with no outputs and outputs which do not consume any input symbols, empty symbols (epsilon symbols) are necessary for some FSTs. The epsilon symbol is either represented with 0 or with ϵ.

Thus, a FST is a FSM with two regular languages, an upper and a lower language. A FST maps the inputs in its upper language into the outputs in its lower language. By choosing the lower language properly to mark structures in the input language, a FST can be used as a parser for a regular language. Such a machine is illustrated in Figure 2.2 which marks the pattern "a" with brackets for the language accepted by the FSM in Figure 2.1.

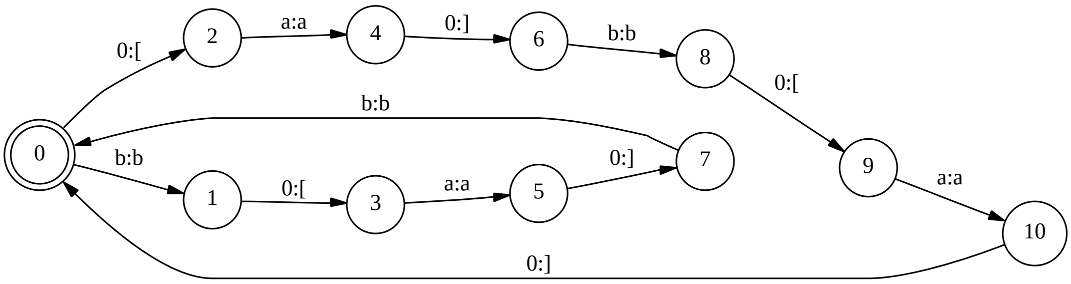


Figure 2.2: A FST for the language accepted by the FSM in Figure 2.1

Some sample input output mappings for the FST in Figure 2.2 are listed below.

```
input:  ababab
```

```
output: [a]b[a]b[a]b
input:  ababa
output: REJECTED
input:  abaaba
output: [a]b[a][a]b[a]
```

While their lower and upper languages exhibit all the characteristics of the regular languages, the language of the FSTs themselves may not be intersected, and therefore can not be subtracted. But a FST's upper and lower languages are still regular languages, and intersection and subtraction operations can be applicable.

FSTs also have operations which are unique to themselves. One operation which can be done on the FST machines is *cascading*. In cascading, FSTs are combined together, such that one FST's output is fed into another FST's input. Through cascading, modular FST systems can be easily developed. Moreover, with an another operation called *composition*, these cascades of the FSTs can be converted into a single FST, which is smaller than the sum of the all FSTs in the cascade.

2.2.3 Error Tolerant Finite State Parsing Algorithm

Similar to the CFG minimum distance parsing algorithm, the error tolerant finite state parsing (ETFSP) algorithm[4] [20, 21] relaxes the FSM's parsing method with insertion, deletion, and replacement tolerance. Under this scheme, while consuming the input symbols, instead of transitions with the correct input symbols, every possible transition is followed. In this way, many candidate in-grammar substrings are formed for the ungrammatical substring.

The candidate transition paths are terminated when the candidate in-grammar substrings deviate more than a threshold (*maximum tolerance*) from

[4]In his original paper Oflazer has named this algorithm as "Error Tolerant Morphological Analysis".

the original string. The deviation from the original string is measured by the edit distance, which is the minimum number of insertion, deletion or replacement operations to transform the deviated string into its correct form. Figure 2.3 illustrates the edit distance between the words "hesaplar" and "hespler".

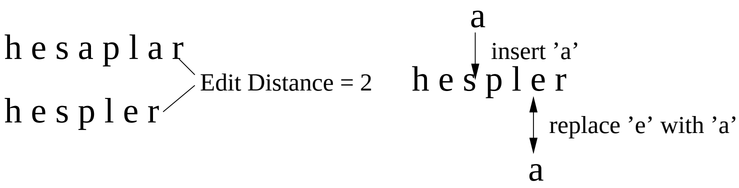


Figure 2.3: Edit distance between two words.

The *cut-off* edit distance is the algorithm used in [20] to prune the search on a FSM's graph. As it is defined in [20], the cut-off edit distance is the method which gives the minimum potential edit distance between an ungrammatical input and the partial output. The cut-off edit distance between an input of length m and its possibly candidate in-grammar form of length n, is defined as below, where t is the maximum tolerance, l is $max(1, n-t)$, and u is $min(m, n+t)$. In the formula, $cuted(input, partial)$, represents the cut-off edit distance and the $prefix(word, i)$ represents the prefix of a $word$ with i characters.

$$cuted(input, partial) = \min_{l \leq i \leq u}(editDistance(prefix(input, i), partial))$$

ETFSP can be thought as a depth first search (DFS) on the FSM's graph, with cut-off pruning, for reaching to a final state. The core of the ETFSP which makes it different from the DFS is the error tolerant DFS visit described in the algorithm in Figure 2.4. For error tolerant parsing the error tolerant DFS visit is initially called as $error\_tolerant\_dfs\_visit(q_0, \epsilon, \epsilon)$, which sets $state$ to q_0, and $upper$ and $lower$ to ϵ.

While in conventional finite state parsing only a single path is followed, in ETFSP there is an area of searched paths as illustrated in Figure 2.5, where the darkened area represents the paths searched by the ETFSP.

procedure $error\_tolerant\_dfs\_visit(state, upper, lower)$
for all $arc \in state$ **do**
  $push(upper, arc.upper\_symbol)$
  $push(lower, arc.lower\_symbol)$
  if $cuted(input, upper) < max\_tolarance$ **then**
    if $state \in F$ **and** $editDistance(input, upper) < max\_tolarance$ **then**
      Output the *lower* as result.
    end if
    $error\_tolerant\_dfs\_visit(arc.next\_state, upper, lower)$
  end if
  $pop(upper)$
  $pop(lower)$
end for

Figure 2.4: Error tolerant dfs visit algorithm

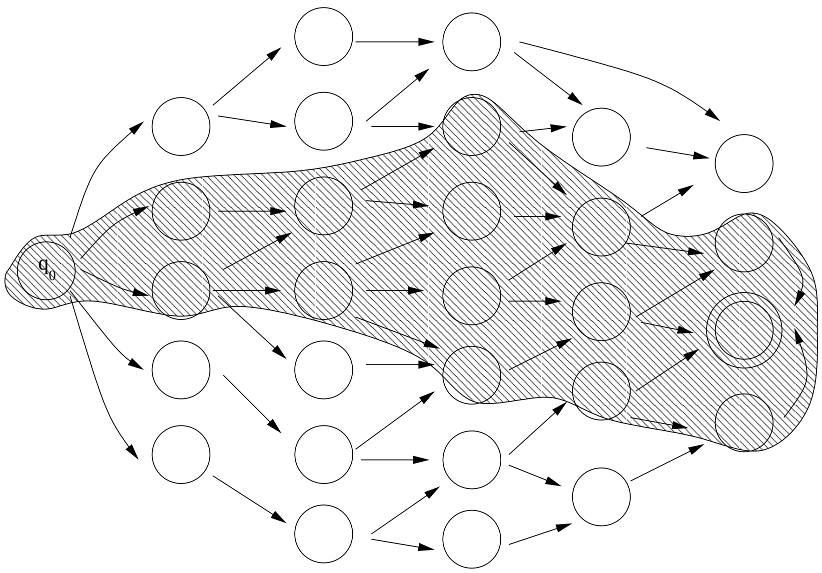


Figure 2.5: The search area of the ETFSP on the FSM graph.

2.2.3.1 Efficiency of ETFSP

Like many searching algorithms, the time complexity of ETFSP is bounded by the area it searches. In worst case it tries all possible paths in the search area of the FSM's graph. On each path before extending our initial partial candidate in-grammar string, we calculate the cut-off edit distance between the candidate and the input. Since in [20] it has been shown that edit distance can be calculated with dynamic programming very efficiently, the complexity of ETFSP is dominated by the number of paths in the search area, which can change dramatically according to the input and machine's topology[5]

[5]In [20] it was claimed that ETFSP will work best on FSMs with dense graphs.

Chapter 3

Grammar

Our aim in writing a grammar for the Turkish banking domain, is to map utterances in Natural Language into banking domain operations, such as reporting the amount of an account or buying funds. Mixed-initiative interaction with the user is currently not one of the concerns of the system, since we intend only to prove the usability of our parsing module, rather than develop a complete dialogue system.

In contrast to wide-coverage CFGs, in our grammar we do not employ high-level syntactic analysis of the Turkish banking domain utterances, since they cannot contribute much to our purpose of mapping utterances into functions. Representing a word as a noun or verb is not necessary. Without employing any part of speech information, in a closed domain we can interpret a word's semantic contribution to the sentence from itself and its context.

As listed below, although the Turkish word "dolar" can be interpreted in many ways, only the first two interpretations may occur in the banking domain, and the relative position of these words to the other constituents of the sentence would be sufficient to decide on the correct interpretation.

- A singular noun, referring to the currency used in USA.
- A verb, meaning that a duration does end.

- A plural noun, referring to the plural from of a musical note.
- A verb, meaning that an agent does wind.
- A verb, meaning that an object does fill.

3.1 Linguistic Properties of Turkish Language

Turkish is an Ural-Altaic language with many interesting morphological and syntactical phenomena not found in many western languages.

3.1.1 Morphology

Turkish is an agglutinative language with suffixational morphology. Suffixes are added to the root word like *beads threaded on a string*, and each suffix added to the word contributes an additional information piece.

In the Turkish language, derivational and inflectional morphological phenomena is highly productive. Theoretically the number of root word and suffix combinations one can use in Turkish is infinite. In practice, though, the number of combinations occurring in daily language is far less than the theory says, memorizing each combination for Turkish has been found impractical. The possible root word and suffix combinations for Turkish are described with a list of root words and suffixes, and rules governing their orderings[1].

While the lexicon and morphotactics can describe the possible words in the Turkish language, this description itself is not enough for creating the words' surface forms, which are the forms occurring in the language. Different phenomena such as vowel harmony, vowel drops and consonant drops, may change the surface forms of the words and suffixes in Turkish language. The lexicon representation of the root words and suffixes are realized in different surface forms under the

[1] The list of words and suffixes is formally called a *lexicon*, and rules are called the *morphotactics*.

government of the *morphographemic* rules. A simple example, showing how the surface forms of the same suffixes may change according to their context, is given below.

Lexicon form	hesap + im(*first person possessive suffix*) + da(*locative suffix*)
Surface form	hesabımda
Lexicon form	portföy + im(*first person possessive suffix*) + da(*locative suffix*)
Surface form	portföyümde

Since the surface forms of the root words and suffixes are subject to change according to their context, lexical level root words and suffixes are described in an abstract way such that their morphographemics is described with ease in the lexicon.

3.1.2 Syntax

Turkish language syntax relies more on morphological information than most other languages. In Turkish, word cases are marked by their morphemes, whereas in English, cases are marked by their positions. Also many auxiliary words, such as the ones used as propositions, or articles, or auxiliary verbs in the English language and in western languages, do exist as morphemes in Turkish.

One of the linguistic reasons that Turkish syntax relies so much on the morphology is that Turkish is a free word order language. English is termed to be a *SVO* language, meaning that in a typical English sentence the subject precedes the verb and the verb precedes the object. However this not the case for the Turkish sentences. In Turkish all six combinations of subject, verb and object are grammatically permissible.

But in Turkish the order of the constituents of a sentence effects its meaning. Sentences initial constituent determines the topic, the constituent preceding the verb is the emphasis, and the constituent succeeding the verb gives the background or "after thought" information. Under such pragmatic control, although all possible orderings are sound in Turkish syntax, some constituent orderings are much more frequent than the others [12], as illustrated in Table 3.1.

Order	**Percentage**
S O V	48%
S V O	25%
O V S	13%
O S V	8%
V S 0	6%
V O S	0%

Table 3.1: Percentage of different word orders (among 500 utterances)

Especially in a dialogue environment, the rules of the pragmatics largely limits the frequency of these word orderings. A question inquiring about the amount of a bank account cannot be asked with as many word orderings as a sentence lacking such a specific context. For this reason, although at first glance Turkish sentences may seem to lack word order information, it is not the case.

3.2 Banking Domain

The Turkish Banking domain that we have designed is inspired from the telephone and Internet services recently started [1, 3]. We have borrowed both their service definitions and formal terms, while designing our own domain.

The domain can be examined in two important sections, the objects of the domain, and the operations of the domain. The objects of the domain are the elements which exist in our closed banking world, whereas the operations of the domain are the possible actions in that world.

3.2.1 Objects of the Banking Domain

The objects of the banking domain have two distinctive properties, their *types* and *configurations*. The type of an object serves as an abstract description for the object, whereas its configuration gives finer details. The configuration of an object is represented with different *attributes*. The values of these attributes decide on the configuration of the object. Table 3.2 lists the object types, while Table 3.3 lists attributes and possible attribute values.

Object type	**Attribute**
Account	A (account type) and C (currency)
Currency	C (currency)
Transaction	T (transaction type)
Fund	B (fund type)
Term duration	TD
Number	-

Table 3.2: Object and their attributes in the Banking domain

Attribute	**Values**
Account(A)	AL (acc. loan), AI(acc. investment) AT1(savings acc. term), AT2(savings acc. non-term) AC1(savings acc. credit), AC2(savings acc. non-credit)
Currency(C)	C0(indefinite foreign currency), C1(TL), C2(FR),C3(USD),C4(DM)
Transaction(T)	T0(any), T1(put), T2(draw), T3(buy), T4(sell)
Term Duration(TD)	TD1 (monthly), TD2(yearly)
Fund (B)	B1(B one type fund), B2(B two type fund)

Table 3.3: Attributes and their values in the Banking domain

As expected, the most important object of the banking domain is a bank account. Bank accounts can be of three types, they are either investment accounts, loan accounts, or savings accounts. The savings account type has four sub-types, credit savings account, non-credit savings account, termed savings account, and non-termed savings account. All six types of the accounts are represented with an account type attribute (A) as depicted in Table 3.2 and Table 3.3. Another attribute of the account is its currency (C), whose possible values are listed in Table 3.3. Although the configuration of a bank account object depends to the possible values of these two attributes, because of domain constraints some combinations of the attributes are not allowed. A loan, or an investment, or a credit savings account's currency must be Turkish Lira. Such constraints regarding the domain are called *domain constraints*.

Although the next important thing in the banking domain is money, it has not been defined explicitly as an object. The currency[2] object with the help of the number object represents the concept of money. A currency object has only one currency attribute (C), as it is listed in Table 3.2. The possible values of the currency attributes (C) are, C0 for any indefinite foreign currency, C1 for Turkish Lira, C2 for Swiss Frank, C3 for US dollar, and C4 for German mark.

A transaction object serves as a transaction type parameter in the system. It is used for getting the operations history of the account. The transaction object has only one attribute which is the transaction type (T). The transaction type attribute may have four values, T1 for incoming transfers (put), T2 for withdrawals (draw), T3 for purchasing operations (buy), or T4 for selling operations (sell). An additional value T0, describes indefinite transaction types.

Another object in the banking domain is fund. Funds are the only type of investments that the system allows. They have only one attribute indicating their type, B1 or B2, for B one and B two funds [3]. They can be sold and bought and, by default, any operation on the funds is reflected to the investment account.

The last object used in the domain is the duration object, which represents the duration of the account's term. A duration object has only one attribute which is the duration type (TD).

---

[2]Note that, currency is a name which is used both for the object type and for the attribute.
[3]Note that although B1 and B2 are two possible fund type attribute values, whereas "B one" and "B two" are real banking domain object configurations.

3.2.2 Operations

The operations in the banking domain are represented as functions with arguments. The argument of the functions are instances of the banking domain objects. Although not implemented the functions are expected to be overloaded, so a function which takes arguments is expected to work in cases where some arguments are missing. The acquisition of the missing argument is assumed to be done through a database search or through a mixed-initiative dialogue, both of which are omitted in this work.

Table 3.4 lists the possible operations in the banking domain with their arguments.

Domain Operations	**Arguments**
Get total amount of the account	*account*, *currency*
List last *number* operations	*account*, *number*, *operations*
Learn term account interest rates	*account*
Learn term end date of an account	*account*
Learn total amount of interests in an account	*account*
Learn interest rates for different currencies	*currency*, *number*, *duration*
Learn amount of an account after the term end date	*account*, *currency*
Learn the amount of the funds in a portfolio	*account*, *fund*
Learn the value of funds	*account*, *fund*
Learn interest rates for repo	
Learn currency rates	*currency*
Buy funds	*account*, *number*, *fund*

Table 3.4: Table of Banking Domain Operations with their Arguments.

3.3 Development of the Grammar

3.3.1 Development Environment

Our regular grammar, mapping transcribed utterances into the banking domain operations, has been developed using Xerox's finite-state tool (XFST) and its finite state machine description language [16].

XFST's FSM definition language [2] supports all regular expressions (RE) and provides additional operations to develop complex grammars. Some regular expressions and their XFST counterparts are listed in Table 3.4.

Operation Name	**RE**	**XFST Counterpart**
Concatenation of L_1 and L_2	L_1L_2	$L_1\ L_2$
Union of L_1 and L_2	$L_1 + L_2$	$L_1 \vert L_2$
Kleene star of L_1	L_1*	L_1*
Kleene plus of L_1	L_1^+	L_1+
Parentheses	( )	[]
Round Parentheses of L_1	$L_1 + \epsilon$	(L_1)

Table 3.5: Basic XFST operations.

In the XFST's FSM definition language an additional facility, besides the operators introduced to define complex regular expressions, is the *define* command. The define command is used to give aliases to regular grammars. Previously defined *aliases* may be used like normal symbols in grammar design. In this way complex grammars can be developed easily by defining their rather simpler sub-parts and then combining them together. Quotes (" ") are used to explicitly distinguish between normal symbols and previously defined aliases. Any character sequence in quotes is interpreted as symbols by the XFST's FSM compiler.

3.3.2 Alphabet

The symbols of the regular grammar have been chosen as the lexical level representations of the morphemes. For our sentence level grammar we have extracted these multi-character symbols from the Turkish morphological analyzer written by Oflazer [19].

One of the reasons that we have chosen morphemes as our symbols instead of words themselves is their semantic importance, as discussed in Section 3.1.1. Another reason for our preference is that the Turkish morphology has complex morphographemics and morphotactics, which are very hard to describe implicitly in a sentence level grammar. A previous attempt of ours to develop a sentence level grammar by using surface forms of the words has made us conclude this.

Since Turkish characters are not supported on all platforms, we have used upper-case marking. In this scheme, the case information of the sentence is discarded and all characters are taken to be lower case, except for the characters 'C', 'G', 'I', 'O', 'U', 'S', which stand for Turkish characters 'ç', 'ğ', 'ı', 'ö', 'ü',and 'ş'.

Some symbols from our sentence level grammar are given below with their corresponding literal forms.

Grammar symbol:	portfOy	yatIrIm	"+mAK"
Literal form:	portföy	yatırım	*suffix* mek or mak, depending on morphographemy

3.3.3 Marking

Our sentence level regular grammar defines a transducer which marks the semantic structures on its input sentences. The upper language of this transducer is the set of sentences written in their lexical forms, whereas the lower language of the transducer is the set of sentences marked with their arguments and operation types. Figure 3.1 illustrates the input/output of the transducer.

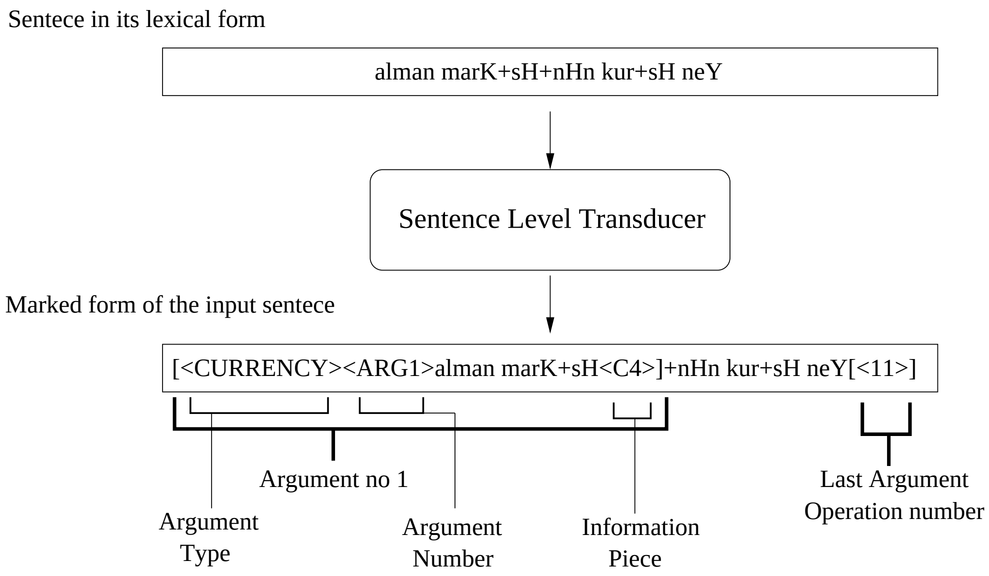


Figure 3.1: Input output description of the sentence level transducer.

In our annotation scheme, we mark two types of structures on the input sentence, *arguments* and *information pieces*.

An argument is a portion of a sentence with a particular semantic contribution. This contribution of the argument is employed by the domain operation which is implied by the sentence. Hence, the portions marked by arguments are the lexical representation of the domain objects, and the domain operation's *arguments*.

Arguments define and describe the attributes of the domain object with information pieces. Information pieces determine the attribute values of the domain objects. When occurring in an argument, the information pieces define the configuration of a domain object.

An additional functionality of the information pieces is to mark argument's *types* and *numbers*. These are the two information pieces that every argument must have. The argument's type defines the domain object's type, and the argument's number defines the order of the domain object in the argument list of the domain operation. The only exception to this nature of the arguments is the mandatory last argument of a sentence which marks the domain operation type. Sentences' last arguments have only a single special information piece which enumerates the operation meant by the sentence.

Information pieces on the sentence analysis are surrounded by less than '<' and greater than '>' signs and in this way they are distinguished from the lexical level words of the sentence. Except for the less than '<' and greater than '>' signs, information pieces indicating the objects attribute values are the same as in Table 3.3. For argument types, the information pieces are capital letter forms of the object types (e.g. FUND), and for argument numbers they are numbers added to the "ARG" abbreviation. Final information piece type is the information pieces used to indicate the domain operation numbers, which are just index numbers of the operations.

Instead of using the greater than and less than signs, arguments on the sentences are surrounded with brackets ("[" "]"), so that the marking structure on the sentence remains flat. The example sentence and its analysis in Figure 3.1 is quoted below.

```
Lexical form: alman marK+sH+nHn kur+sH neY
Analysis    : [<CURRENCY><ARG1>alman marK+sH<C4>]+nHn
              kur+sH neY[<11>]
```

Except its mandatory last argument, this sentence has only one argument which is the type of "CURRENCY". The information piece "ARG1" tells us that the argument is the first argument in the argument list of the domain operation. Finally the information piece "C4" remarks that the argument's currency attribute is the forth currency type, Mark. The last argument with its information piece categorizes this sentence as a sentence asked for learning the currency rates.

3.3.4 Design

We have defined our sentence level grammar by making use of the closure properties of the regular languages. The whole grammar is actually the union of many sentence patterns, which are the concatenation of simple constituent patterns. This whole set of patterns are aliased with the define operator of the XFST and so have been easily brought together.

In order to distinguish between the aliases and the language symbols we have employed the convention of using capital letters for the predefined aliases. The only exception to this convention is the information pieces which are alway surrounded with the quotation marks.

While aliasing, we tried to give proper names for the patterns such that a semantic category for the symbols can be easily realized. Below are three such patterns, used frequently in more complex patterns.

```
define AMOUNT [miktar|bakiye|tutar];
define DATE [gUn|zaman|tarih];
define VALUE [eder|deGer];
```

These simple patterns are combined to form more complex patterns such as verb forms to require information or verb forms to order someone to report something. Since the domain is strictly closed, possible morphemes to represent such concepts are highly limited, and thus can be listed easily.

The marking process described in the previous section is done by concatenating additional symbols to the pattern of morphemes describing the argument or the information piece. The upper symbols of these symbols are set to *epsilon*(0), so that they can be added to the sentence without consuming any upper symbol from the input pattern.

Some sample marking patterns are listed in below

```
define TL [tUrK BL lira "+sH"| tlE] [0:"<C1>"];
```

```
define ACCCREDIT [[kredi "+lH" [0:"<AC1>"]] |
                  [kredi "+ZHz" [0:"<AC2>"]]];
define SA [0:"["];
define EA [0:"]"];
```

Aliases “SE” and “EA” define patterns which are added to the sentence constituent whenever they represent an argument. Like these simple aliases, there are aliases which are used just for convenience in writing the grammar. Another alias used for convenience is “BL”, which defines the blank symbol (” ”) used as the separator between the words.

Although union and concatenation operations can be used to increase the diversity of a grammar, they are not enough for defining a grammar in precise. In our grammar, in addition to these operations we have also employed the subtraction operator provided by the Xerox FSM definition language. Longer patterns are constrained by the subtraction of smaller patterns, to enforce the domain rules on them. Below is a sentence pattern which demonstrates this schema.

```
define S18   [[[SA A2 ARG1 [CURRENCY - TL]EA] BL
             AMOUNTQUESTION BL [TL|lira] (BL ol "+mHS")
             (BL REPORT) F11] .-l. DC5];
```

In this sentence pattern, there are two types of subtractions. The first subtraction is the regular language subtraction, which excludes the Turkish Currency from the list of currencies. The second subtraction (.-l.) is a special *lower-language subtraction*, which is described in Section 2.2.2, applying the fifth domain constraint on the lower side of the pattern. This constraint pattern simply describes the existence of unidentified currency attribute, and thus its subtraction from the sentence pattern makes the sentence pattern refuse any input with such an attribute.

Using these design and marking schematics we have developed 19 sentence template patterns. When these patterns are unified, they cover more than 15 million instance sentences and can mark 12 banking operations. Features of the FSM built from this grammar are discussed in the next chapter.

3.4 Our System

Since the sentence level analysis grammar is written in the lexical level, we need another grammar to interface with the surface form of the sentences and produce *multi-character* lexical level morpheme sequences.

Although the morphological analysis FSM[4] developed by Oflazer [19] produces analyses of the words, the resulting analyses are sequences of single character symbols. For employing the resulting analyses we need another transducer grammar, which will map these single character symbols into their multi-character forms.

This tokenization grammar is compiled automatically from the alphabet of the sentence level grammar. For each symbol in the sentence level analyzer's alphabet, a pattern which maps the symbol's single-character form into its multi-character form is defined. Then the tokenization grammar is defined as the Kleene plus of the union of these patterns.

Rather than cascading the tokenization transducer and the morphological analysis transducers, we have chosen to compose them. In this way we ended up with a smaller transducer, a subset of the whole morphological analysis system, which analyzes only the words composed from our domain symbols. This small transducer is used as our word level grammar[5].

Figure 3.2 illustrates both the modules of the parsing system, and the flow of the information during parsing.

Any input sentence entered into the system is first chopped into its words.

[4]The morphological analysis FSM mentioned here can be acquired upon request by Kemal Oflazer for research only purposes.

[5]Besides this word level grammar, after evaluating our experimental results we have developed another word level analysis grammar. The necessity and development of this grammar will be discussed in Chapter 4.

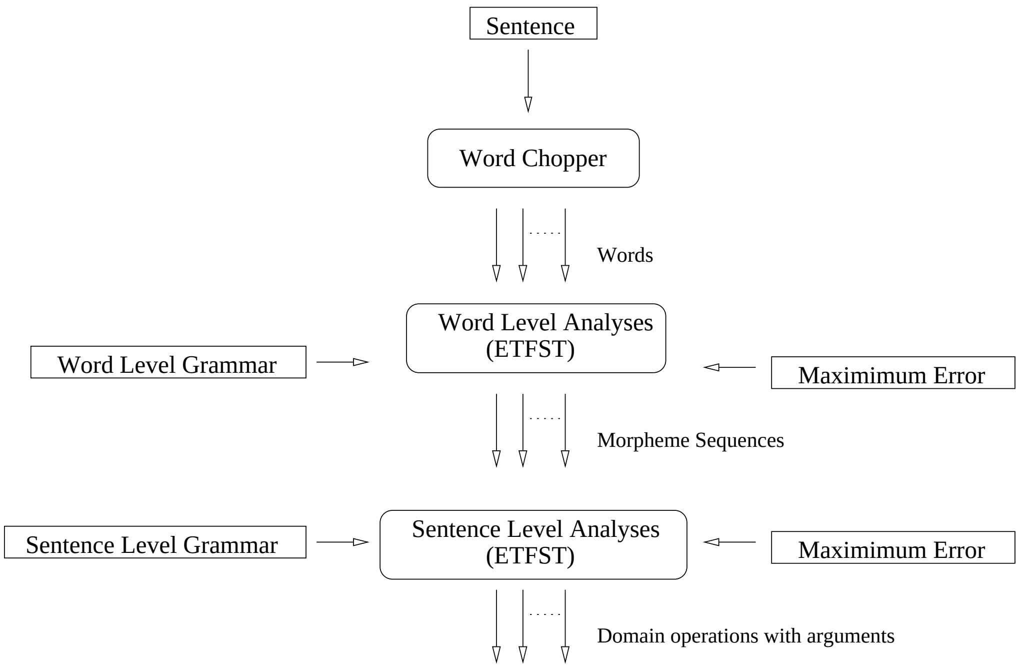


Figure 3.2: The whole parser module

Then these words are forwarded into level one analysis, to the word analysis transducer.

Since the word analysis transducer is executed with error tolerance, word level errors are corrected automatically as in the example below. In this sense the word level analysis can be considered as a form of spelling correction. The maximum error at this stage is relative to the length of the word. It is calculated as $(n/4)+1$ where n stands for the length of the word[6]. In this way the maximum error has been kept smaller for the shorter words.

```
correct input: geldi
output       : gel+DH
wrong   input: gelti
output       : gel+DH
```

[6]We have chosen number n/4 after our experiments.

The analyses of the words are combined into morpheme sequences by taking their cartesian products. Since some words may have more than one analysis taking this cartesian product is necessary. Each element of this product is a possible morpheme sequence with a normalized total word level error, which is the sum of all edit distances over the length of the sentence.

Finally, these morpheme sequences are forwarded to the sentence level transducer, and all of these sequences are evaluated regardless of their error rates received from the previous stage. The error tolerant execution of the transducer facilitates sentence level corrections, such as repetitions of words, word droppings and word replacements. In this way, regardless of their misleading form the sentences are marked correctly as illustrated below.

```
Correct        alman markI hesabImIn vade bitim zamanInda tutarI ne
Surface Form: kadar para olacak

Erroneous      alman markI hesabImIn vade bitim zamanInda lcutarI ne
Surface Form: kadar para yatIracak

Lexical        alman marK+sH hesab+Hm+nHn vade bitim zaman+sH+ndA
Analysis       tutar+sH neY kadar para yatIr+yAcAk

Analysis     : [<ACCOUNT><ARG1>alman marK+sH<C4> hesab]+Hm+nHn vade
               bitim zaman+sH+ndA tutar+sH neY kadar
               [<CURRENCY><ARG2>para] ol+yAcAk[<7>]


Correct
Surface Form: alman markI hesabImIn vade bitiS tarihi hangi tarih

Erroneous      alman markI hesabImIn vaapde bitiS tarihi hangi
Surface Form: tarih Unknovnvord

Lexical Form: alman marK+sH hesab+Hm+nHn vade bitiS tarih+sH hangi
```

```
              tarih

Analysis    : [<ACCOUNT><ARG1>alman marK+sH<C4> hesab]+Hm+nHn vade
              bitiS tarih+sH hangi tarih[<4>]


Correct       alman markI hesabImIn vade bitim gUnUnde miktarI ne
Surface Form: kadar alman markI olacak

Erroneous     alman markI hSesabmIn zaman bitim ngUUnde miktarI ne
Surface Form: kadar Unknovnvord alman markI olacak

Lexical Form: alman marK+sH hesab+Hm+nHn zaman bitim gUn+sH+ndA
              miktar+sH neY kadar  alman marK+sH ol+yAcAk

Analysis    : [<ACCOUNT><ARG1>alman marK+sH<C4> hesab]+Hm+nHn
              vade bitim gUn+sH+ndA miktar+sH neY kadar
              [<CURRENCY><ARG2>alman marK+sH<C4>] ol+yAcAk[<7>]
```

Chapter 4

Implementation and Evaluation

The error tolerant finite state transducers with their finite state machine counterparts, as discussed in the previous sections, were developed in the C++ programming language. The decision of programming language was made taking into account two important aspects of the work planned to be done: efficiency and maintainability.

A previous attempt for developing an error tolerant finite state tool system in the C language, had been abandoned after developing a prototype. Although that system proved to be very efficient, because of the poor facilities for maintaining programs written in C, it has been hard to modify for further use and experiment.

On the other hand, C++ language, being as efficient as C[1], facilitates both object oriented and generic programming, increasing maintainability. For these reasons, our Extended Finite State Tools (EFST[2]) library and the other applications based on this library, are developed in the C++ language.

[1] We haven't observed any difference in efficiency between our two implementations, written in C and C++ languages.

[2] License of the EFST library is GPL. Refer to www.gnu.org for further info on the license.

4.1 Implementation

The EFST library developed in this thesis, provides error tolerant finite state transducer and error tolerant finite state recognizer classes. The finite tools are compatible with the Xerox's XFST system[3]. A grammar developed in XFST can be exported as a text file and that text file can be easily used to construct finite state machines with the EFST library.

The two major client applications of the EFST library are *due* and *gendat*. We discuss them in the following subsections.

4.1.1 Due

Due, which stands for *dialogue utterance evaluator*, is the software implementing the parser module in Figure 1.2. It takes two finite state transducer grammars exported from XFST as its argument. The first argument is the word level analysis grammar file and the next one is the sentence analysis grammar file. The last input that *due* takes is the file of the input sentences.

As given in Figure 4.1, *due* implements the parser module through two cascaded error tolerant finite state transducers (fst): T_1 is the word level analyzer and T_2 is the sentence level analyzer, where G_1 and G_2 are the corresponding grammars.

Both word level and sentence level analysis are done by starting from initial zero tolerance and relaxing the tolerance until an analysis is found or tolerance reaches a strict maximum[4]. An upper bound for the error tolerance is necessary since the computation time of the system increases exponentially with the increasing tolerance as it is implied in Chapter 2.

[3]The print out grammar option in XFST is actually provided for prologue programs or human readers. In this way both descriptions are far from being efficient or sound. A perl script preprocessor is used to convert these forms into a more simple and sound form.

[4]This strict maximum in word level analysis is defined relative to the word length, as described in Section 3.4.

procedure $due(G_1, G_2, Sentences)$
// G_1 is word level analysis grammar, and G_2 is sentence level analysis grammar
$T_2 \leftarrow fst(G_2)$
$T_1 \leftarrow fst(G_1, T_2)$ // T_1 is cascaded with T_2
$Tape_{AllAnalysis} \leftarrow \emptyset$
for all $s \in Sentences$ **do**
  $Tape_{sentences} \leftarrow \epsilon$
  for all $word \in s$ **do**
    $Tape_{word} \leftarrow \emptyset$
    $t_{initial} \leftarrow 0$ // Initial tolerance set to zero
    $relaxingParse(t_{initial}, maxError_{WordLevel}, word, Tape_{word}, T_1)$
    // Tries different error tolerances for finding a parse
    $Tape_{sentences} \leftarrow Tape_{sentences} \bigotimes Tape_{word}$
  end for
  for all $sequence \in Tape_{sentences}$ **do**
    $Tape_{Analysis} \leftarrow \emptyset$
    $t_{initial} \leftarrow 0$
    $relaxingParse(t_{initial}, maxError_{SentenceLevel}, sequence, Tape_{Analysis}, T_2)$
    $Tape_{AllAnalysis} \leftarrow Tape_{AllAnalysis} \bigoplus Tape_{Analysis}$
  end for
end for
return $Tape_{AllAnalysis}$

Figure 4.1: Error tolerant parsing algorithm.

procedure $relaxingParse(tolerance, max, sequence, Tape, Transducer)$
while $tolerance < max$ and $Tape$ is empty **do**
  $Tape \leftarrow Transducer.parse(sequence, tolerance)$
  $tolerance \leftarrow tolerance + 1$
end while

Figure 4.2: Relaxing parse algorithm

The $\otimes$ operator, represents a cartesian product of *tapes*, which are the containers for *symbol sequences*. When a cartesian product of two tapes is taken, the resulting tape contains the all possible concatenations of the first tape's symbol sequences with the second tape's symbol sequences. With this operation at the word level analysis stage, all possible lexical interpretations of a sentence are generated easily.

Another operator $\oplus$ represents the union operation for the *tapes*. Through this operation the set of all possible analyses for all sentences is created at the end of the sentence level analysis stage.

As in the algorithm in Figure 4.1, *due* first creates the cascaded word and sentence level analysis ETFSTs, and for each input sentence generates the corresponding lexical analyses. Then, these analyses are processed by the sentence level analysis transducer, and collected in the results tape, $Tape_{AllAnalysis}$.

4.1.1.1 Selection of The Best Analysis

Although in Figure 4.1 the analyses for the sentences seem to lack any order, this is not the case in the implementation. By employing the additional total error field on the symbol sequences, for each analysis at the word level a total error is assigned to the corresponding symbol sequence. This total error, *word level error*, is the normalized total edit distance of all subsequences of the symbols. The example below exemplifies the word level error.

```
surface level: alman markI zamImIn vade bitiS hangi avih
lexical level: alman marK+sH zaman+Hm+nHn vade bitiS hangi tarih
error        : :0:0:0:0:0:2:2:2:0:0:0:0:0:0:0:2:[0.25]
```

The error sequence above tells us that the 7 word sentence in the example has been mapped to a lexical level representation with 16 lexical level symbols[5]. The consecutive three 2s on the error term indicates that the three corresponding

[5]Note that, the space used between the words is a lexical symbol in our grammar.

symbols, "zaman" "+Hm" "+nHn", have been parsed with 2 edit distance. The last error 2 says that the last symbol, "tarih", is derived from the surface form "avih" with 2 edit distances. The total normalized error 0.25 is calculated by dividing the total errors on the sequence (= 4) to the total number of symbols (= 16).

In the sentence level all lexical level sentences are analyzed without considering their error rates. As with the word level error, a *sentence level error* is calculated by normalizing the sentence level edit distance with the number of symbols.

All of the word level analyses with their sentence level analyses are placed into a priority queue, whose comparison function orders the word level analysis and sentence level analysis pairs according to their total errors received in the two stages. In this way, the analyses for a sentence have been ordered according the their sentence and word level error rates.

4.1.2 Datagen

Datagen is the noisy test data generation tool in our work. It generates ungrammatical sentences for the banking domain with random errors. The sentences generated by *datagen* are both in lexical level and in surface level deviated from their original forms. While lexical errors can be considered as syntactic errors, surface level errors can be thought of as spelling errors.

Datagen takes five arguments: the file name of the sentences written in their surface form, *lexical level errors*, *surface level error rate*, *surface level error*, and the prefix of the output files. This prefix is used in the three output file names with the extension "inf", "lex" and "sur". These output files are respectively the information file, lexical level deviated sentences file and both in lexical level and in surface level deviated sentences file.

Similar to *due*, *datagen* also employs the same word level analysis grammar and starts its process by parsing the surface form words into their lexical forms. After a lexical level representation of the sentence is reached, according to the value of lexical level errors, random morphemes are applied random insertions deletions and replacement. Then these morphemes are parsed back into their surface forms and *surface level error rate* number random words are chosen and applied random character insertions, deletions and additions such that the new form of the word has *surface level error* edit-distance from its original form.

The examples below demonstrates the evolution of correct input sentences into erroneous sentences in the *datagen* program. In these examples, the former number in the surface error is the number of words deviated, and the latter number is the magnitude of these deviations.

```
Correct Form   : B tipi likit fonun bu zamandaki deGeri kaC

After 1 Lexical
Error          : B likit fonun bu zamandaki deGeri kaC

After 2, 1 char
Surface Errors : B likCit fonun bu zamandaki deGerc kaC



Correct Form   : amerikan dolarI iCin aylIk faiz oranI kaC

After 2 Lexical
Errors         : amerikan dolarI iCin dOvizlik oranI kaC

After 1, 2 char
Surface Errors : amerikan dolr iCin dOvizlik oranI kaC
```

4.2 Evaluation

Although NLP is one of the oldest research areas in theoretical computer science, the evaluation methods for NLP systems are yet not well established. From many subfields of NLP, Information Extraction, Text Summarization, and Speech Recognition, many different evaluation metrics and methods have been proposed, but there is still no *de facto standard*.

Similarly in natural language parsing, there are many evaluation methods and evaluation metrics [8]. The most used and cited methods are corpus dependent methods, where the performance of a system is measured according to a previously parsed text.

Another respected evaluation method for natural language parsers, used especially in task-oriented natural language systems, is subjective evaluation [5]. In subjective evaluation, human subjects are allowed to interact freely with the systems and are required to accomplish specified tasks. After the trial, the users are interrogated or questioned about the system, and different measurements, such as *task completion rate* and *task completion time* are taken. Being a must test for any industrial application [25], this evaluation method is also employed for systems developed for academic research [4, 13]. The greatest problem in evaluating the systems with human subjects is the cost of the method. Although volunteer human subjects can be found for some attractive domains, it is difficult to do so for many other domains, such as ours. And the academic nature of our research forbids us from acquiring persons for testing our system[6].

In parallel, corpus oriented methods are also unsuited for evaluating our system. Apparently, there is no available corpus for the banking dialogue domain neither in Turkish nor in any other language. At the beginning of our research we had attempted many times to negotiate with the banks and develop such a corpus with their help, but we have been refused.

[6]In other words, hiring human subjects would be too expensive for us.

4.2.1 Our Evaluation Method

We have evaluated our grammar and parsing module in two ways, *Grammar Coverage* and *Robustness*.

In linguistically motivated works, grammar coverage is a frequently used evaluation method. Under this scheme, the quality of a grammar is determined according to the syntactic structures it covers and ungrammatical structures it rejects.

Since we only try to cover the simple linguistic phenomena required to identify banking domain operations, we could not directly use the grammar coverage metric. By adapting the definition of the grammar coverage, we have evaluated our grammar in terms of the properties of the minimum FSM representing it.

For evaluating the robustness of our parsing scheme, we have first generated in-grammar sentences and than inserted *linguistic noise* in these sentences (an approach inspired from [5]). The *datagen* program described in the previous section is used for this process. In this way, we have been able to measure the performance of our scheme for different error-levels.

4.2.2 Coverage

As mentioned in the previous chapter, the number of sentences covered by our grammar is 15384631, while the minimum FSM representing our grammar has 1343 states and 2343 transition arcs. The number of sentence patterns we have written to achieve this coverage is 19, and they themselves cover 12 banking domain operations.

In order to examine how the grammar and thus the FSM scales with newly added sentences, we have measured the changes on the arc, state, symbol, and path numbers with the increasing sentence pattern numbers.

Starting from the first sentence pattern we have extended the FSM's grammar

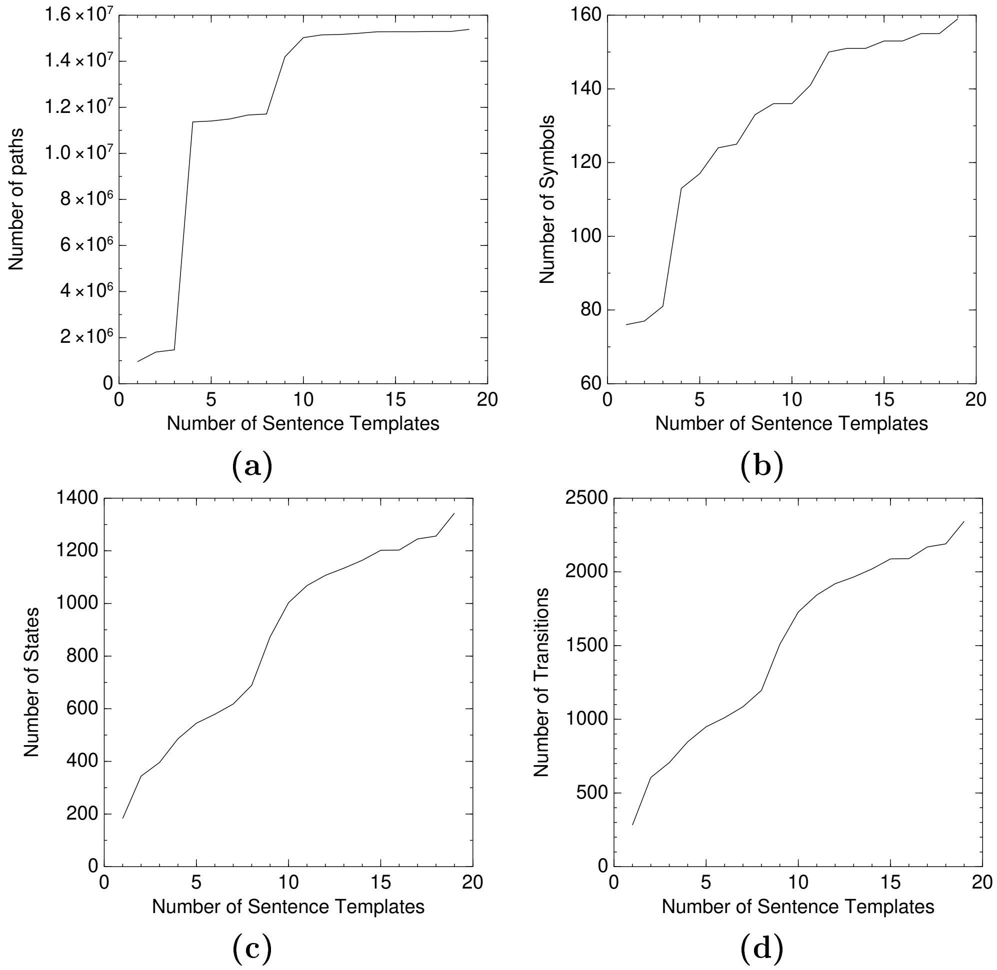


Figure 4.3: The change on the FSM's size with the increasing sentence patterns.

step by step. In each step, we have measured the size of the machine and then added the next sentence pattern into the definition of the whole grammar.

In Figure 4.3, the graphs **(a)** and **(b)** illustrates the number of paths in the FSM and the alphabet size of the FSM with the increasing sentence pattern number. The number of paths in the FSM are the number of sentences covered by the grammar. Graphs **(c)** and **(d)** illustrate the increase in the number of states and arcs with the increasing sentence pattern number. As is observable from the graphs in Figure 4.3 the size of the FSM increases linearly with the increasing sentence pattern number. This observation shows that our grammar will not generate a FSM which will blow up in size with the increasing sentence patterns. Therefore we can conclude that the FSM will scale with the increasing path numbers.

4.2.3 Robustness

FSMs can be used both as recognizers and as generators for the regular languages. By employing the sentence generation facility provided by XFST, from our banking domain grammar, we have generated a corpus of 728 sentences[7].

Since these sentences were in their lexical form, using the morphological analysis grammar [19] we have translated them back into their surface forms. After having a corpus of 728 random in-grammar sentences in their surface forms, we have generated noisy forms of this corpus using our *datagen* program.

For all 14 different noise combinations (listed in Table 4.1) we have prepared corresponding noisy corpora, and then tested our parsing scheme.

Table 4.1 presents the results for the 14 noisy corpora and one in-grammar corpus, where, LE, is the number of lexical level errors (syntactic errors), SER is the surface error rate, which indicates the number of words in which surface errors (spelling errors) are added, and SE is the surface error which is the number of character errors inserted on a chosen word (This number can be thought as the level of spelling error). CA is a shorthand form of correct analyses. It indicates the percentage of fully correct analyses, where operation type and arguments are exactly guessed. PCA is an abbreviation for partially correct analyses. Partially correct analyses are the analyses where the machine interprets the operation type correctly, but misses some arguments. Since each operation is assumed to be overloaded, we can expect to receive the missing arguments through database queries or asking questions to the client, therefore these types of analyses can be also counted as *satisfactory*. Below, is an example of a PCA. Although in the system's analysis, the correct attribute for the second argument could not been interpreted, it can be extracted using the context or a database query.

[7] XFST program does not generate unique random sentences. After the generation we have chosen only the unique sentences, hence the number.

Noise			CA %	PCA %	NA %	WA %	PWA %	t in msec
LE	SER	SE						
0	0	0	100.00	0.00	0.00	0.00	0.00	78.45
0	1	1	100.00	0.00	0.00	0.00	0.00	146.81
0	1	2	99.04	0.14	0.00	0.82	0.00	211.51
0	2	1	100.00	0.00	0.00	0.00	0.00	165.45
0	2	2	97.80	0.55	0.00	0.96	0.69	664.30
1	0	0	88.46	2.47	0.00	6.46	2.61	134.52
1	1	1	87.50	4.67	0.00	4.95	2.88	198.60
1	1	2	85.58	3.85	0.14	6.46	3.98	349.30
1	2	1	86.26	4.12	0.00	6.32	3.30	319.93
1	2	2	82.42	4.40	0.27	7.83	5.08	757.45
2	0	0	70.88	6.04	2.20	10.99	9.89	205.03
2	1	1	69.51	8.38	2.61	10.16	9.34	280.67
2	1	2	70.47	6.46	2.61	12.09	8.38	485.81
2	2	1	70.60	4.95	2.20	12.64	9.62	429.23
2	2	2	67.99	7.83	2.34	14.42	7.42	964.95

Table 4.1: Robustness test results of the Parser

```
Correct    [<ACCOUNT><ARG1>alman marK+sH<C4> hesab]+Hm+nHn
Analysis:  vade bitim  tarih+sH+ndA miktar+sH neY kadar
           [<CURRENCY><ARG2>marK<C4>] ol+yAcAk[<7>]

PCA      : [<ACCOUNT><ARG1>alman marK+sH<C4> hesab]+Hm+DA
            vade bitiS zaman+sH+ndA neY kadar
           [<CURRENCY><ARG2>para] biriK+mHS ol+yAcAk[<7>]
```

NA stands for the “no analyses”, and is the percentage of the sentences for which no analyses could be found within the maximum error tolerances. The maximum error tolerance in the word level analysis varies with word length as described in Section 3.4, and the maximum error in the sentence level analysis was set to 5.

WA is the percentage of wrong analyses, where the machine interprets the operation as something else. The example below illustrates such a case.

```
Correct   [<ACCOUNT><ARG1>cari hesab]+Hm+nHn faiz oran+sH
Analysis: neY[<3>]

Wrong
Analysis: [<ACCOUNT><ARG1>hesab]+Hm+nHn tutar+sH neY[<1>]
```

PWA are the partially wrong analyses, where the machine interprets the operation correctly but misinterprets the argument attributes and reads arguments in different configurations. The basic difference between a partially correct analysis and the partially wrong analysis is that the later one requires additional information correction steps. The example below demonstrates such a case.

```
Correct   Su gUn+ki [<FUND><ARG2>B tipi deGiSken fon<B2>]
Analysis: eder+sH kaJ[<9>]

PWA     : Su gUn+ki [<FUND><ARG2>B tipi likid fon<B1>]
          fiyat+sH kaJ[<9>]
```

Finally, the last column on the table is devoted to the average parsing time for a sentence in mili-seconds[8].

When we examine the table we see that the percentage of correct analyses drops with increasing noise. One exception to this observation are the transitions between the two consecutive rows where the *spelling error level* is replaced with the *number of spelling errors*. Table 4.2 lists these instances. The result of this observation is that the magnitude of misspellings are more important than their numbers in a sentence. A sentence with many simple errors on its words is easier to understand than a sentence with a single misleading word in it.

In Table 4.1, when we examine the change of time with increasing noise, in each three sections, we observe that the computation time increases dramatically with the increasing spelling error magnitude. From the results, it is obvious, how

[8]These results are taken on a conventional Athlon 800 personal computer with 96 MB of RAM.

Lexical Error	Surface Error Rate	Surface Error	Correct Analyses
0	1	2	99.04
0	2	1	100
1	2	1	86.26
1	2	2	82.42
2	1	2	70.47
2	2	1	70.60

Table 4.2: A sub-table from Table 4.1

the amount of surface level errors dominates the overall computation time. The explanation of this situation lies in the method we have employed to build our word level analysis grammar, which have been explained in Section 3.4.

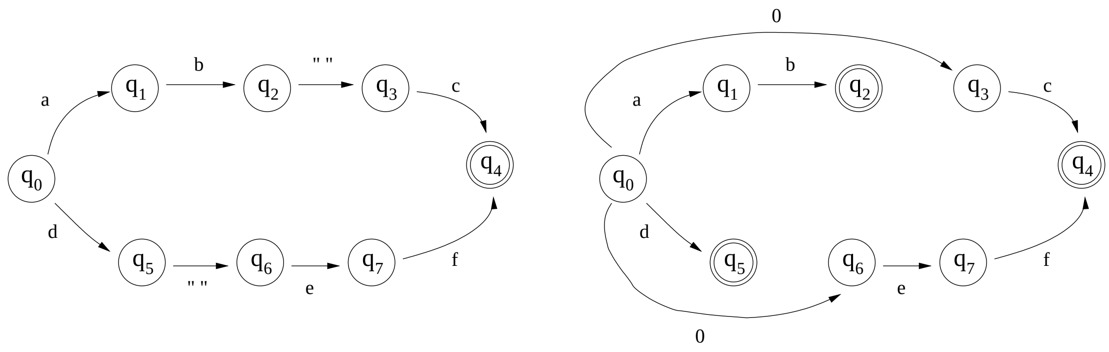
By taking the Kleene plus of our sentence level symbols' union, and composing it to the original morphological analyzer, we had restricted the analysis grammar such that its analyses include only the combinations of our morphemes. Since we did not actually define any restrictions for the order of these morphemes, the resulting grammar was including every valid combination of our morphemes, even though they never occur in our sentences.

Thus, with the increasing error magnitude the error tolerant word level analyzer generates too many analyses and word sequences for a given input sentence, and this ambiguity causes significant increase in the computation time.

4.2.3.1 Word Level 2 grammar

After making this observation we developed a method for generating a FSM accepting only the morpheme sequences in our sentence level grammar.

In this method, as exemplified in Figure 4.4, we simply delete the arcs with blank symbols on them, then add the states of these arcs to the final states set, and link the start state q_0 to the target states of these arcs with an ϵ transition. In this way we end up with a machine which accepts only the words of the sentences accepted in the previous grammar.



The machine accepting the sentences "ab c" and "d ef"

The machine accepting the words "ab" and "c" and "d" and "ef"

Figure 4.4: The morphotactics FSM generation method.

When applied to our sentence level grammar this method generates a FSM which accepts the actual morpheme sequences in our sentence level grammar. Thus this resulting machine represents the morphotactics in our domain.

In Table 4.3 we see the results taken with the new world level analysis system, WordLevel2. Since WordLevel2 generates far less word sequences than the former word level analyzer, the system is much faster in this configuration. Also with the decreasing importance of the spelling error amounts, lexical level errors become more important and create significant change in computation times.

Unfortunately, the quality of the results also drops with the execution time, since the WordLevel2 is very strict and can not analyze words with lexical level errors well.

4.2.3.2 Hybrid Model

In order to increase the analysis quality of the system, while being time-efficient, we have developed a simple hybrid model for the word level analysis.

In this scheme, every input word is processed by the WordLevel2 analyzer

Noise			CA %	PCA %	NA %	WA %	PWA %	t in msec
LE	SER	SE						
0	0	0	100.00	0.00	0.00	0.00	0.00	6.03
0	1	1	100.00	0.00	0.00	0.00	0.00	8.17
0	1	2	99.45	0.14	0.00	0.41	0.00	10.88
0	2	1	100.00	0.00	0.00	0.00	0.00	10.21
0	2	2	98.35	0.41	0.00	0.55	0.69	19.30
1	0	0	86.95	2.47	0.00	6.73	3.85	21.73
1	1	1	85.99	4.67	0.00	5.49	3.85	25.14
1	1	2	83.38	4.81	0.14	6.87	4.81	32.05
1	2	1	84.62	3.98	0.00	7.01	4.40	31.58
1	2	2	81.04	4.81	0.27	7.55	6.32	47.51
2	0	0	66.35	6.18	5.22	10.99	11.26	55.29
2	1	1	64.84	8.79	4.67	10.03	11.68	57.32
2	1	2	65.38	6.18	4.53	12.77	11.13	76.13
2	2	1	67.03	5.77	3.57	12.77	10.85	73.59
2	2	2	64.97	7.42	3.98	13.32	10.30	85.32

Table 4.3: Robustness test results with the WordLevel2.

unless Wordlevel2 can not generate any analysis. If the WordLevel2 analyzer can not generate any analyses for a word, then the default word level analyzer is called.

As seen in Table 4.4, with this configuration the analysis quality results are more comparable to our first configuration, while the computation times are significantly reduced.

4.2.3.3 Scaling

In order to test the scaling of system's robustness, we have developed 3 additional corpora, with smaller numbers of sentence patterns. These sentence patterns are chosen such that each pattern is a subset of its successor. In this way we tried to simulate the growth of the grammar with the newly added sentence patterns.

From these 3 corpora, which contain 394, 244, and 563 sentences, using the

Noise			CA %	PCA %	NA %	WA %	PWA %	t in msec
LE	SER	SE						
0	0	0	100.00	0.00	0.00	0.00	0.00	6.26
0	1	1	100.00	0.00	0.00	0.00	0.00	8.53
0	1	2	99.86	0.14	0.00	0.00	0.00	21.85
0	2	1	100.00	0.00	0.00	0.00	0.00	10.04
0	2	2	99.31	0.00	0.00	0.69	0.00	51.79
1	0	0	87.91	2.47	0.00	6.59	3.02	34.52
1	1	1	86.68	4.67	0.00	5.49	3.16	38.43
1	1	2	84.75	3.85	0.00	7.14	4.26	65.11
1	2	1	85.03	4.12	0.00	7.01	3.85	46.61
1	2	2	83.93	3.71	0.14	7.42	4.81	199.00
2	0	0	68.82	6.04	3.85	11.13	10.16	84.75
2	1	1	66.35	8.38	4.40	10.44	10.44	88.76
2	1	2	67.86	6.04	3.16	13.32	9.62	135.78
2	2	1	68.68	4.81	3.30	13.05	10.16	102.18
2	2	2	68.54	7.28	2.47	13.87	7.83	321.54

Table 4.4: Robustness test results with the Hybrid word level analysis.

datagen program we created 42 erroneous corpora for the 14 different error configurations, as we have done for the previous corpus. Then, by employing each word level analysis schema on each corpus we have gathered timing and analysis quality results.

Figure 4.5 illustrates the change in the average percentage of correct analyses for all error combinations, with the increasing sentence pattern numbers. From the graph we observe that the analysis quality of the system tends to drop slightly with the increasing number of sentence patterns. This drop in quality can be explained with the increase of the number of possible banking domain operations that can be annotated to an input sentence.

Figure 4.6 illustrates the change of average computation time of all error combinations with the increasing pattern numbers. As can be seen from the graph, except for the wordLevel1 analyses scheme, there is no significant change in the average computation time of the system.

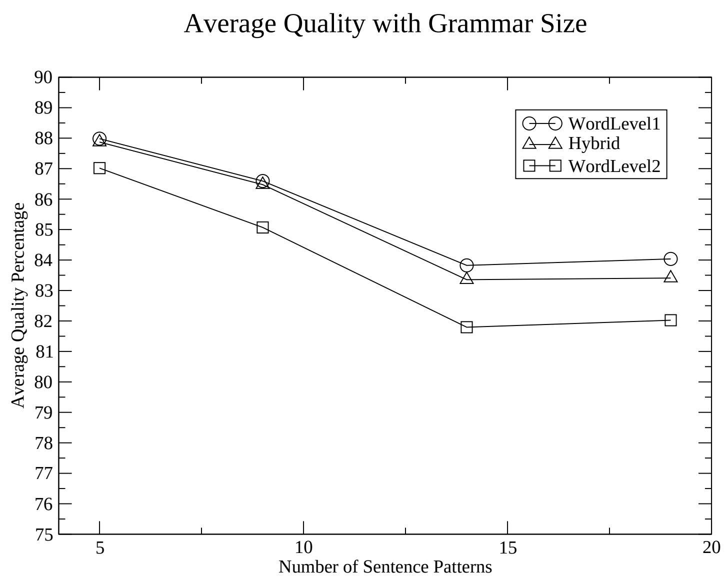


Figure 4.5: The change of average correct analyses percentage in different error levels with the sentence pattern numbers

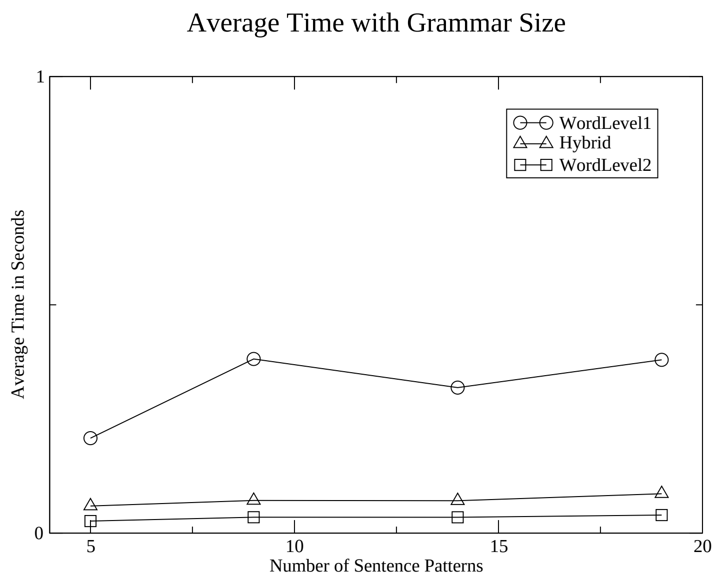


Figure 4.6: The change of average time in different error levels with the sentence pattern numbers

4.2.4 Final Remarks

Through our experiments we have observed that our methodology is both robust and time-efficient, while having a potential scalability. The hybrid word level analysis looks particularly promising since it combines both performance in quality of analyses and time-efficiency.

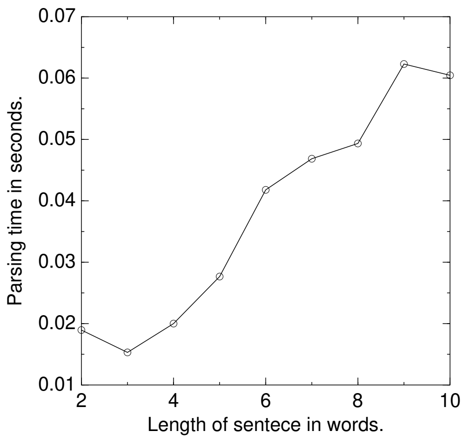


Figure 4.7: Change of the parsing time with the increasing sentence length.

In addition to these experiments, we conducted one final experiment to observe how the parsing time is effected by sentence length. We grouped the erroneous sentences from the 728 sentences corpus, according to their number of words. We then tested our parsing system with hybrid word level analysis for these sentence groups, which had 1 LE, 1 SER, and 1 SE. As illustrated in Figure 4.7, parsing time tends to increase linearly with the increasing sentence length.

Since the effect of the newly added sentence patterns on the whole grammar is not regular, the results we have obtained from our experiments are not exact, and are not sufficient for generalizing a formula for the efficiency of the system. Nevertheless these results are enough to disprove doubts regarding the scalability and efficiency of the finite state approach in closed domains.[9]

[9]The detailed test result tables can be found in Appendix A

Chapter 5

Conclusion

In this thesis, we have presented an error tolerant finite state parsing scheme and its partial grammar for a closed domain, namely Turkish Banking. Our experiments demonstrate that finite state based robust methods can be successfully employed to build practical parsers for closed domains.

After our initial experiments we have augmented our basic scheme with a hybrid word level analysis module. The resulting system can parse an ungrammatical sentence in less than half of a second with the 75 percentage accuracy. For an average level of errors the parsing time of the system drops to tenth of a second, while the quality of its results exceed 85 percentage.

These results in the quality of the analyses are comparable to the ones achieved in the literature of robust CFG parsing. Assuming that each order in the banking domain will be approved before executed, it can be concluded that the parser module is satisfying in quality.

Moreover, our experimental results, suggest that the system scales well with the increasing number of sentence patterns, both in terms of computation time, and storage (size of the FSM). In contrast to robust CFG parsing methods, in our parsing scheme, longer sentences do not effect the computation time dramatically, but only cause a linear increase.

Further evaluations are needed to prove the quality of the system. Instead of limiting our experiments to just the syntactic corpora, in the future we need to extend our grammar and conduct subjective evaluations on the system to conclude its efficiency for the closed domain, task oriented sentences.

Right now, we evaluate all edit distances in the same way and we simply give a constant 1 error point for them. But, in real life, in both word level and sentence level, each type of edit distance is not equal. As mentioned before, by taking into account that in spontaneous sentences word insertions are the most common type of errors, we can decrease the edit distance given for word insertions, such that the symbol insertions are tolerated more.

We could, for example, construct hand-written rules to arrange the edit distance among our grammar symbols. Such rules would enable us to tolerate the replacement of two similar suffixes more, in order to rank an analysis with such phenomena higher among the other analyses.

In our current system, we choose the best analysis for a sentence by the summation of its normalized word level and sentence level edit distances. In this scheme some correct analysis are missed, since they rank not first, but second or third. An alternative ranking scheme can be developed in order to increase the quality of these rankings, but right now we are satisfied with our yet simple ordering method.

As mentioned in the previous chapter, the system's time-efficiency is effected greatly by the increase in ambiguity coming from the word level analysis stage. Rather than representing these analyses with the cartesian product of words, we could use a *lattice* structure. By combining this representation with the local decisions taken in the sentence level grammar, we should be able to boost the speed of our model further[1].

[1]Even though local analyses may cause a decrease in the analysis quality, this lose may be compensated for with methods employing a portion of the time gained.

Bibliography

[1] Interactive telephone guide of İş Bankası.
Available at http://www.isbank.com.tr.

[2] MLTT: Syntax and semantics of regular expressions.
Available at http://www.xrce.xerox.com/research/mltt/fst/fssyntax.html.

[3] TeleWeb service site of the Yapı Kredi Bankası.
Available at http://www.teleweb.com.tr/teleweb/ykb.html.

[4] J. F. Allen, B. W. Miller, E. K. Ringger, and T. Sikorski. A robust system for natural spoken dialogue. In *Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 62–70, San Francisco, 1996. Morgan Kaufmann Publishers.

[5] M. Araki, T. Watanabe, and S. Doshita. Evaluating dialogue strategies for recovering from misunderstandings. In *IJCAI-97 workshop W7 Collaboration, Cooperation and Conflict in Dialogue Systems*, 1997.

[6] H. Aust and O. Schroer. An overview of the Philips dialog system. In *DARPA Broadcast News Transcription and Understanding Workshop*, February 1998.

[7] H. Bratt, J. Dowding, and K. HunickeSmith. The SRI telephone-based ATIS system. In *ARPA Spoken Language System Technology Workshop*, Austin, Texas, June 1995.

[8] T. Briscoe, J. Carroll, and A. Sanlippo. Parser evaluation: A survey and a new proposal. In *1st International Conference on Language Resources and Evaluation (LREC-98)*, Grenada, Spain, 1998.

[9] J. Dowding, J. M. Gawron, D. Appelt, and J. Bear. Gemini: A natural language system for spoken-language understanding. In *Proc. ARPA Human Language Technology Workshop '93*, pages 43–48, Princeton, NJ, 1994.

[10] J. Dowding, R. Moore, F. Andry, J. M. Gawron., and D. Moran. Combining linguistic and statistical knowledge sources in natural-language processing for ATIS. In *ARPA Spoken Language System Technology Workshop*, 1995.

[11] G. M. Ferguson, J. F. Allen, B. W. Miller, and E. K. Ringger. The design and implementation of the trains-96 system: A prototype mixed-initiative planning assistant. Technical report, Computer Science Dept., University of Rochester, October 1996.

[12] E. Gocmen, O. Sehitoglu, and C. Bozsahin. An outline of Turkish syntax. Technical report, METU Department of Computer Engineering, March 1995.

[13] D. R. Hipp. *Design and Development of Spoken Natural Language Dialog Parsing Systems*. PhD thesis, Duke University, June 1993.

[14] J. Hobbs, D. Appelt, J. Bear, D. Israel, M. Kameyama, M. Stickel, and M. Tyson. *Finite-State Language Processing*, chapter FASTUS: A Cascaded Finite-State Transducer for Extracting Information from Natural-Language Text, pages 383–406. MIT Press, 1997.

[15] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, Mass., 1979.

[16] L. Karttunen, T. Gaál, and A. Kempe. MLTT:Xerox finite-state tool. Technical report, Xerox, 1997.

[17] A. Lavie. GLR* – an efficient noise-skipping parsing algorithm for context free grammars. In *Third International Workshop on Parsing Technologies*, Tilburg, The Netherlands, August 1993.

[18] A. Lavie. *GLR*: A Robust Grammar-Focused parser for Spontaneously Spoken Language*. PhD thesis, Carnegie Mellon University, 1996.

[19] K. Oflazer. Two-level description of Turkish morphology. *Literary and Linguistic Computing*, 9(2), 1994.

[20] K. Oflazer. Error-tolerant finite state recognition with applications to morphological analysis and spelling correction. *Computational Linguistics*, 22(1), 1996.

[21] K. Oflazer and C. Güzey. Spelling correction in agglutinative languages. In *4th ACL Conference on Applied Natural Language Processing*, Stuttgart, Germany, 1994.

[22] G. Ralph. The NYU system for MUC-6 or where's the syntax. In *Sixth Message Understanding Conference (MUC6)*, pages 167–175, November 1995.

[23] C. P. Rosé. *Robust Interactive Dialogue Interpretation*. PhD thesis, Carnegie Mellon University, 1997.

[24] C. P. Rosé and A. Lavie. *Balancing Robustness and Efficiency*. Kluwer Academic Press, 2001.

[25] H. Strik, A. Russel, H. v.d. Heuvel, C. Cucchiarini, and L. Boves. A spoken dialogue system for the public transport information. In *Proceedings of the Department of Language and Speech*, volume 19, pages 129–142, 1996.

Appendix A

Evaluation Results

Noise			CA %	PCA %	NA %	WA %	PWA %	t in msec
LE	SER	SE						
0	0	0	100.00	0.00	0.00	0.00	0.00	58.51
0	1	1	100.00	0.00	0.00	0.00	0.00	90.78
0	1	2	99.29	0.36	0.00	0.36	0.00	235.93
0	2	1	99.82	0.00	0.00	0.18	0.00	200.23
0	2	2	97.16	1.07	0.00	1.78	0.00	702.45
1	0	0	87.39	4.62	0.00	6.75	1.24	122.61
1	1	1	87.03	3.91	0.00	6.04	3.02	182.84
1	1	2	87.03	3.55	0.00	6.75	2.66	275.45
1	2	1	85.97	4.62	0.00	6.57	2.84	266.82
1	2	2	82.24	4.80	0.71	8.35	3.91	591.24
2	0	0	71.76	8.70	1.78	11.72	6.04	180.85
2	1	1	72.11	8.35	1.95	10.66	6.93	256.75
2	1	2	65.19	7.64	4.26	14.21	8.70	309.82
2	2	1	72.11	5.51	2.66	11.55	8.17	318.83
2	2	2	66.43	7.64	4.26	13.32	8.35	726.82

Table A.1: Test results on 14 sentence patterns with WordLevel1.

Noise			CA %	PCA %	NA %	WA %	PWA %	t in msec
LE	SER	SE						
0	0	0	100.00	0.00	0.00	0.00	0.00	5.86
0	1	1	100.00	0.00	0.00	0.00	0.00	7.60
0	1	2	98.76	0.89	0.00	0.36	0.00	10.39
0	2	1	100.00	0.00	0.00	0.00	0.00	10.09
0	2	2	97.34	1.24	0.00	1.42	0.00	14.97
1	0	0	86.32	4.80	0.00	6.93	1.95	20.87
1	1	1	84.90	4.09	0.00	6.57	4.44	23.59
1	1	2	86.32	3.55	0.18	6.93	3.02	27.09
1	2	1	85.44	4.80	0.00	6.57	3.20	30.50
1	2	2	81.88	5.68	0.53	7.82	4.09	40.91
2	0	0	66.07	8.53	4.09	13.14	8.17	50.80
2	1	1	66.61	8.70	3.73	10.83	10.12	56.11
2	1	2	61.28	9.06	6.04	13.32	10.30	61.40
2	2	1	66.96	6.75	4.44	11.90	9.95	61.39
2	2	2	63.23	8.17	5.68	13.14	9.77	70.96

Table A.2: Test results on 14 sentence patterns with WordLevel2.

Noise			CA %	PCA %	NA %	WA %	PWA %	t in msec
LE	SER	SE						
0	0	0	100.00	0.00	0.00	0.00	0.00	6.16
0	1	1	100.00	0.00	0.00	0.00	0.00	7.98
0	1	2	99.47	0.00	0.00	0.53	0.00	15.44
0	2	1	100.00	0.00	0.00	0.00	0.00	10.25
0	2	2	98.58	0.18	0.00	1.24	0.00	40.57
1	0	0	86.68	4.62	0.00	7.10	1.60	35.08
1	1	1	85.79	3.73	0.00	6.93	3.55	37.74
1	1	2	86.86	3.02	0.00	7.46	2.66	58.08
1	2	1	85.97	4.62	0.00	6.75	2.66	43.36
1	2	2	84.01	4.62	0.00	7.64	3.73	169.82
2	0	0	67.67	8.70	3.55	12.61	7.46	77.62
2	1	1	69.63	7.99	3.20	10.83	8.35	85.22
2	1	2	65.36	7.99	3.73	13.85	9.06	133.20
2	2	1	69.45	5.68	3.55	12.61	8.70	90.32
2	2	2	67.50	7.28	3.91	13.68	7.64	189.57

Table A.3: Test results on 14 sentence patterns with Hybrid.

Noise			CA %	PCA %	NA %	WA %	PWA %	t in msec
LE	SER	SE						
0	0	0	100.00	0.00	0.00	0.00	0.00	96.23
0	1	1	100.00	0.00	0.00	0.00	0.00	138.61
0	1	2	99.59	0.00	0.00	0.41	0.00	312.09
0	2	1	100.00	0.00	0.00	0.00	0.00	157.30
0	2	2	97.95	1.23	0.00	0.41	0.41	885.66
1	0	0	86.07	5.33	0.00	8.20	0.41	137.50
1	1	1	91.39	3.69	0.41	3.69	0.82	304.75
1	1	2	87.30	2.87	0.00	9.02	0.82	413.57
1	2	1	90.57	2.87	0.00	5.74	0.82	305.41
1	2	2	88.11	2.87	0.00	8.20	0.82	590.98
2	0	0	71.72	5.74	2.46	16.39	3.69	251.19
2	1	1	77.87	5.33	1.64	13.52	1.64	254.84
2	1	2	72.54	9.43	2.46	11.07	4.51	447.62
2	2	1	74.18	7.38	2.46	13.11	2.87	422.87
2	2	2	75.00	6.15	2.87	11.07	4.92	716.93

Table A.4: Test results on 9 sentence patterns with WordLevel1

Noise			CA %	PCA %	NA %	WA %	PWA %	t in msec
LE	SER	SE						
0	0	0	100.00	0.00	0.00	0.00	0.00	6.19
0	1	1	100.00	0.00	0.00	0.00	0.00	8.36
0	1	2	99.59	0.00	0.00	0.41	0.00	10.49
0	2	1	100.00	0.00	0.00	0.00	0.00	9.22
0	2	2	97.95	1.23	0.00	0.41	0.41	16.43
1	0	0	85.25	4.92	0.00	9.43	0.41	19.63
1	1	1	91.80	3.69	0.00	3.69	0.82	24.67
1	1	2	86.89	3.69	0.41	8.20	0.82	33.24
1	2	1	89.75	2.87	0.00	6.56	0.82	28.44
1	2	2	86.48	2.87	0.00	9.43	1.23	48.93
2	0	0	69.67	6.97	3.69	15.57	4.10	47.95
2	1	1	72.54	6.15	5.33	14.34	1.64	46.48
2	1	2	68.44	9.43	4.92	12.30	4.92	56.52
2	2	1	70.08	9.43	3.69	13.11	3.69	55.00
2	2	2	72.54	6.56	2.87	12.30	5.74	83.11

Table A.5: Test results on 9 sentence patterns with WordLevel2

Noise			CA %	PCA %	NA %	WA %	PWA %	t in msec
LE	SER	SE						
0	0	0	100.00	0.00	0.00	0.00	0.00	6.72
0	1	1	100.00	0.00	0.00	0.00	0.00	8.85
0	1	2	100.00	0.00	0.00	0.00	0.00	14.39
0	2	1	100.00	0.00	0.00	0.00	0.00	9.67
0	2	2	100.00	0.00	0.00	0.00	0.00	37.09
1	0	0	85.66	4.92	0.00	9.02	0.41	29.22
1	1	1	91.80	3.69	0.00	3.69	0.82	38.89
1	1	2	87.30	2.46	0.41	9.02	0.82	56.11
1	2	1	90.57	2.87	0.00	5.74	0.82	45.16
1	2	2	88.52	2.87	0.00	7.38	1.23	160.98
2	0	0	71.72	5.74	2.87	16.39	3.28	83.28
2	1	1	75.00	5.33	4.10	13.93	1.64	75.66
2	1	2	71.31	9.02	2.87	12.30	4.51	119.75
2	2	1	73.36	7.79	2.46	13.11	3.28	89.30
2	2	2	75.41	6.15	2.87	11.48	4.10	233.07

Table A.6: Test results on 9 sentence patterns with Hybrid

Noise			CA %	PCA %	NA %	WA %	PWA %	t in msec
LE	SER	SE						
0	0	0	100.00	0.00	0.00	0.00	0.00	33.96
0	1	1	100.00	0.00	0.00	0.00	0.00	51.22
0	1	2	99.24	0.25	0.00	0.51	0.00	103.07
0	2	1	100.00	0.00	0.00	0.00	0.00	58.65
0	2	2	98.98	0.00	0.00	1.02	0.00	315.33
1	0	0	88.58	5.08	0.00	6.09	0.25	90.74
1	1	1	89.85	3.55	0.00	6.09	0.51	162.84
1	1	2	88.32	4.82	0.51	6.09	0.25	218.96
1	2	1	91.37	3.81	0.00	4.57	0.25	146.83
1	2	2	88.58	5.33	0.76	4.31	1.02	331.29
2	0	0	74.37	10.66	2.79	11.42	0.76	164.90
2	1	1	72.59	9.64	3.05	13.71	1.02	197.74
2	1	2	67.26	10.15	6.35	14.72	1.52	286.37
2	2	1	74.11	8.38	4.57	10.15	2.79	204.37
2	2	2	67.26	9.39	6.85	15.23	1.27	506.65

Table A.7: Test results on 5 sentence patterns with WordLevel1

Noise			CA %	PCA %	NA %	WA %	PWA %	t in msec
LE	SER	SE						
0	0	0	100.00	0.00	0.00	0.00	0.00	5.41
0	1	1	100.00	0.00	0.00	0.00	0.00	6.62
0	1	2	98.98	0.25	0.00	0.76	0.00	8.81
0	2	1	100.00	0.00	0.00	0.00	0.00	7.92
0	2	2	99.24	0.25	0.00	0.51	0.00	13.27
1	0	0	88.32	5.33	0.00	6.09	0.25	17.34
1	1	1	89.34	4.31	0.00	6.09	0.25	21.14
1	1	2	88.32	5.84	0.51	5.08	0.25	24.37
1	2	1	91.12	4.57	0.00	4.06	0.25	21.47
1	2	2	89.09	5.58	0.51	4.06	0.76	26.14
2	0	0	70.05	11.93	5.08	11.42	1.52	37.89
2	1	1	70.30	10.91	4.57	12.18	2.03	39.44
2	1	2	65.48	10.66	8.12	13.71	2.03	42.23
2	2	1	70.56	9.14	6.35	10.41	3.55	41.37
2	2	2	66.50	9.64	7.87	13.96	2.03	51.90

Table A.8: Test results on 5 sentence patterns with WordLevel2

Noise			CA %	PCA %	NA %	WA %	PWA %	t in msec
LE	SER	SE						
0	0	0	100.00	0.00	0.00	0.00	0.00	5.66
0	1	1	100.00	0.00	0.00	0.00	0.00	6.88
0	1	2	99.75	0.00	0.00	0.25	0.00	12.03
0	2	1	100.00	0.00	0.00	0.00	0.00	8.15
0	2	2	99.49	0.00	0.00	0.51	0.00	20.86
1	0	0	88.58	5.08	0.00	6.09	0.25	29.24
1	1	1	90.10	3.55	0.00	6.09	0.25	36.55
1	1	2	88.58	4.57	0.25	6.35	0.25	48.30
1	2	1	91.37	3.81	0.00	4.57	0.25	36.73
1	2	2	89.34	5.08	0.25	4.57	0.76	83.02
2	0	0	71.83	10.41	4.31	12.44	1.02	85.36
2	1	1	70.81	9.90	4.06	13.71	1.52	96.14
2	1	2	67.51	10.41	5.58	14.72	1.78	106.17
2	2	1	72.84	8.12	5.08	10.66	3.30	80.38
2	2	2	68.78	8.12	5.84	15.99	1.27	161.27

Table A.9: Test results on 5 sentence patterns with Hybrid

Appendix B

Grammar

Below are some patterns used in the sentence level analysis grammar.

```
#********************************************************
#    Base Definitions
#********************************************************

define BL " "; #can be replaced with 0

define SA [0:"["];  #begin argument

define EA [0:"]"]; #end argumenet

define SP 0:" "; #seperator of the arguments and etc.

define ARG1 0:"<ARG1>"; #first argument

define ARG2 0:"<ARG2>"; #second argument

define ARG3 0:"<ARG3>"; #third argument
```

```
#**********************************************************
#    Function Type Definitions
#**********************************************************

define F1 [0:"[<1>]"];


#**********************************************************
#    Argument Type Definitions
#**********************************************************

define A1 0:"<ACCOUNT>";

define A2 0:"<CURRENCY>";


#**********************************************************
#    Semantic Definitions
#**********************************************************

define MANY [kaJ];

define MUCH [neY BL kadar];

define WHAT [neY];

define AMOUNTQUESTION [MANY|MUCH];

define AMOUNT [miktar|bakiye|tutar];

# foreign currency
define FC [dOviz] [0:"<CO>"];
```

```
define TL [tUrK BL lira "+sH"| tlE] [0:"<C1>"];

define FR  [fransIz BL frank "+sH"|frank] [0:"<C2>"];

define USD [amerikan BL dolar "+sH"|dolar] [0:"<C3>"];

define DM  [alman BL  marK "+sH"|marK] [0:"<C4>"];

define CURRENCY [TL|FR|USD|DM|FC];

define MONEY [para|CURRENCY];

define TERM [[vade "+lH"  [0:"<AT1>"]] | [vade "+ZHz" [0:"<AT2>"]]];

define ACCCREDIT [[kredi "+lH" [0:"<AC1>"]] |[kredi "+ZHz"
                 [0:"<AC2>"]]];

define ACCLOAN [tUket "+yHcH" BL kredi "+sH" [0:"<AL>"]];

define ACCSAVINGS [mevduat|tevdiat];

define ACCCURRENT [cari];

define ACCINVESTMENT [yatIrIm [0:"<AI>"]];

define ACCTYPE [ACCINVESTMENT|ACCCURRENT|ACCLOAN];

define ACCOUNT [[ ([ACCTYPE BL]|[([ACCCREDIT|TERM] BL)
                 ([CURRENCY] BL)
               (ACCSAVINGS BL)]) hesab ] .-u. DC1];

define PORTFOLIO [portfOy [0:"<AI>"]];
```

```
# Verbs used in present
define EXISTVERBPRES [bulun |dur |gOzUK];

# Verbs used in past
define EXISTVERBPASS [biriK | ol];

# Verbs with tenses
define EXISTVERBINTENSE [[var | [EXISTVERBPRES] ["+mAktA" | "+Hyor"]] |
                        EXISTVERBPASS "+mHS"];

define EXISTVERBACC [[[EXISTVERBINTENSE BL ol]| EXISTVERBPRES]
                    "+DHk" "+Hn" "+yH"];

#FIXME, ben+yA -> bene in morpheme.fst out of date...
define ME [ban];

define INFOVERB [[bil "+DHr"] | sOyle | Ozetle];

#adhoc constraints for dealing with morpheme.fst's failure.

define CFIXME [[sOyle | Ozetle] => _ [["+yA" bil "+Hr"]| "+r"]];
define CFIXME2 [$[bil "+DHr" "+r"]];

define INFOREQ [[[(ME "+yA" BL) INFOVERB
                ([["+yA" bil "+Hr"] | ["+Hr"| "+r"]] BL mi "+ZHn" (Hz))]
                & CFIXME] .-u. CFIXME2];

define IWISH [iste "+Hyor" "+yHm"];

define INFOWISH [[[(ME "+yA" BL) INFOVERB "+mA" "+Hn" (Hz) "+yH"]|
                OGren "+mAK"]
                BL IWISH];
```

```
define REPORT [INFOREQ|INFOWISH];



#*******************************************************
#    Sentences
#*******************************************************

#Domain Cons 2 currency type should be same

define DC2 [[$TL $[CURRENCY - TL]] |
           [$USD   $[CURRENCY - USD]]|
           [$DM    $[CURRENCY - DM ]]|
           [$FR    $[CURRENCY - FR ]]|
           [$dOviz $[TL]           ]];


#Domain Cons 3 dOviz types cannot be used with ACCINVESTMENT ACCLOAN;

define DC3 [ $[ACCINVESTMENT|ACCLOAN] $[CURRENCY - TL]|
            $[CURRENCY - TL] $[ACCINVESTMENT|ACCLOAN]];


# Hesap Bakiyesi Isteme Tipleri (requesting aqount amount info)

define S1 [[ SA A1 ARG1 [ACCOUNT|PORTFOLIO] EA
          "+Hm" "+DA" BL AMOUNTQUESTION BL
          SA A2 ARG2 MONEY EA BL
          [EXISTVERBINTENSE (BL REPORT)|  EXISTVERBACC BL REPORT]
          F1] .-u. DC2 .-u. DC3 ];


define S2 [[AMOUNTQUESTION BL
```

```
           SA A2 ARG2 MONEY EA BL
           EXISTVERBINTENSE BL
           SA A1 ARG1 [ACCOUNT | PORTFOLIO] EA
           "+Hm" "+DA"
           (BL REPORT) F1] .-u. DC2 .-u. DC3];

define S3 [ SA A1 ARG1 [ACCOUNT | PORTFOLIO] EA
            "+Hm" "+nHn" BL
            AMOUNT "+sH" BL
            [AMOUNTQUESTION | WHAT] ("+DHr")
            (BL REPORT)
            F1
          ];
```

Appendix C

Sample Sentence Analyses

Below are lexical and sentence level analyses of erroneous sentences with their lexical level and sentence level normalized errors.

```
B tipi likit fvnun Simdiki deGeri kaC

Lexical Level: B tipi likid fon+nHn Simdi+ki deGer+sH kaJ
Lex. Error   : [0.0625]
Analysis     : [<FUND><ARG2>B tipi likid fon<B1>]+nHn Simdi+ki
               deGer+sH kaJ[<9>]
Anal. Error  : [0]


Su zamandaki on deGeri kaC

Lexical Level: Su zaman+DA+ki fon deGer+sH kaJ
Lex. Error   : [0.0833333]
Analysis     : Su zaman+DA+ki [<FUND><ARG2>fon] deGer+sH kaJ[<9>]
Anal. Error  : [0]
```

```
clmn markI tevdiat hesabImIn faiz oranI ne

Lexical Level: alman marK+sH tevdiat hesab+Hm+nHn faiz oran+sH neY
Lex. Error   : [0.117647]
Analysis     : [<ACCOUNT><ARG1>alman marK+sH<C4> tevdiat hesab]+Hm+nHn
               faiz oran+sH neY[<3>]
Anal. Error  : [0]
```

```
amerikan dolarI hesabImIn vade bitim tarihinde bakiyesi kaC dGlaa
olacak

Lexical Level: amerikan dolar+sH hesab+Hm+nHn vade bitim tarih+sH+ndA
bakiye+sH kaJ dolar ol+yAcAk
Lex. Error   : [0.0769231]
Analysis     : [<ACCOUNT><ARG1>amerikan dolar+sH<C3> hesab]+Hm+nHn
               vade bitim tarih+sH+ndA bakiye+sH kaJ
               [<CURRENCY><ARG2>dolar<C3>] ol+yAcAk[<7>]
Anal. Error  : [0]
```

```
alman markI yatIr number aylIk faiz oranI ne

Lexical Level: alman marK+sH yatIrIm number ay+lHk faiz oran+sH neY
Lex. Error   : [0.111111]
Analysis     : [<CURRENCY><ARG1>alman marK+sH<C4>] iCin
               [<NUMBER><ARG2>number]
               [<TERMDURATION><ARG3>ay<TD1>]+lHk faiz oran+sH neY[<6>]
Anal. Error  : [0.030303]
```

```
alman markI mevduat Unknovnvord hesabImdaki son number satma iSlemini
bana bildir
```

```
Lexical Level: alman marK+sH mevduat  hesab+Hm+DA+ki son number sat+mA
               iSlem+sH+nH ban+yA bil+DHr
Lex. Error   : [0]
Analysis     : [<ACCOUNT><ARG1>alman marK+sH<C4> mevduat
               hesab]+Hm+DA+ki son [<NUMBER><ARG2>number]
               [<TRANSACTION><ARG3>sat<T4>+mA iSlem+sH]+nH ban+yA
               bil+DHr[<2>]
Anal. Error  : [0.0232558]
```

```
alman markI mevduat vzade bitiS tarihinde miktarI ne olacak
```

```
Lexical Level: alman marK+sH mevduat vade bitiS tarih+sH+ndA miktar+sH
               neY ol+yAcAk
Lex. Error   : [0.0454545]
Analysis     : [<ACCOUNT><ARG1>alman marK+sH<C4> hesab]+Hm+nHn vade
               bitiS tarih+sH+ndA miktar+sH neY ol+yAcAk[<7>]
Anal. Error  : [0.1]
```

```
amerikan tarihi heuabImIn faizi ne
```

```
Lexical Level: amerikan tarih+sH hesab+Hm+nHn faiz+sH neY
Lex. Error   : [0.0769231]
Analysis     : [<ACCOUNT><ARG1>amerikan dolar+sH<C3> hesab]+Hm+nHn
               faiz+sH neY[<3>]
Anal. Error  : [0.0526316]
```

```
B tipi daeGiSken fonun zampanki fiyatI ne
```

```
Lexical Level: B tipi deGiSken fon+nHn zaman+DA+ki fiyat+sH neY
```

```
Lex. Error   : [0.235294]
Analysis     : [<FUND><ARG2>B tipi deGiSken fon<B2>]+nHn bu
               zaman+DA+ki fiyat+sH neY[<9>]
Anal. Error  : [0.08]

amerikan tipi likit kfgon ederi kaC

Lexical Level: amerikan tipi likid fon eder+sH kaJ
Lex. Error   : [0.166667]
Analysis     : [<FUND><ARG2>B tipi likid fon<B1>] eder+sH kaJ[<9>]
Anal. Error  : [0.0555556]


ofen bir ederi nac

Lexical Level: fon bir eder+sH neY
Lex. Error   : [0.5]
Analysis     : [<FUND><ARG2>fon bir<B1>] eder+sH neY[<9>]
Anal. Error  : [0]

lvman markI mevduat hesabImIn vade bitiS zamanInda tutarI kaC var
markI olcaS

Lexical Level: alman marK+sH mevduat hesab+Hm+nHn vade bitiS
               zaman+sH+ndA tutar+sH kaJ var marK+sH ol+yAcAk
Lex. Error   : [0.129032]
Analysis     : [<ACCOUNT><ARG1>alman marK+sH<C4> mevduat hesab]+Hm+nHn
               vade bitiS zaman+sH+ndA tutar+sH kaJ
               [<CURRENCY><ARG2>alman marK+sH<C4>] ol+yAcAk[<7>]
Anal. Error  : [0.0238095]


dOviz hesabImIn vade likit np
```

```
Lexical Level: dOviz hesab+Hm+nHn vade likid neY
Lex. Error   : [0.0909091]
Analysis     : [<ACCOUNT><ARG1>dOviz<C0> hesab]+Hm+nHn faiz yUzde+sH
               neY[<3>]
Anal. Error  : [0.166667]


amerikan dolarI tevdiat hesabImIn vade portfOy gUnUnde bakiyesi
Unknovnvord kaBC amerikan dolarI olacak

Lexical Level: amerikan dolar+sH tevdiat hesab+Hm+nHn vade portfOy
               gUn+sH+ndA bakiye+sH  kaJ amerikan dolar+sH ol+yAcAk
Lex. Error   : [0.03125]
Analysis     : [<ACCOUNT><ARG1>amerikan dolar+sH<C3> tevdiat
               hesab]+Hm+nHn vade bitim gUn+sH+ndA bakiye+sH kaJ
               [<CURRENCY><ARG2>amerikan dolar+sH<C3>] ol+yAcAk[<7>]
Anal. Error  : [0.047619]


B tipi deGGIken Unknovnvord fonun fiyatI ne

Lexical Level: B tipi deGiSken  fon+nHn fiyat+sH neY
Lex. Error   : [0.142857]
Analysis     : [<FUND><ARG2>B tipi deGiSken fon<B2>]+nHn fiyat+sH
               neY[<9>]
Anal. Error  : [0.0526316]


B deGiSken fonun bugUnki fayaItI Unknovnvord ne

Lexical Level: B deGiSken fon+nHn bugUn+ki fiyat+sH  neY
Lex. Error   : [0.133333]
```

```
Analysis      : [<FUND><ARG2>B tipi deGiSken fon<B2>]+nHn bugUn+ki
                fiyat+sH neY[<9>]
Anal. Error   : [0.136364]


B tipi oS Unknovnvord satIn iml

Lexical Level: B tipi ol  satIn al
Lex. Error    : [0.3]
Analysis      : [<FUND><ARG3>B tipi likid fon<B1>] satIn al[<12>]
Anal. Error   : [0.117647]


iki gUnki B ipci adeGiSkmn fon sonu kaC

Lexical Level: iki gUn+ki B tipi deGiSken fon son kaJ
Lex. Error    : [0.3125]
Analysis      : bu gUn+ki [<FUND><ARG2>B tipi deGiSken fon<B2>]
                fiyat+sH kaJ[<9>]
Anal. Error   : [0.130435]
```