EXAMPLE BASED MACHINE TRANSLATION WITH TYPE ASSOCIATED TRANSLATION EXAMPLES

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By
Hande DOĞAN

January, 2007

ABSTRACT

EXAMPLE BASED MACHINE TRANSLATION WITH TYPE ASSOCIATED TRANSLATION EXAMPLES

Hande DOĞAN
M.S. in Computer Engineering
Supervisor: Assist. Prof. Dr. İlyas Çiçekli
January, 2007

Example based machine translation is a translation technique that leans on machine learning paradigm. This technique had been modeled by the learning process as: a man is given short and simple sentences in language A with their correspondences in language B; he memorizes these pairs and then becomes able to translate new sentences via these pairs in the memory. In our system the translation pairs are kept as translation templates. A translation template is induced from given two translation examples by replacing differing parts in these examples by variables. A variable replacing a difference that consists of two differing parts (one from the first example, and the other one from the second example) is a generalization of those two differing parts and these variables are supported with part-of-speech tag information in order to deteriorate incorrect translations. After the learning phase, translation is achieved by finding the appropriate template(s) and replacing the variables.

Keywords: Example Based Machine Translation, Type Associated Translation Template Induction, Machine Learning.

ÖZET

TİP DESTEKLİ ÇEVİRİ KALIPLARI İLE ÖRNEK TABANLI OTOMATİK ÇEVİRİ

Hande DOĞAN
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Yard. Doç. Dr. İlyas Çiçekli
Ocak, 2007

Örnek tabanlı otomatik çeviri sistemleri makine öğrenmesine dayanan yöntemlerden yararlanarak çeviri yaparlar. Bu çeviri süreci şöyle özetlenebilir: Bir insan birinci dilde basit ve kısa cümleleri ikinci dildeki karşılıkları ile birlikte ezberledikten sonra muhakeme ile yeni verilen cümleleri daha önceden öğrendikleri aracılığıyla çevirebilir. Bizim sistemimizde çeviri örnekleri çeviri kalıpları olarak tutulmaktadır. Çeviri kalıpları, iki çeviri örneğinden, örneklerin farklı kısımlarınının yerine değişkenler koyularak öğrenilmektedir. Değişik kısımların yerine geçen değişkenler, çeviri örneklerinin herbirinden gelen değişik kısımları genelleştirmektedir. Bu sistemde değişkenlerin genelleştirdikleri kısımların tip bilgilerini de çeviri kalıplarının yapısına ekleyerek yanlış çeviri sonuçlarının sistemce üretilmesinin engellenmesi amaçlanmaktadır.

Anahtar sözcükler: Örnek Tabanlı Otomatik Çeviri, Tip Destekli Çeviri Kalıpları, Makine Öğrenmesi.

To my mother...

Acknowledgement

I would like to express my gratitude to Assist. Prof. Dr. İlyas Çiçekli for his supervision, support, and guidance throughout my graduate studies.

I would like to thank committee members Prof. Dr. H. Altay Güvenir and Assoc. Prof. Dr. Ferda Nur Alpaslan, for reading and commenting on this thesis.

I would like to thank my family, especially my mother and my sister for supporting and believing in me throughout my life.

I would like to thank to Mehmet Köseoğlu for his great support, encouragement and understanding while writing this thesis.

And I would like thank also to Sami Ezercan for his help and great support. For their moral support I would like to thank to my colleagues at Aselsan Inc.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Translation had always been a complex cognitive progress, so computational (automatic) translation does. It involves detailed information on language, world, and culture, so automatic translation process involves every aspect of Natural Language Processing. Machine translation is simply defined as translation of texts from one natural language to another using computers [15]. In other words, machine translation is a sub-field of computational linguistics that investigates the use of computer software to translate text from one language to another.

Machine translation has a social contribution, since every day it is becoming more vital to communicate between the people who do not speak a common language. One of the very earliest pursuits in computer science, machine translation was seen as a subtle computational process, but today a number of systems are available which produce translation results that have sufficient quality to be useful in a number of specific domains.

According to European Association of Machine Translation (EAMT), some of current machine translation systems often allow for customization by domain or profession - improving output by limiting the scope of allowable substitutions. Improved output quality is one of the most important features of machine translation systems which can also be achieved by human intervention: for example, some systems are able to translate more accurately if the user has unambiguously identified which words in the text are names, cities etc.. With the assistance of these techniques, machine translation has proven useful as a tool to assist human translators, and in some cases can even produce output that can be used ”as is” [2].

The history of machine translation starts in the 1950s after the second world war. The Georgetown experiment in 1954 involved fully automatic translation of more than sixty Russian sentences into English. In this demonstration, there were 6 linguistic rules and 250 items in dictionary list (English-Russian) and the system was specialized in the area of organic chemistry [16].

As stated in [31], machine translation requires a lot of knowledge about the language like dictionaries, grammar rules, rewriting rules. So the paradigms are grouped according to the behavior of the systems to acquire knowledge about the languages.

There are various paradigms for machine translation. In [29], Somers classified different machine translation paradigms according to their possession of knowledge (whether from a corpus or hand-driven linguistic rules), these groups are shown as **Rule Based Machine Translation** and **Corpus Based Machine Translation** in Figure 1.1, along with their theoretical foundation (statistical or example-driven) for achieving translation process, **Statistical Machine Translation** and **Example Based Machine Translation**.

In the following sections, different approaches for machine translation are discussed by citing example systems, and lastly Example Based Machine Translation details are given. The system explained in this thesis covers extensions made for the learning and translation phases of an Example Based machine translation system.

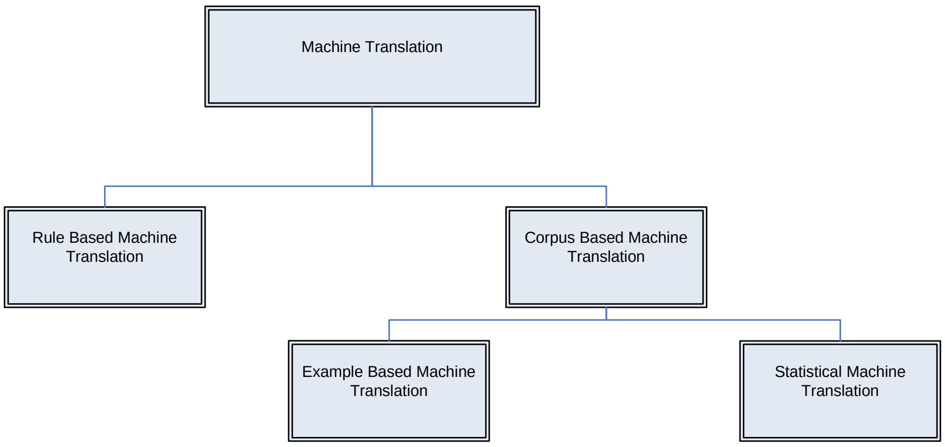
Machine Translation
Rule Based Machine Translation
Corpus Based Machine Translation
Example Based Machine Translation
Statistical Machine Translation

Figure 1.1: Paradigms for Machine Translation

1.1 Rule Based Machine Translation

Rule based machine translation (RBMT) is based on linguistic rules. Translation process consists of:

- analyze input text morphologically, syntactically and semantically
- generate text via structural conversions based on internal structure

Steps mentioned above uses a dictionary and a grammar which must be obtained by linguist(s) and this requirement is the main problem of RBMT as it is a time-consuming process, often referred as knowledge acquisition problem. Yet this feature makes RBMT diverge from automatic translation, since it needs a serious amount of linguistic knowledge. In this type of systems development and maintenance of rules are very hard, and we are not guaranteed to get the system operate as well as before addition of a new rule.

RBMT systems are large scale rule based systems, so their computational cost is really high, as they must implement every aspect of rules for a natural language (syntactic, semantic, structural transfer etc.)[31].

A Danish translation agency [3] is using a rule based machine translation system called PaTrans (stands for patent translator)[21] to translate patent applications.

In [18], Mahesh and Nirenburg has proposed a text meaning representation model (TMR) to be used in the rule based machine translation system Mikrokosmos. In this study, the representation model is seen as a way of knowledge sharing and it has been implemented for Spanish and Japanese lexicons in Mikrokosmos.

Carl et. al. [7], developed an advanced plug-in software module called Case-Based Analysis and Generation Module. This module serves as a front-end for conventional rule-based machine translation systems, the system is based on rule based machine translation whereas it also uses translation memory, so it is not wrong to say that this module is a linkage between rule based machine translation and example based machine translation.

Nagao, when he first started to work on the translation problem, tried to adopt his work to rule based machine translation, but he got pretty poor results, like the generated sentences were not readable, so he proposed another approach, which in advance leaded to example based machine translation [20].

1.2 Statistical Machine Translation

Corpus based machine translation (also referred as data driven machine translation) is an alternative approach for machine translation to overcome the problem of knowledge acquisition problem of rule based machine translation. Corpus Based Machine Translation (CBMT) uses, as it name points, a bilingual parallel corpus to obtain *knowledge* for new incoming translation. There are two different branches in Corpus Based Machine Translation as shown in Figure 1.1: Statistical Machine Translation and Example Based Machine Translation.

Systems using Statistical Machine Translation (SMT) paradigm uses bilingual corpus to learn translation models and monolingual corpus to learn the grammar of the target language. The SMT processes as seen in the Figure 1.2: the best translation is looked up according to the maximized probabilities got from two models (translation model derived from bilingual corpus and language model derived from monolingual corpus).

The statistical machine translation systems are rooted from the study of Brown et.al.[4]. In [4], they assign a probability $Pr(S,T)$ to every pair of sentences. This probability is the probability of S in source language interpreted as T, in the target language. They expect that this probability will be very small for the translations that are not correct. They view translation as given S they are seeking for T in order to maximize $Pr(S,T)$ as shown in Figure 1.2.

The project Apertium uses a statistical approach based on text-to-text translation for machine translation [10], actually this system is based on the previous systems interNOSTRUM and Traductor Universia which take the benefit of finite-state transducers and statistical techniques [13].

The Carabao Do-It-Yourself Machine Translation Kit [1] also uses statistical techniques where n-grams are used for translation, this system allows the users to build up their own dictionary, and do the translation between the languages that they can support input.

SMT like RBMT generates results from translations of single words and eliminating these results by probabilistic rules and linguistic rules respectively, which causes these approaches to yield the floor to example based machine translation.

1.3 Example Based Machine Translation

Example Based Machine Translation (EBMT) is an alternative model for rule based systems in machine translation world. In rule based systems linguistic knowledge is established by rules, but in EBMT, the linguistic knowledge is extracted from previous ”examples” of translations [28].

Makato Nagao, who had first proposed, example based machine translation

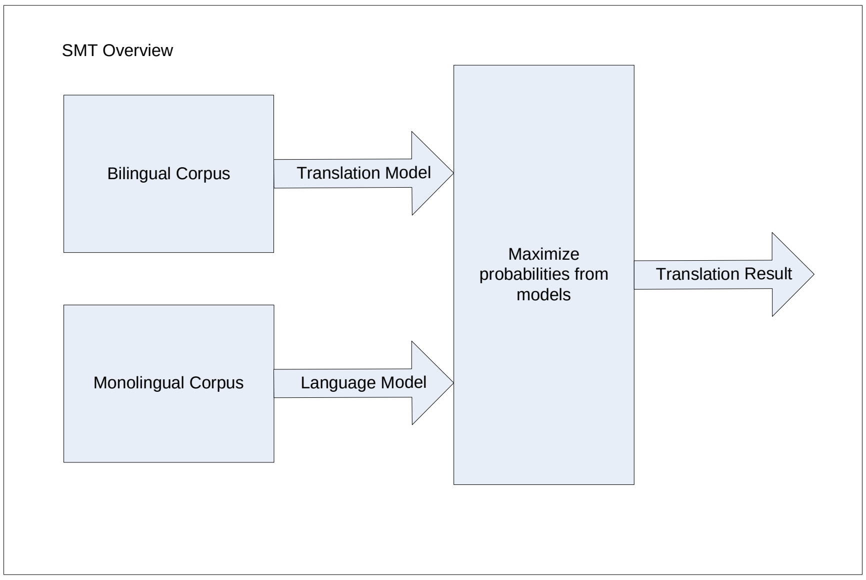

Figure 1.2: Statistical Machine Translation Overview

(he had actually proposed as *machine translation by analogy*), inspired this idea from the necessity to help Japanese people learn a second language like English. He had modeled the learning process as: a Japanese man is given short and simple English sentences with their Japanese correspondences; he memorizes these pairs and then becomes able to translate new sentences via these pairs in the memory. Actually this learning pattern summarizes the basic principles of example based machine translation.

Example based machine translation (also called as *analogy based, memory based, case based* and *experience guided*) stands somewhere between RBMT and SMT, as it integrates both data driven and rule based techniques.

In [28], Somers and Collins regards EBMT process as case-based reasoning. This paradigm has been evolved as an alternative to rule based systems. In this paradigm the experience is derived from past ’cases’, and the problem is solved using this experience. A new translation (which corresponds to ”new problem” in Case-Based Reasoning) is achieved by finding the most appropriate example from the translation database and using this example as a model for that sentence’s translation.

Actually the difference between rule based systems and case-based systems is summarized in the following quote of Riesbeck and Schank taken from [28]:

> "A rule based system will be flexible and produce nearly optimal answers but it will be slow and prone to error. A case based system will be restricted to variations on known situations and produce approximate answers but it will be quick and its answers will be grounded in actual experience. In very limited domains the tradeoffs favor the rule based reasoner but the balance changes as domains become more realistically complex." (Riesbeck & Schank, 1989)

In the following subsections, complete translation process within EBMT and the variations of the up-to-date developed EBMT systems according to differences in these process steps are discussed.

1.3.1 Translation Process within EBMT

It will be suitable first to define the steps of EBMT with the famous quote of Nagao, who is thought to be the inventor of EBMT, taken from [29]:

> "Man does not translate a simple sentence by doing deep linguistic analysis, rather, man does translation, first, by properly decomposing an input sentence into certain fragmental phrases,then by translating these phrases into other language phrases, and finally by properly composing these fragmental translations into one long sentence. The translation of each fragmental phrase will be done by the analogy translation principle with proper examples as its reference." (Nagao, 1984)

Nagao's this statement identifies the translation process using EBMT approach:

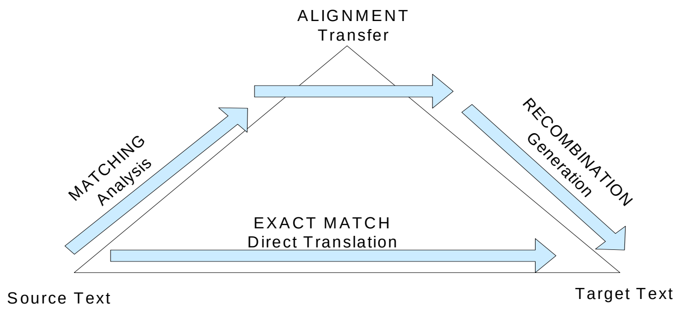

Figure 1.3: The Vauquois pyramid adopted for EBMT

- Matching fragments in database of examples (translation pairs)
- Specifying corresponding translation fragments
- Recombine results from previous steps to get the target text

Figure 1.3 shows the famous pyramid that identifies these three steps of example based machine translation. Labels in italics are the traditional labels, whereas labels in CAPITALS are the terms for EBMT [29]. Although different techniques are used by researchers at each step, what all have in common is the same work is done at the end.

I will briefly illustrate the translation process via an example from [27]. We want to translate the English sentence in 1.1 to the Japanese correspondent:

$$\text{He buys a book on international politics.} \tag{1.1}$$

If we know the following translation examples 1.2 and 1.3, we can translate sentence 1.1 into sentence 1.4 by imitating examples and combining fragments of them:

$$\begin{gathered}\underline{\text{He buys}}\text{ a notebook}\\ \underline{\text{Kare ha}}\text{ nouto wo kau}\end{gathered} \tag{1.2}$$

$$\text{I read } \underline{\text{a book on international politics}} \qquad (1.3)$$
$$\text{Wattashi ha } \underline{\text{kokusaiseiji nitsuite kakareta hon wo yomu}}$$

$$\underline{\text{Kare ha kokusaiseiji nitsuite kakareta hon wo yomu}} \qquad (1.4)$$

1.3.2 Problems of the Approach

Before going on with the details of the translation process, I will briefly mention the problems arising from the data requirements of the EBMT approach. As the system is example based, gathering examples, number of examples, suitability of them and lastly storage of examples builds up the main problems and divergence point of example based machine translation systems.

1.3.2.1 Parallel Corpora

As mentioned earlier, one of the most important knowledge base that EBMT uses is *parallel aligned corpora*. Here parallel stands for the text and its correspondent translation kept together. Aligned corpus is the two texts that have been analyzed into corresponding segments. Alignment problem can be overcome by building the corpora by hand, but this is an error-prone and time-consuming process.

In some domains there are specialized studies for building up parallel corpora like Canadian and Hong Kong parliaments provide bilingual corpora for parliament proceedings [29]. World Wide Web can be an excellent resource for an example based machine translation system, so some systems make use of web pages that have versions for multi-languages as bilingual corpus. But for the resources that are driven from these type of resources (world wide web etc.) the parallel corpora problem arises, as there is a probability that sentence and its translation can be in different orders.

	Example Size	Translation Accuracy
Construction 1	100	30%
	774	65 %
Construction 2	100	75%
	689	100 %

Table 1.1: Experiment results taken from MT systems with Japanese ad nominal particle construction

1.3.2.2 Example Size

After gathering examples from a resource, there remains yet another problem: "Are these examples enough for a translation system?". As the system is "example" based, another important point for a system's performance is how many examples will be used. Although not only the example size but also the way that they are stored affects the results together, results that are taken from machine translation systems show that example size affects the performance dramatically as seen in Table 1.1 composed from [29]. In this experiment, adding examples to the database improved the system performance, starting from 100 adding 100 examples each time (till 774) enhanced the performance from 35% to 65%. In another experiment, the system's performance was about 75% with 100 examples, and reached to 100% with 689 examples.[29]

1.3.2.3 Suitability of Examples

Even though we have a lot of examples, the system would still be not accurate enough. Another important point is that the examples must be suitable. By suitability it is meant that: a lot of examples will lead to the same translation example, or examples can be in conflict the same phrases can lead to different translations.

Öz & Çiçekli [22], involved a similarity metric to count the frequency of examples, so large number of similar examples will have a high score.

One of the most frequently used techniques is distinguishing exceptional and general examples. This approach causes EBMT to behave more like RBMT.

1.3.2.4 Example Storage

Storage of examples directly changes the paradigm for the matching phase of an EBMT system. Actually there is a fact that the simpler the examples are kept, the harder is the matching phase. The simplest way to keep examples is to keep the source text and its correspondence in the target language, but in this case the matching for an incoming text is pretty hard.

There are several methods to keep the examples:

- **Annotated Tree Structure**

  This data structure is used by Sato&Nagao [27], Sadler et.al [25].

  In Sato&Nagao's system, a translation example consists of 3 parts:

  - An English word-dependency tree
  - A Japanese word-dependency tree
  - Correspondence links

  As can be seen from Figure 1.4, the representation of an example, building this type of a corpus is very time-consuming and demanding.

  A similar method is used by Watanabe,[33] , in that system examples are stored in a tree structure as seen in Figure 1.5 , with parse information.

  A related way is used by Poutsma, 1998 and Way,1999 [24], in this case examples are stored parsed by a data-oriented parsing technique and for matching case, subtrees are combined for whole translation process.

  Sample storage of example 1.5 is seen in Figure 1.6.

$$\text{John likes Mary} \tag{1.5}$$

```
ewd_e({e1,[buy,v],
        [e2,[he,pron]],
        [e3,[notebook,n],
           [e4,[a,det]]]]).

%% Sample storage of: He buys a notebook

jwd_e([j1,[kau,v],
         [j2,[ha,p]],
            [j3,[kare,pron]]],
         [j4,[wo,p],
              [j5,[nouto,n]]]]).

%% Sample storage of: Kare ho nouto wo kau.

clinks([[e1,j1],[e2,j3],[e3,j5]]).

%% e1 <-> j1, e2 <-> j3, e3 <-> j5
```

Figure 1.4: Correspondence Link Representation for Example Pairs

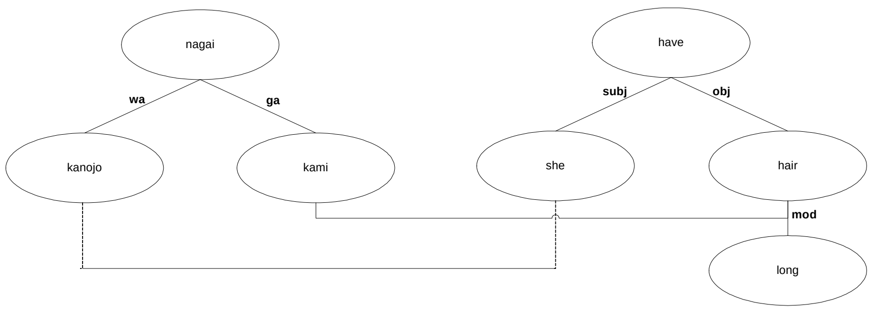

Figure 1.5: Annotated Tree Structure for *she have long hair*

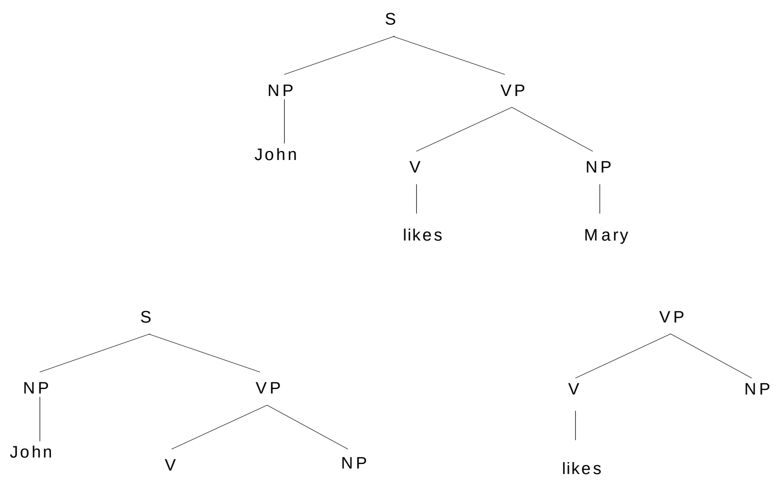

Figure 1.6: Parse trees belonging to *John likes Mary*

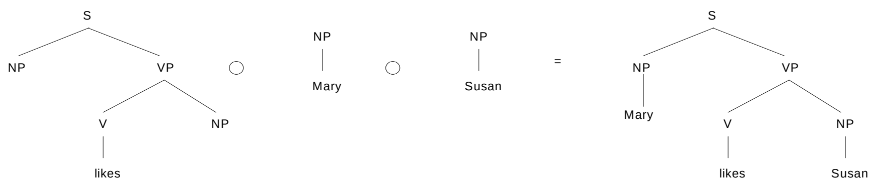

Figure 1.7: Translation of *Mary likes Susan* using subtrees

So with this example, the system will be able to translate the sentence 1.6, by combining subtrees 2 and 3 as seen in Figure 1.7.

$$\text{Mary likes Susan} \tag{1.6}$$

Zhao & Tsuji, 1999, used a multi-dimensional feature graph where features are speech acts, semantic roles, syntactic categories, functions etc.[34]

- **Generalized Examples**

  In this type of example storage, similar examples are stored as a single generalized example.[29]

  There are three methods for example generalization [6]:

  - Manual generation of equivalence classes (generalized parts in an example)

- Automatic extraction of equivalence classes
- Transfer rule induction

The most applicable and accepted generalization technique is transfer rule induction.

In [9], Çiçekli&Güvenir used this technique to derive translation templates from bilingual examples in the system called TTL. In this system, generalized form of two examples is called translation template. A translation template is inferred from two examples briefly by replacing different parts of sentences by variables. This algorithm will be explained in detail in Chapter 2.

From examples 1.7 and 1.8, we can learn template 1.9, if the different part of these sentences correspond to each other [9]:

$$\text{I will drink } \underline{\text{orange juice}} \leftrightarrow \underline{\text{Portakal suyu}} \text{ içeceğim} \tag{1.7}$$

$$\text{I will drink } \underline{\text{coffee}} \leftrightarrow \underline{\text{Kahve}} \text{ içeceğim} \tag{1.8}$$

$$\text{I will drink } X^1 \leftrightarrow X^2 \text{ içeceğim} \tag{1.9}$$

$$\text{orange juice} \leftrightarrow \text{portakal suyu}$$

$$\text{coffee} \leftrightarrow \text{kahve}$$

In [6] Brown used the technique similar to Çiçekli&Güvenir,2001, for generalizing the different parts of the sentences with category names instead of variables as in [9].

Using this type of example storage, examples 1.10 and 1.11, can be stored in form of 1.12. [6]

$$\text{205 delegates met in London} \tag{1.10}$$

$$\text{200 delegates met in Paris} \tag{1.11}$$

$$\textbf{<number>}\text{delegates met in }\textbf{<city>} \tag{1.12}$$

- **Statistical Approaches**

  In this method, precomputed probabilities of bilingual word pairs (translational models) are stored instead of examples. The language and translational models are optimized to get the target string.

1.3.3 Matching Phase of EBMT

Matching phase is the most important step of translation. In this step, the database is searched for the source sentence, to find the best match example for it. All of the methods used to solve the problems described in the previous section, directly influences the matching step and so the overall performance of the system. Below are the common methods for matching:

- **Character Based:** It is used by Sato, 1992 [26]. A distance or similarity measure is kept and matching depends only on that measure. Unfortunately this method cannot produce right results for the cases of the indirectly related words. For example the system cannot translate *"Give me the big ball"* using the example *"Give me the small ball"*, as relation between big and small is not kept.

- **Word Based:** It is used by Nagao, Sumita and Iida [30]. In word based matching method, matching is said to be done when the words in the source string can be replaced by near synonyms in the example. Examples 1.13 and 1.14 are from the Nagao's system [20].

$$\text{A man eats vegetables} \leftrightarrow \text{Hito wa yasai o taberu} \tag{1.13}$$

$$\text{Acid eats metal} \leftrightarrow \text{San wa kinzoku o okasu} \tag{1.14}$$

Assume that the input 1.15 is given:

$$\text{He eats potatoes} \tag{1.15}$$

The system correctly matches to the first example as *potatoes* are more similar to *vegetables* than *acid.*

- **Annotated Word Based:** This technique goes further in linguistic knowledge. It uses the part of speech tags. It can be said that examples are kept partially parsed. This methodology is spreadly used by Cranias et.al. [11] [12], Veale&Way [32]. This kind of matching makes use of annotated tree structure (described in the previous section) where explicit links are kept for correspondences. Usage of part of speech tags, really contributes a lot to the sentence composition phase which will be explained in the next section.

- **Parsing Based:** In case of example storage with generalization, this method is used to match the best example (generalized example) to translate the given text.

  With generalized example 1.16, we have the given sentence 1.17, where it is assumed that X^1 and Y^1 are two variables and they are translations of each other:

$$\text{I will drink } X^1 \leftrightarrow Y^1 \text{ içeceğim} \tag{1.16}$$

$$\text{I will drink tea} \tag{1.17}$$

  The given sentence 1.17 can be parsed using generalized example 1.16, so it will be possible to get the translated sentence 1.18, if we know the correspondence 1.19:

$$\text{Çay içeceğim} \tag{1.18}$$

$$\text{tea} \leftrightarrow \text{çay} \tag{1.19}$$

The details of this type of matching will be given in detail in the next chapter.

1.3.4 Adaptation Phase of EBMT

From the previous step we have the correct examples that will be used for translation, but which part(s) of these examples will be used. Adaptation clarifies this question and specifies the fragments of the examples that are to be used for translation.

Assume that we have the sentence 1.20 to be translated into some language. From matching step we have correctly found two examples in 1.21 that suits our input string. After adaptation we will have the underlined fragments that are to be recombined to get the target sentence.

$$\text{He buys a book on politics} \tag{1.20}$$

$$\begin{gathered}\underline{\text{He buys}}\text{ a notebook}\\ \text{I read }\underline{\text{ a book on politics}}\end{gathered} \tag{1.21}$$

1.3.5 Recombination Phase of EBMT

As we have the fragments that are to be combined to get the translated string, it seems pretty easy to get just by concatenation. For the example above we get the correct result for English, because English is a little or no inflectional language. For other languages like Japanese and Turkish which carry strong inflection property, yet remains some problems.

In German, nearly all words are inflected due to verb, like in 1.22 :

Der schöne Junge ass seinen Frühstück (1.22)
Ich sah den schönen Jungen

Both of the sentences include the handsome boy (der schöne Junge), but in different cases (nominative and accusative respectively). After we got the fragments to be used for translation, we still have a problem to be solved: Which of these cases will be used?

To solve this problem Grefenstette, 1999 [14] uses a statistical technique. He extracts n-grams (generally trigrams or bigrams) and uses the most probable case (nominative or accusative) according to n-grams result. The results can be extracted from corpora or just from world wide web. The results that he has taken when he searched AltaVista for "ich sah den" and "ich sah der": 341 is for the former case, and only 17 is for the latter case.

If you ping a search for Turkish, "seni gördüm", "sen gördüm", "sende gördüm" and the results were 940, 201 and 458, respectively. Furthermore to test the trigram case I have searched for "ben seni gördüm" and "ben sen gördüm" the results were 24 and 0 respectively. This is not a reliable result as the subject can be extracted from the verb, the subject "ben" need not to be used, but it gives a sense for the power of statistical approach.

1.4 Thesis Outline

In this thesis we propose a method to prevent the incorrect translations that the previous system, explained in the next chapter, produces. The previous system learns structures called translation templates from the bilingual corpus and keeps this templates to be used in the translation phase. The templates learned in the previous system contains variables and these variables do not contain the type information (noun, verb etc.) of the words that these variables replace. We propose a learning algorithm that associates the type information while replacing the differing words with variables. In this manner, the type associated template learning algorithm prevents the system to produce incorrect translation results.

The remaining part of the thesis contains the detailed information about the previous system that we have made extensions on and the enhancements that are done. In the next chapter, previous version of the system is explained. Chapter 3, gives the details of type associated translation templates, confidence factor assignment to type associated translation templates are described in Chapter 4. Whole system architecture with learning and translation components is given in Chapter 5. After giving the test results in Chapter 6, the thesis ends with Conclusion and Future Work composing Chapter 7.

Chapter 2

Translation Template Extraction

The system explained in this thesis is based on the previous system developed by Çiçekli and Güvenir. This system is described in [9].

The system in [9], uses English-Turkish pairs as it has been used for English - Turkish translation. As stated in [5], the usage of templates in example based machine translation, decreases the number of examples needed for translation process as the examples are kept in a generalized manner, so in the TTL system the translation templates are learned from the bilingual corpus to be used in the translation time. For the learning process the inductive learning hypothesis is taken as principle. The inductive learning hypothesis approximates the target function well over a sufficiently large set of examples (bilingual corpus) that will also approximate the target function (translation) well over other unobserved examples [19].

The working principle of the system is illustrated in Figure 2.1. Firstly, examples are generalized using bilingual corpus (*translation template extraction*) and when a source sentence is fed to the system, the appropriate templates are chosen (*matching*) and translation process is completed with the recombination of these templates to get the target sentence.

In order to summarize the process, I will use the following examples from [9].

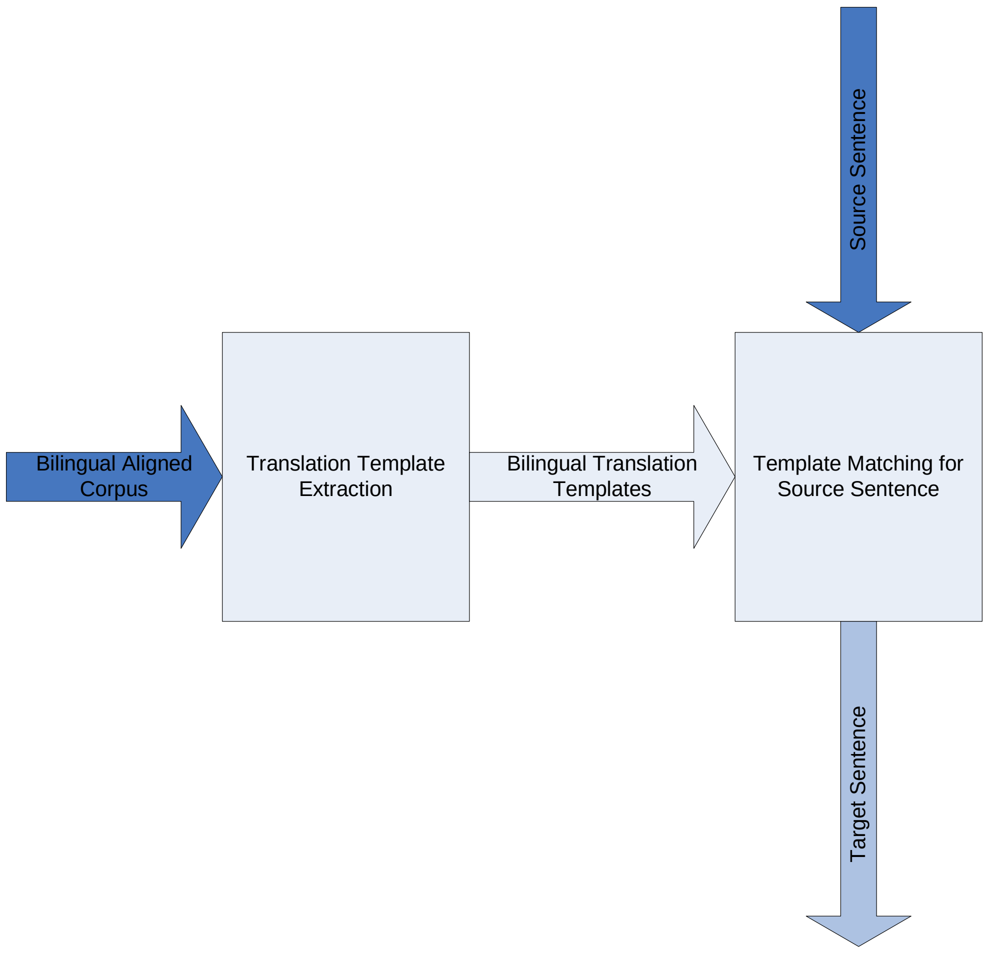
Source Sentence
Bilingual Aligned Corpus
Translation Template Extraction
Bilingual Translation Templates
Template Matching for Source Sentence
Target Sentence

Figure 2.1: Basic Principles of the TTL System described in [9]

$$\begin{aligned} \text{I will drink orange juice} &\leftrightarrow \text{portakal suyu içeceğim} \\ \text{I will drink coffee} &\leftrightarrow \text{kahve içeceğim} \end{aligned} \tag{2.1}$$

In examples 2.1, there are similar parts in both languages (*I will drink* and *içeceğim*, respectively) and there are differing parts (*orange juice, portakal suyu* and *coffee, kahve*). The first heuristic to build up a translation template is to replace differing parts with variables. So in the system the examples 2.1, will be kept as in 2.2:

$$\text{I will drink } X^1 \leftrightarrow Y^1 \text{ içeceğim} \tag{2.2}$$

Here along with the translation template 2.2, the templates 2.3 are learned:

$$\begin{aligned} \text{orange juice} &\leftrightarrow \text{portakal suyu} \\ \text{coffee} &\leftrightarrow \text{kahve} \end{aligned} \tag{2.3}$$

In translation templates 2.3, there are no variables, in [9], these type of translation templates are called **atomic translation templates** (also called as *facts*), whereas translation templates with one or more number of variables are called **similarity translation template** or **difference translation template** classified according to the part that the variables replace.

2.1 Inferring Translation Templates

The algorithm defined in [9] derives translation templates using two different substitution methods. Similarity (similarity between E_a and E_b where E_a and E_b are two different examples from aligned corpus) substitution and difference (D^1_*

and D_*^2 are differences belonging to $lang_1$ and $lang_2$ respectively, and they do not contain any common string) substitution.

For each pair of examples in the corpus a *match sequence* is generated. The structure of the a match sequence M_{ab} extracted from examples E_a and E_b is as in 2.4. [9]

$$S_0^1,D_0^1,S_1^1,.....,D_{n-1}^1,S_n^1 \leftrightarrow S_0^2,D_0^2,S_1^2,.....,D_{m-1}^2,S_m^2 \qquad (2.4)$$
$$\text{where } n,m >= 1$$

Here S represents similar parts for the examples where D represents the different parts. If the number of differences or similarities are zero, then no template is learned from these two examples.

2.6 shows an example match sequence for examples 2.5.

$$\begin{aligned} \text{black book +PL} &\leftrightarrow \text{siyah kitap +PL} \\ \text{black car +PL} &\leftrightarrow \text{siyah araba +PL} \end{aligned} \qquad (2.5)$$

$$\text{black (book,car) +PL} \leftrightarrow \text{siyah (kitap,araba) +PL} \qquad (2.6)$$

So the elements of the match sequence are:

$$\begin{aligned} S_0^1 &= \text{black} \\ D_0^1 &= \text{(book,car)} \\ S_1^1 &= \text{+PL} \\ S_0^2 &= \text{siyah} \\ D_0^2 &= \text{(kitap,araba)} \\ S_1^2 &= \text{+PL} \end{aligned}$$

So after that match sequence is found, the translation templates are found using similarity or difference substitution methods.

2.1.1 Learning Similarity Translation Templates

In this method all different parts of the examples are substituted with variables. The most important point for forming this kind of translation template is to have enough number of facts (atomic templates) that proves the template to be correct. Assume that the number of differences in a match sequence M_{ab} is n. Then *n-1* different parts must correspond to each other. In match sequence 2.7, there is only 1 difference, so we need 0 facts learned earlier, in other words, we are sure that these differing constituents are translations of each other as in 2.7.

$$\begin{aligned} \text{book} \leftrightarrow \text{kitap} \\ \text{car} \leftrightarrow \text{araba} \end{aligned} \tag{2.7}$$

So we can infer that:

$$\text{black } X^1 \text{ +PL} \leftrightarrow \text{siyah } Y^1 \text{ +PL} \tag{2.8}$$

In a translation template the variables that replace English differences are denoted by X, and for Turkish parts variables are denoted by Y. The superscripted numbers show which variables are correspondences of each other in English and Turkish, so having the same number superscripted means, these variables correspond to each other.

For examples in 2.9, we infer the match sequence 2.10.

$$\begin{aligned} \text{at least three notebook +PL} \leftrightarrow \text{en az üç defter} \\ \text{at most three book +PL} \leftrightarrow \text{en fazla üç kitap} \end{aligned} \tag{2.9}$$

$$\begin{array}{c}\text{at (least, most) three (notebook,book) +PL} \;\leftrightarrow \\ \text{en (az,fazla) üç (defter, kitap)}\end{array} \tag{2.10}$$

In match sequence 2.10, there are 2 differences, so we must previously know one of the following combination of facts 2.11 in order to infer a translation template from these examples.

$$\begin{array}{rcl}\text{least} \leftrightarrow \text{az} & , & \text{most} \leftrightarrow \text{fazla} \\ \text{least} \leftrightarrow \text{defter} & , & \text{most} \leftrightarrow \text{kitap} \\ \text{notebook} \leftrightarrow \text{defter} & , & \text{book} \leftrightarrow \text{kitap} \\ \text{notebook} \leftrightarrow \text{az} & , & \text{book} \leftrightarrow \text{fazla}\end{array} \tag{2.11}$$

Assume that before obtaining match sequence 2.10, we have learned the facts in 2.12:

$$\begin{array}{rcl}\text{least} & \leftrightarrow & \text{az} \\ \text{most} & \leftrightarrow & \text{fazla}\end{array} \tag{2.12}$$

So for the match sequence 2.10, the number of unknown differences for both languages decreases to 1, and we can infer translation template 2.13 along with facts in 2.14:

$$\text{at } X^1 \text{ three } X^2 \leftrightarrow \text{en } Y^1 \text{ üç } Y^2 \tag{2.13}$$

$$\begin{array}{rcl}\text{notebook} & \leftrightarrow & \text{defter} \\ \text{book} & \leftrightarrow & \text{kitap}\end{array} \tag{2.14}$$

```
procedure similarityTT($M_{ab}$)
    $M_{ab}$ *is the match sequence of examples* $E_a$ *and* $E_b$ *and defined as:*
    $M_{ab} = S_0^1, D_0^1, ...., D_{n-1}^1, S_n^1 \leftrightarrow S_0^2, D_0^2, ...., D_{m-1}^2, S_m^2$
    if( n = m = 1)
        $learnedTranslationTemplate = S_0^1, X^1, S_1^1 \leftrightarrow S_0^2, Y^1, S_1^2$
        $learnedFact_1 = D_{0,ea}^1 \leftrightarrow D_{0,ea}^2$
        $learnedFact_2 = D_{0,eb}^1 \leftrightarrow D_{0,eb}^2$
    else if(n = m >1)
        *here we assume that n-1 correspondences are known previously as facts*
        *assume that unmatched difference pairs are:*
        $((D_{k_n,ea}^1, D_{k_n,eb}^1), (D_{l_n,ea}^2, D_{l_n,eb}^2))$
        replacing all matched pairs with $X^{1..n-1}$ and $Y^{1..n-1}$ we get
        match sequence $M_{ab}$ as: $M_{ab}WDV$
        $learnedTranslationTemplate =$
            $M_{ab}WDV$ if $X^1 \leftrightarrow Y^1$ and .... and $X^n \leftrightarrow Y^n$
        $learnedFact_1 = D_{k_n,ea}^1 \leftrightarrow D_{l_n,ea}^2$
        $learnedFact_2 = D_{k_n,eb}^1 \leftrightarrow D_{l_n,eb}^2$
```

Table 2.1: Similarity Translation Template Extraction Algorithm

At this point we can say that learning a translation template yields learning of zero or two facts, as long as we must have the appropriate ground (previously learned facts) for completion of learning process of a similarity translation template.

As you can see in Table 2.1, the algorithm works for equal number of differences in both languages, L_a and L_b. Assume that there are two examples like in 2.15. [9]

$$\begin{aligned} &\text{I come}\underline{\text{+PAST}} \leftrightarrow \text{gel}\underline{\text{+PAST}}\text{+1SG} \\ &\text{you go}\underline{\text{+PAST}} \leftrightarrow \text{git}\underline{\text{+PAST}}\text{+2SG} \end{aligned} \tag{2.15}$$

$$\text{(I come,you go) +PAST} \leftrightarrow \text{(gel,git) +PAST (+1SG,+2SG)} \tag{2.16}$$

In this case the number of differences for L_a is 1, while the number of differences for L_b is 2. This type of situations are faced frequently, as the linguistic structure of the languages are so different from many aspects.

The Similarity TTL algorithm defined in Table 2.1, cannot learn a translation template from these two examples as they have different number of differing parts. To overcome this problem, the algorithm is fed with all possibilities of match sequences. In other words, if the number of differing constituents are different from each other like in 2.16, differing parts are pieced in order to get appropriate match sequence like in 2.17.

$$\text{(I,you),(come,go) +PAST} \leftrightarrow \text{(gel,git) +PAST (+1SG,+2SG)} \tag{2.17}$$

So by piecing the match sequence 2.16, we get 2.17. This time the STTL algorithm can learn a translation template from these examples.

2.1.2 Learning Difference Translation Templates

In previous section, we were focused on inferring a translation template by replacing the differing parts with variables and remaining the similar parts, as stated earlier this type of templates are called similarity translation templates, now we will try to infer translation templates by keeping the different parts and replacing the similar parts with variables, this time the learned translation template will be called difference translation template.

For difference template learning, the similarities are replaced with variables, and the difference pairs are splitted in order to get two match sequences with similarity variables. Assume that there are two examples as in 2.18.

$$\text{I }\underline{\text{break+PAST the}}\text{ window} \leftrightarrow \text{pencere }\underline{\text{+ACC kır+PAST}}\text{+1SG} \tag{2.18}$$

$$\text{You }\underline{\text{break+PAST the}}\text{ door} \leftrightarrow \text{kapı}\underline{\text{+ACC kır+PAST}}\text{+2SG} \tag{2.19}$$

The match sequence for 2.18 is given in 2.20.

```
procedure differenceTT($M_{ab}$)
    $M_{ab}$ is the match sequence of examples $E_a$ and $E_b$ and defined as:
    $M_{ab} = S_0^1, D_0^1, ...., D_{n-1}^1, S_n^1 \leftrightarrow S_0^2, D_0^2, ...., D_{m-1}^2, S_m^2$
    if( $n = m > 1$)
        if( number of corresponding similarities = n-1 )
            Assume that unmatched similarity is: $(S_{k_n}^1, S_{l_n}^2)$
            Replace all corresponding similarities $S_{k_i}^{lang1}, S_{l_i}^{lang2}$
                with $X^i$ for English and $Y^i$ for Turkish
            Split the match sequence $M_{ab}WSV$ into two match sequences
            with respect to differences
            $learnedTranslationTemplate_1 = M_aWDV$
            $learnedTranslationTemplate_2 = M_bWDV$
            $fact = S_{k_n}^1 \leftrightarrow S_{l_n}^2$
```

Table 2.2: Difference Translation Template Extraction Algorithm

(I,you) break+PAST the (window,door) $\leftrightarrow$
(pencere, kapı) +ACC kır+PAST (+1SG,+2SG) (2.20)

In order to infer a difference translation template, we need at least one non-empty difference and similarity on both sides. From 2.20, we can infer two difference translation templates and an atomic template (fact), like in 2.21.

$$\begin{aligned} \text{I } X^1 \text{ window} &\leftrightarrow \text{pencere } Y^1 \text{ +1SG} \\ \text{You } X^1 \text{ door} &\leftrightarrow \text{kapı } Y^1 \text{ +2SG} \\ \text{break+PAST the} &\leftrightarrow \text{+ACC kır+PAST} \end{aligned} \tag{2.21}$$

If the number of similarities is equal to $n > 1$, like inferring a similarity template, we need a prior knowledge of $n - 1$ corresponding similarities.

The difference translation template algorithm is defined in Table 2.2. If we apply the algorithm to 2.22, firstly we get the match sequence as shown in 2.23. We have only one similar string on both sides, that means we do not need any prior knowledge. So by splitting the match sequence in 2.23, we learn the difference translation templates as shown in 2.24 and additionally an atomic template in 2.25.

$$\begin{aligned}&\text{I bring+PAST my blue notebook} \leftrightarrow\\&\quad\text{mavi defter+1SGPoss+ACC getir+PAST+1SG}\\&\text{she bring+PAST my green book} \leftrightarrow\\&\quad\text{yeşil kitap+1SGPoss+ACC getir+PAST+3SG}\end{aligned} \tag{2.22}$$

$$\begin{aligned}&\text{(I,she) bring+PAST my (blue notebook, green book)} \leftrightarrow\\&\quad\text{(mavi defter, yeşil kitap) +1SGPoss+ACC getir+PAST (+1SG,+3SG)}\end{aligned} \tag{2.23}$$

$$\begin{aligned}\text{I } X^1 \text{ blue notebook} &\leftrightarrow \text{mavi defter } Y^1 \text{ +1SG}\\ \text{she } X^1 \text{ green book} &\leftrightarrow \text{yeşil kitap } Y^1 \text{ +3SG}\end{aligned} \tag{2.24}$$

$$\text{bring+PAST my} \leftrightarrow \text{+1SGPoss+ACC getir+PAST} \tag{2.25}$$

Like in similarity translation template extraction algorithm in case of unequal similarity parts in the match sequence, we fed the algorithm with proper possibilities of pieced similar parts, to get equal number of similarities on both sides of the sequence.

2.2 Problem Description

In the previous section, I have explained the previous work that forms the basis of the system described in this thesis. There is a weak point in the previously described translation template extraction method. After the translation template is inferred either by difference replacing or by similarity replacing, all the information about the variables are lost.

Let me explain the problem with an example: assume that we have the training examples as shown in 2.26. The match sequence for the examples will be like in 2.27 and the output of the similarity translation template algorithm is shown in 2.28.

$$\begin{aligned} \text{I come +PAST} &\leftrightarrow \text{gel+PAST+1SG} \\ \text{I go +PAST} &\leftrightarrow \text{git+PAST+1SG} \end{aligned} \tag{2.26}$$

$$\text{I (come,go) +PAST} \leftrightarrow \text{(gel,git)+PAST+1SG} \tag{2.27}$$

$$\begin{aligned} \text{I } X^1 \text{ +PAST} &\leftrightarrow Y^1 \text{ +PAST+1SG} \\ \text{come} &\leftrightarrow \text{gel} \\ \text{go} &\leftrightarrow \text{git} \end{aligned} \tag{2.28}$$

For the sake of example, we are assuming that we have the prior knowledge (fact) described in 2.29.

$$\text{shy} \leftrightarrow \text{utangaç} \tag{2.29}$$

Assume that the translation system with prior knowledge given above, is fed with the Turkish input *utangaçtım* whose lexical form is given in 2.30.

$$\text{utangaç +PAST+1SG} \tag{2.30}$$

The system's matching component will correctly choose the translation template given in 2.31 as the second part of the template matches with the sentence to be translated. After finding the appropriate template there remains process of filling the variable part of the template. We know that the system knows the fact 2.29.

$$\text{I } X^1 \text{ +PAST} \leftrightarrow Y^1 \text{ +PAST+1SG} \tag{2.31}$$

Using the templates given in 2.29 and 2.31, we translate the sentence 2.30 as given in 2.32.

$$\text{I shy +PAST} \leftrightarrow \text{utangaç +PAST+1SG} \tag{2.32}$$

As can be seen in 2.32, the answer of the system for the input *utangaçtım* is *I shy +PAST* which is grammatically incorrect.

The reason of this failure is, as I stated earlier in this section, the information about the variables (differences or similarities) are lost while replacing them with variables, this observation points out that we need to hold an additional information about the replaced variables in a translation template.

From a linguistic point of view the most characteristic information about a word is its part-of-speech (POS) tag. A part-of-speech tag is the linguistic category of words. It describes the type of a word. Common linguistic categories include *nouns, verbs, adjectives etc.*

Assume that we have hold the type (POS tag) information of the words that we have replaced with variables, the translation template in 2.31 will look like as in 2.33. The type information associated with the variable in 2.33 is *verb* as it is the type of replaced variables *gel* and *git*.

$$\text{I } X^1_{verb} \text{ +PAST} \leftrightarrow Y^1_{verb} \text{ +PAST+1SG} \tag{2.33}$$

So if we turn back to our translation problem, we fed the system again with the sentence *utangaçtım*, now the system finds the same template again but this time it fails to produce result in 2.32 as the type of *utangaç* (*adjective*) does not match with the type expectation of the translation template (*verb*).

In our system, we aim to eliminate the results that the system produces, by the help of type information associated with the translation templates.

The details of associating the type information to the translation templates will be explained in the next chapter.

Chapter 3

Type Associated Translation Templates

In this chapter, I will describe the learning process of type associated (supported) translation templates in detail. Our study is based on [8] with modifications for the whole translation process. In the first section, I will emphasize the modification of the similarity translation template algorithm, the second part will be about getting the type information of variables.

3.1 Modification of Similarity Translation Template Extraction Algorithm

In this part, I will explain the type associated similarity translation template extraction in detail. This system is based on learning of similarity translation templates, while associating the type information difference translation template learning is left out of the scope.

Before giving the pseudo code of the algorithm, I will explain the algorithm by giving examples. All of the examples will be given as an English-Turkish pair like used in the system.

Assume that we have the examples as shown in 3.1 as used in previous chapter, so the match sequence will be just like in 3.2.

$$\begin{aligned} \text{I come +PAST} &\leftrightarrow \text{gel+PAST+1SG} \\ \text{I go +PAST} &\leftrightarrow \text{git+PAST+1SG} \end{aligned} \tag{3.1}$$

$$\text{I (come,go) +PAST} \leftrightarrow \text{(gel,git)+PAST+1SG} \tag{3.2}$$

We realize that the different parts in the match sequence 3.2 is *(come,gel)* and *(go,git)* respectively. In order to keep the translation template with type information, firstly we must get the type (part-of-speech tag) of the words in the differing parts.

There are multiple ways to get the type information of a word, including getting the type information via a morphological analyzer interface. In this case, every time we want to learn a translation template from the same word we will access the interface and get the type information. We have chosen a different way to store the training examples. The training set (examples) are stored in their lexical levels this representation's details will be given in Chapter 5, System architecture.

To intensify the whole learning process I will give one more example after each step of the learning algorithm is explained. The translation pairs that I will use for this exemplification are *(boys are coming, oğlanlar geliyorlar)* and *(boys are not going, oğlanlar gitmiyorlar)*, whose match sequences are:

boy+Noun +Pl be+Verb +Pres +Pl (come+Verb,not+Adv go+Verb) +Prog ↔
oğlan+Noun +A3pl +Pnon +Nom (gel+Verb +Pos,git+Verb +Neg) +Prog1 +A3pl)

3.2 Learning Process

To illustrate the learning process, I will use the example given in 3.1. According to our example storage paradigm the match sequence is as shown:

I+Pron+Pers +Nom +1P +Sg (come+Verb, go+Verb) +PastTense +123SP $\leftrightarrow$
(gel+Verb,git+Verb) +Pos +Past +A1sg

According to the similarity translation template extraction algorithm, the inferred templates will be just like in 3.3.

$$\begin{array}{l} \text{I+Pron+Pers +Nom +1P +Sg } X^1_{verb} \text{ +PastTense +123SP} \leftrightarrow \\ \quad Y^1_{verb} \text{ +Pos +Past +A1sg} \\ \qquad\qquad \text{come+Verb} \leftrightarrow \text{gel+Verb} \\ \qquad\qquad \text{go+Verb} \leftrightarrow \text{git+Verb} \end{array} \tag{3.3}$$

In the previous example we have inferred the type of the variable as *verb* as both examples contained the same type of variables, in case of having examples like in 3.4 the scenario changes.

$$\begin{array}{l} \text{black+Adj notebook+Noun +Sg} \leftrightarrow \\ \quad \text{siyah+Adj defter+Noun +A3sg +Pnon +Nom} \\ \text{one+Num+Ord notebook+Noun +Sg} \leftrightarrow \\ \quad \text{bir+Num+Ord defter+Noun +A3sg +Pnon +Nom} \end{array} \tag{3.4}$$

$$\begin{array}{l} \text{(black+Adj,one+Num+Ord) notebook+Noun +Sg} \leftrightarrow \\ \quad \text{(siyah+Adj,bir+Num+Ord) defter+Noun +A3sg +Pnon +Nom} \end{array} \tag{3.5}$$

The match sequence 3.5, contains one difference but this time we are faced with two different variable types *Adj* and *Num* in both sides. So now we need to

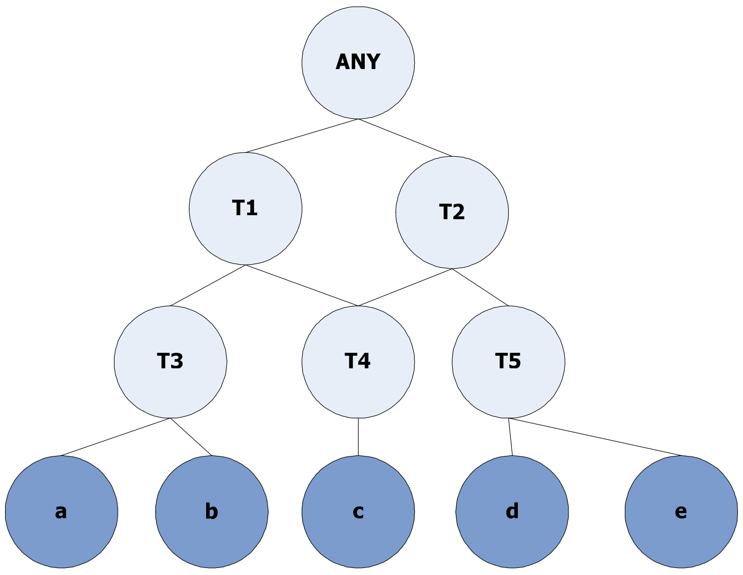

Figure 3.1: Structure of the lattice

somehow generalize these types and associate that type with translation template. In the next section, I will explain the details for inferring a type sequence from two different type sequences.

3.3 Lattice

Lattice means arrangement of crossing thin strips of a material. We have chosen our linguistic part-of-speech tag arrangement model as a lattice. The main reason beyond this analogy is the requirement that we have to arrange the part-of-speech tags hierarchically and there can be some cross-cutting types that belongs to more than one category.

To find a common category for two different types we developed a lattice like structure which resembles an undirected acyclic graph. Figure 3.1, shows the structure of the lattice that we have used in our system.

Here the leafs are the constituents whose type is the first parent of that leaf. As can be seen from the figure, a type can be thought as a subtype of more than one category like in case of T_4 is subtype of both T_1 and T_2.

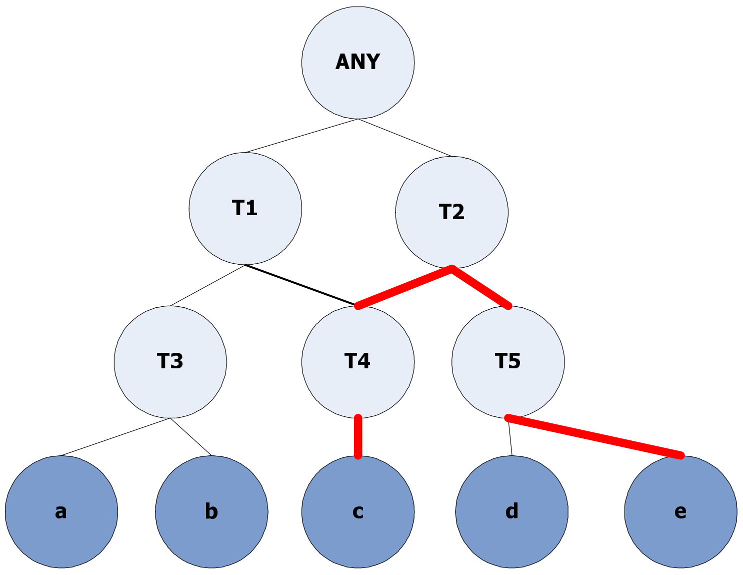

Figure 3.2: Common type assignment for constituents c and e

The next step is to decide how to figure out a common type for two different types using this lattice. Assume that the differences in our match sequence is c and e, so their type sequence are T_4 and T_5, respectively.

At this point out algorithm assigns a common type for two different types, the root of the sub tree formed by the shortest path from leaf c to leaf e, by this method we infer the most specialized type information for these two types.

Turning back to our example, Figure 3.2 shows the shortest path from *node* c to *node* e. Here the root belonging to sub tree is T_2, so the common type is assigned as T_2.

There is another point for type assignment, assume that examples are like in 3.6 will have the match sequence as shown in 3.7.

$$\begin{aligned} \text{I am go +PROG} &\leftrightarrow \text{git+PROG+1SG} \\ \text{I come +PAST} &\leftrightarrow \text{gel+PAST+1SG} \end{aligned} \tag{3.6}$$

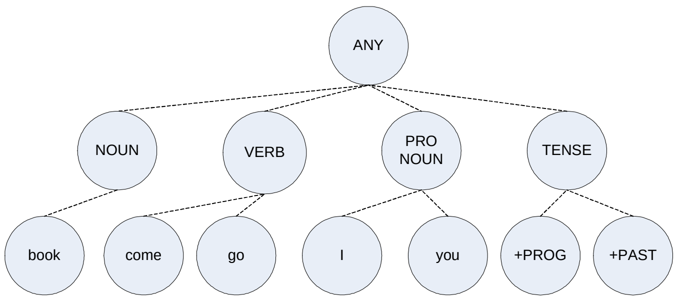

Figure 3.3: Sample English lattice for the examples 3.6

$$\begin{aligned} &\text{I (am go +PROG,come +PAST)} \leftrightarrow \\ &\quad \text{(git +PROG,gel +PAST) +1SG} \end{aligned} \tag{3.7}$$

For the English part we have the difference sequences *am go +PROG* and *come +PAST*. As you can see, there are different number of constituents in both sides. We infer the type of a variable by pair-wise searching a type in the lattice. Now the problem is how we will decide the pairs, since there are different number of constituents.

The part of the lattice needed for the examples given in 3.6 is given in Figure 3.3. Since there are 3 constituents in example E_a and 2 constituents in E_b, there will be one empty string insertion for E_b. In the next section I will explain how the empty string will be inserted in detail.

The main lattice structures used for Turkish and English are given in Appendices A and B, respectively.

The categorization for both languages are taken from the morphological analyzers. For Turkish, the morphological analysis operations are done using a Turkish lexicon file implemented for PC-KIMMO, and in English case Xerox's English morphological analysis tools results are kept in the system, and used from an interface in the system.

As Turkish is an agglutinative language, the main categorization is dependent on the affixes, in general the main categories like noun,verb etc. are directly under the root ANY, but for affixes they are grouped according to the main categories that an affix can follow. For example an ACC (accusative) affix can follow nouns and adjectives in Turkish, so this affix is under the category NOUN-SUFFIX and ADJ-SUFFIX. Another significant categorization principle for Turkish is the categorization of the affixes causing derivation. Derivative words are frequently seen type of words in Turkish. Derivation is the process of creating new lexemes from other lexemes by adding a derivational affix. So this kind of affixes are grouped according to the main categories that they can follow and the category of the word that they yield after the derivation process. For example the *"li"* whose lexical equivalent is *ˆDB+Adj+Without* affix in Turkish follows a noun and turns it to an adjective like *ses* to *sesli*, so the *ˆDB+Adj+Without* category's parent is *NOUN-DB-ADJ*.

For English, the lexical categories are structured according to the main categories and a few sub-categories for affixes are created and the affixes are for noun and verb categories.

3.4 Empty String Insertion

If we turn back to our example in 3.7, firstly we will try the all possible places of an empty string can be inserted as in 3.8.

$$\begin{aligned} &(\text{am go +PROG} &,& \quad \epsilon \text{ come +PAST}) \\ &(\text{am go +PROG} &,& \quad \text{come } \epsilon \text{ +PAST}) \\ &(\text{am go +PROG} &,& \quad \text{come +PAST } \epsilon) \end{aligned} \tag{3.8}$$

For every possibility, we calculate the shortest distance of the types pairwise and chose the possibility with the minimum value. This value is called generalization score. The generalization score calculation for the possibilities in 3.8 is shown in 3.9.

$genScore_1$ = minDist(am,ϵ) + minDist(go,come) + minDist(+PROG,+PAST)
$genScore_2$ = minDist(am,come) + minDist(go,ϵ) + minDist(+PROG,+PAST)
$genScore_3$ = minDist(am,come) + minDist(go,+PAST) + minDist(+PROG,ϵ)

$$\begin{aligned} genScore_1 &= 2+2+2=6 \quad (3.9)\\ genScore_2 &= 4+2+4=10\\ genScore_3 &= 4+4+2=10 \end{aligned}$$

Minimum distance between an empty string (ϵ) and any category is always taken as 2. As you can see, the most appropriate possibility is the first one since it has the smallest generalization score. So the induced translation template for the match sequence 3.10, will be as shown in 3.11.

I (am go +PROG,come +PAST) $\leftrightarrow$
(git +PROG,gel +PAST) +1SG (3.10)

$$\text{I } X^1_{null or(am)\,Verb\,Tense} \quad \leftrightarrow \quad Y^1_{Verb\,Tense} \text{ +1SG} \qquad (3.11)$$

Now it is the time for finding the empty matches in differing parts of the match sequence in 3.12.

boy+Noun +Pl be+Verb +Pres +Pl (come+Verb,not+Adv go+Verb) +Prog $\leftrightarrow$
oğlan+Noun +A3pl +Pnon +Nom (gel+Verb +Pos,git+Verb +Neg) +Prog1 +A3pl)

For the English part, the differing constituents are as in 3.13, and they contain unequal number of strings.

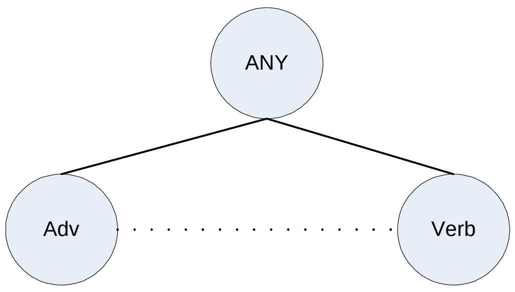

Figure 3.4: A part of English Lattice

$$\begin{gathered}\text{come+Verb} \\ \text{not+Adv go+Verb}\end{gathered} \tag{3.13}$$

All of the possible locations that the empty string can match are to be found, and these possibilities are as in 3.14.

$$\begin{gathered}\text{come+Verb} \rightarrow \text{not+Adv} \quad , \quad \epsilon \rightarrow \text{go+Verb} \\ \epsilon \rightarrow \text{not+Adv} \quad , \quad \text{come+Verb} \rightarrow \text{go+Verb}\end{gathered} \tag{3.14}$$

From the lattice piece in Figure 3.4 the generalization scores for the possibilities are:

$$\begin{aligned} genScore_1 &= minDist(come + Verb, not + Adv) + minDist(empty, go + Verb) \\ genScore_2 &= minDist(empty, not + Adv) + minDist(come + Verb, go + Verb) \end{aligned}$$

$$\begin{aligned} genScore_1 &= 4 + 2 = 6 \\ genScore_2 &= 2 + 2 = 4 \end{aligned}$$

As the second possibility has the smallest generalization score, the correct empty matching is to be done using that possibility. So the differing parts for English becomes:

$$\begin{gathered}(\epsilon\text{,not+Adv})\\ (\text{come+Verb,go+Verb})\end{gathered} \tag{3.15}$$

So the type sequence extraction for the English part is shown in 3.16 and the translation template's English part becomes:

$$\begin{aligned}&\text{nearestParent}(\epsilon\text{,not+Adv}) \text{ nearestParent(come+Verb,go+Verb)}\\ &= \text{nullor(Adv) Verb}\end{aligned} \tag{3.16}$$

$$\text{boy+Noun +Pl be+Verb +Pres +Pl } X^1_{nullor(Adv)\,Verb} \text{ +Prog} \tag{3.17}$$

To extract the Turkish part of the template, the same processes should be applied to Turkish part of the match sequence. The differing parts for Turkish in match sequence 3.12 are:

$$\begin{gathered}\text{gel+Verb +Pos}\\ \text{git+Verb +Neg}\end{gathered} \tag{3.18}$$

The number of strings in the differing parts are equal so we can by-pass the empty string matching part. According to Figure 3.5, we can infer the type sequence for differing part in 3.18, as shown in 3.19.

$$\begin{gathered}\text{nearestParent(gel+Verb,git+Verb) nearestParent(+Pos,+Neg)} =\\ \text{Verb VERB-SUFFIX-SENSE}\end{gathered} \tag{3.19}$$

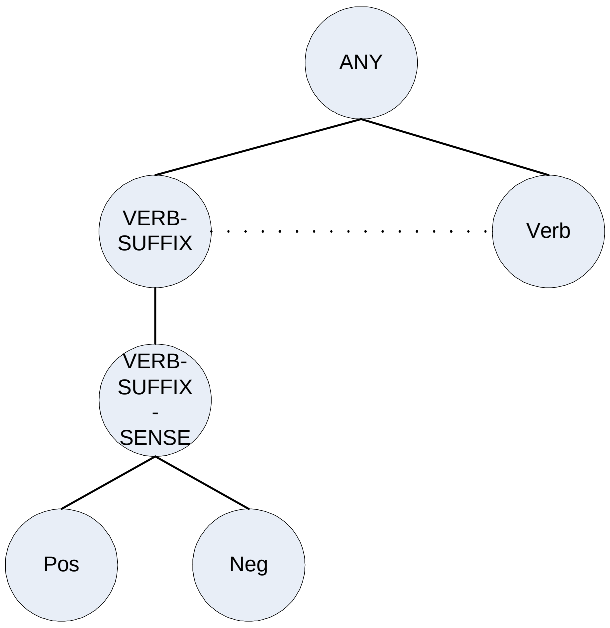

Figure 3.5: Lattice Part for Turkish

So the template's Turkish part becomes:

oğlan+Noun +A3pl +Pnon +Nom $Y^1_{Verb\,VERB-SUFFIX-SENSE}$ +Prog1 +A3pl

Finally, we can say that from examples *boys are going* ↔ *oğlanlar gidiyorlar* and *boys are not coming* ↔ *oğlanlar gelmiyorlar*, we learn the translation template:

boy+Noun +Pl be+Verb +Pres +Pl $X^1_{null or (Adv)\,Verb}$ +Prog ↔
oğlan+Noun +A3pl +Pnon +Nom $Y^1_{Verb\,VERB-SUFFIX-SENSE}$ +Prog1 +A3pl

Along with this template the following facts are also learned:

come+Verb	↔	gel+Verb
not+Adv go+Verb	↔	git+Verb +Neg

3.5 Learning From Learned Templates

As an advantage of type association we can take learning further from extraction from examples. After learning process has been completed, we go on learning with **learning from the learned translation templates**. Actually this process can be seen as *generalization* of two similar translation templates into one template. At the end the newly learned and more general translation template will be able to match with the both of sentences that match with translation templates that are generalized to form it.

Assume that we have the following translation templates learned from the examples:

$$\begin{aligned} \text{at least } X^1_{Num} \text{ book+Noun} &\leftrightarrow \text{en az } Y^1_{Num} \text{ kitap+Noun} \\ \text{at least one+Num } X^1_{Noun} &\leftrightarrow \text{en az bir } Y^1_{Noun} \end{aligned} \tag{3.20}$$

Using these templates we can derive another more generalized template like:

$$\text{at least } X^1_{Num}\ X^2_{Noun} \leftrightarrow \text{en az } Y^1_{Num}\ Y^2_{Noun} \tag{3.21}$$

So the templates in 3.20 are merged to form this new generalized template in 3.21. For generalization the algorithm defined in Table 3.1 is used but there is a difference in finding the match sequences, we feed the match sequence algorithm with replacing the variables with their type sequences as in 3.22.

$$\begin{aligned} \text{at least Num book+Noun} &\leftrightarrow \text{en az Num kitap+Noun} \\ \text{at least one+Num Noun} &\leftrightarrow \text{en az bir Noun} \end{aligned} \tag{3.22}$$

The learning process goes on merging the templates into one more general translation template in 3.21.

In the previous section, I have exemplified the learning process using the examples *boys are going* and *boys are not coming.* Now suppose that in the training set, there exists the examples *girls are going* and *girls are not coming*, the lexical level representation is shown in 3.23.

$$\begin{array}{l} \text{girl+Noun +Pl be+Verb +Pres +Pl come+Verb +Prog} \leftrightarrow \\ \quad \text{kız+Noun +A3pl +Pnon +Nom gel+Verb +Pos +Prog1 +A3pl} \\ \text{girl+Noun +Pl be+Verb +Pres +Pl not+Adv go+Verb +Prog} \leftrightarrow \\ \quad \text{kız+Noun +A3pl +Pnon +Nom git+Verb +Neg +Prog1 +A3pl} \end{array} \qquad (3.23)$$

From the previous example we can doubtlessly say from these two translation examples we can learn the translation template:

$$\begin{array}{l} \text{girl+Noun +Pl be+Verb +Pres +Pl } X^1_{nullor(Adv)\,Verb} \text{ +Prog} \leftrightarrow \\ \quad \text{kız+Noun +A3pl +Pnon +Nom } Y^1_{Verb\,VERB-SUFFIX-SENSE} \text{ +Prog1 +A3pl} \end{array}$$

Now, we have the following two templates:

$$\begin{array}{l} \text{boy+Noun +Pl be+Verb +Pres +Pl } X^1_{nullor(Adv)\,Verb} \text{ +Prog} \leftrightarrow \\ \quad \text{oğlan+Noun +A3pl +Pnon +Nom } Y^1_{Verb\,VERB-SUFFIX-SENSE} \text{ +Prog1 +A3pl} \\ \text{girl+Noun +Pl be+Verb +Pres +Pl } X^1_{nullor(Adv)\,Verb} \text{ +Prog} \leftrightarrow \\ \quad \text{kız+Noun +A3pl +Pnon +Nom } Y^1_{Verb\,VERB-SUFFIX-SENSE} \text{ +Prog1 +A3pl} \end{array}$$

Now if we feed these two templates to the learning algorithm the match sequence will be as in 3.24.

$$\begin{array}{l} \text{(boy+Noun,girl+Noun) +Pl be+Verb +Pres +Pl } X^1_{nullor(Adv)\,Verb} \text{ +Prog} \leftrightarrow \\ \quad \text{(oğlan+Noun,kız+Noun) +A3pl +Pnon +Nom} \\ \qquad Y^1_{Verb\,VERB-SUFFIX-SENSE} \text{ +Prog1 +A3pl} \end{array}$$

As the number of differences is equal to 1, we do not need any prior knowledge to extract a translation template.In both sides the differing parts has equal number of strings so no empty string matching is needed. As types for both sides are the same (*Noun*) the new variable will have the type of *Noun*. So keeping the old variables, the re-learned translation template will be as in 7.1.

$$X^1_{Noun} \text{ +Pl be+Verb +Pres +Pl } X^2_{nullor(Adv)\,Verb} \text{ +Prog} \leftrightarrow$$
$$Y^1_{Noun} \text{ +A3pl +Pnon +Nom } Y^2_{Verb\,VERB-SUFFIX-SENSE} \text{ +Prog1 +A3pl}$$

3.6 Learning Algorithm

So far I have defined the whole learning process in detail. The learning mechanism of the system is summarized by the flowchart seen in Figure 3.6. Firstly, given two translation examples, the match sequences are extracted. If correspondence of the different parts of the sequences can be induces from the facts, these differing parts are replaces with variables. The variables need to carry the type information. To infer the types of the variables, we firstly need the lattice structure of the languages separately, then if the corresponding different parts contain unequal number of constituents the constituents(s) that will match empty strings are found. After the type sequence is inferred it is associated with the variables.

Finally the previous version of the similarity translation template learning algorithm given in Table 2.1 will be modified as in Table 3.1 to achieve the type association to translation templates.

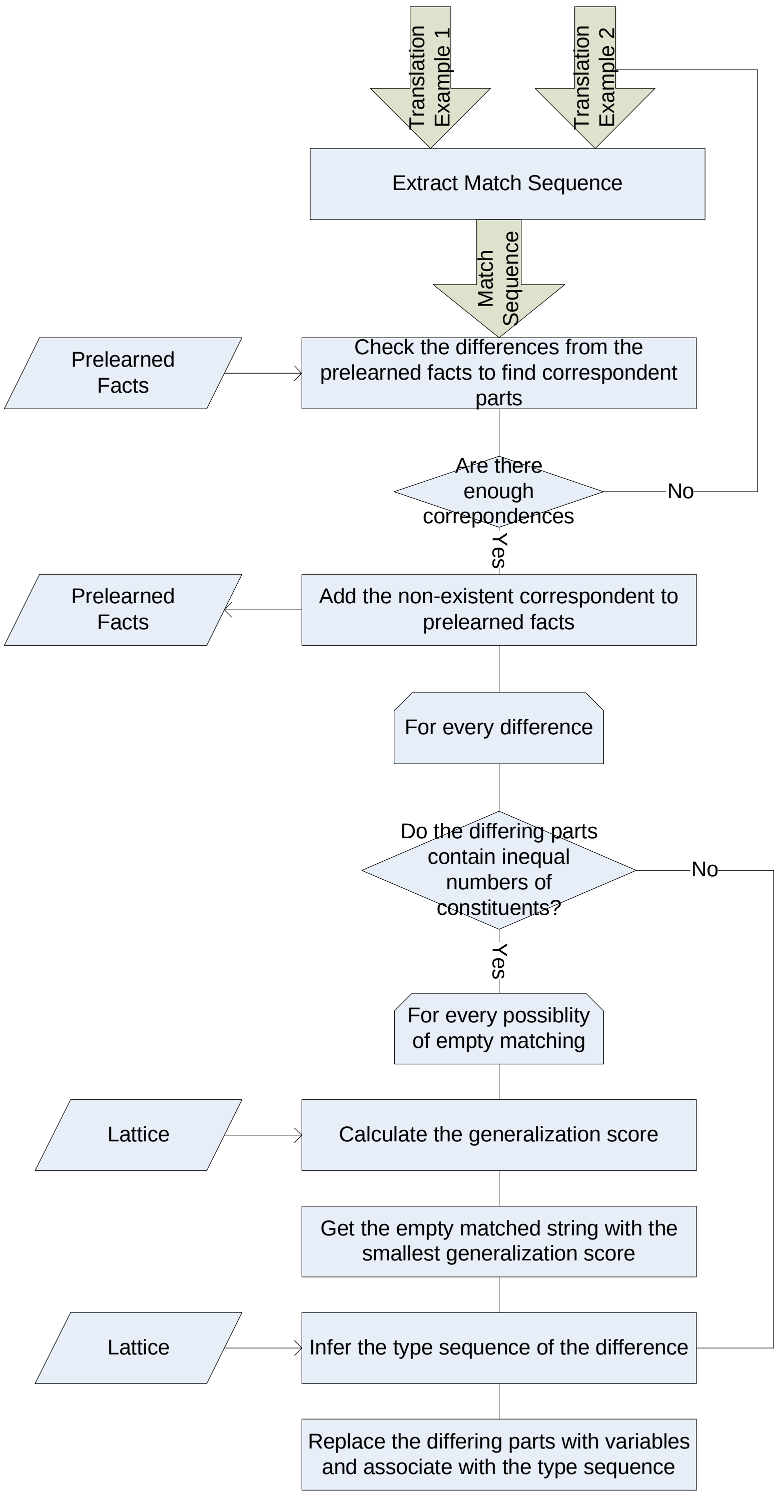
Translation Example 1
Translation Example 2
Extract Match Sequence
Match Sequence
Prelearned Facts
Check the differences from the prelearned facts to find correspondent parts
Are there enough correpondences
No
Yes
Prelearned Facts
Add the non-existent correspondent to prelearned facts
For every difference
Do the differing parts contain inequal numbers of constituents?
No
Yes
For every possiblity of empty matching
Lattice
Calculate the generalization score
Get the empty matched string with the smallest generalization score
Lattice
Infer the type sequence of the difference
Replace the differing parts with variables and associate with the type sequence

Figure 3.6: Flowchart for Learning Algorithm

```
procedure similarityTTWithType($M_{ab}$)
    $M_{ab}$ is the match sequence of examples $E_a$ and $E_b$ and defined as:
    $M_{ab} = S^1_0, D^1_0, ...., D^1_{n-1}, S^1_n \leftrightarrow S^2_0, D^2_0, ...., D^2_{m-1}, S^2_m$
    if( n = m = 1)
        $learnedTranslationTemplate = S^1_0, X^1, S^1_1 \leftrightarrow S^2_0, X^2, S^2_1$
        $learnedFact_1 = D^1_{0,ea} \leftrightarrow D^2_{0,ea}$
        $learnedFact_2 = D^1_{0,eb} \leftrightarrow D^2_{0,eb}$
    else if(n = m >1)
        here we assume that n-1 differences are known previously as facts
        assume that unmatched difference pairs are:
        $((D^1_{k_n,ea}, D^1_{k_n,eb}), (D^2_{l_n,ea}, D^2_{l_n,eb}))$
        replacing all matched pairs in English with $X_{getTypeOf(D_{k_n,e*})^{1..n-1}}$
        replacing all matched pairs in Turkish with $Y_{getTypeOf(D_{l_n,e*})^{1..n-1}}$
        we get match sequence $M_{ab}$ as:
        $M_{ab}WDV$
        $learnedTranslationTemplate = M_{ab}WDV$
            if $X^1 \leftrightarrow Y^1$ and .... and $X^n \leftrightarrow Y^n$
        $learnedFact_1 = D^1_{k_n,ea} \leftrightarrow D^2_{l_n,ea}$
        $learnedFact_2 = D^1_{k_n,eb} \leftrightarrow D^2_{l_n,eb}$
end


procedure getTypeOf(differencePart1, differencePart2)
    if(lengths of differencePart1 and differencePart2 are not equal)
        matchWithEmptyString(differencePart1, differencePart2);
    foreach(token in differencePart1 and differencePart2)
    {
        typeSequence += nearestParentOf(differencePart1[i], differencePart2[i]);
    }
return typeSequence;

procedure matchWithEmptyString(differencePart1, differencePart2)
    foreach(possibility of the empty matches in the shorter string)
    {
        generalizationScores[i] = calculate the total of the lengths
        between the types of possibility[i] and longer string
    }
    return possibility which has the smallest generalization score;
```

Table 3.1: Similarity Translation Template Extraction Algorithm with Type Association

Chapter 4

Confidence Factor Assignment

The learned translation templates are used in translation process and frequently for one source sentence the system produces more than one result, some of these results are produced directly from the training set i.e.: the sentence itself exists in the training data as a fact, doubtlessly this result is the most confident one. Some of the results are produced using more than one translation template, that means the result is generated by substituting one or more variables with the real translations, so this type of results are less confident than the ones generated by the first group.

In our system we want to sort the results according to their *confidence factors*. Confidence factor is the similarity of a translation template within the subset that is formed from the training set elements that matches the translation template. As translation is bidirectional, there are two confidence factors assigned to each translation template, one for the weight from $lang_1$ to $lang_2$, and the other for the reverse direction.

In the previous version of this system, Öz & Çiçekli [22] have used several different methods to assign confidence factors to the translation templates. In this work, a different approach is used while calculating the confidence factors. In [22], along with the confidence factors of the individual translation templates, the concurrent usage possibility of translation templates with the other translation templates are calculated. This method is not covered in the current version of confidence factor assignment.

4.1 Assigning Confidence Factors to Facts

In [22], for weighting the facts (atomic translation templates) Öz & Çiçekli have used a naive method.

Assume that we will calculate the confidence factor of X → Y. In equation 4.1:

$$\mathit{confidence\ factor} = \frac{N1}{N1 + N2} \tag{4.1}$$

- N1 is the number of example pairs where the pairs both contain X and Y as a substring
- N2 is the number of example pairs where the pairs contain X and do not contain Y as a substring

In order to make it clear assume that, the confidence factor of atomic translation template in 4.2 is to be calculated.

$$\text{letter} \rightarrow \text{harf} \tag{4.2}$$

As stated earlier, the confidence factors are assigned according to the training set. In 4.3, the training pairs are given.

$$\begin{aligned}
\text{it is a letter} &\leftrightarrow \text{bu bir mektup +Cop} \\
\text{she write +PAST a letter} &\leftrightarrow \text{bir mektup yaz +PAST +3SG} \\
\text{it is not a letter} &\leftrightarrow \text{o bir harf değil +Cop} \\
\text{letter +s} &\leftrightarrow \text{mektup +PL} \\
\text{boys will not write letter +s} &\leftrightarrow \text{oğlanlar mektup yaz+NEG+FUT+3PL} \\
\text{character} &\leftrightarrow \text{harf}
\end{aligned} \tag{4.3}$$

So for the example 4.2, the confidence factor from $English$ to $Turkish$ is:

- *letter* occurring as *harf* in $lang_2 \rightarrow$ N1 = 1
- *letter* not occurring as *harf* in $lang_2 \rightarrow$ N2 = 4

$$confidence\ factor_1 = \frac{1}{1+4} = 0.2 \tag{4.4}$$

And for confidence factor from $Turkish$ to $English$ for the same example is:

- *harf* occurring as *letter* in $lang_1 \rightarrow$ N1 = 1
- *harf* not occurring as *letter* in $lang_1 \rightarrow$ N2 = 1

$$confidence\ factor_2 = \frac{1}{1+1} = 0.5 \tag{4.5}$$

4.2 Assigning Confidence Factors to Type Associated Translation Templates

Confidence factor assignment to a type associated translation template is different from the correspondent algorithm defined in [22].

$$\textit{confidence factor} = \frac{N1}{N1+N2} \tag{4.6}$$

- N1 is the number of example pairs which are matched by the translation template for both sides
- N2 is the number of example pairs in which one pair matches the translation template while the other does not match

A sentence matches a translation template, when that template can be learned from the substring of the sentence.

Let me explain the calculation method via an example. Assume that we will calculate the confidence factors of translation template 4.7 from the training set in 4.8.

$$\text{at least } X^1_{number\ noun} \leftrightarrow \text{en az } Y^1_{number\ noun} \tag{4.7}$$

$$\begin{aligned} \text{at least one apple} &\leftrightarrow \text{en az bir elma} \\ \text{at least one green apple} &\leftrightarrow \text{en az bir yeşil elma} \\ \text{at least two notebook +s} &\leftrightarrow \text{en az iki defter} \\ \text{at least three beautiful girl +s} &\leftrightarrow \text{en az üç güzel kız} \end{aligned} \tag{4.8}$$

For the confidence factor from *English* to *Turkish*:

- Example pair 1, matches the translation template for both languages so N1 increases by 1
- Example pair 2,does not match the translation template

- Example pair 3,the substring *at least two notebook* and *en az iki defter* matches the template so N1 increases by 1
- Example pair 4,does not match the translation template

$$confidence\ factor_1 = \frac{2}{2+0} = 1.0 \tag{4.9}$$

For the confidence factor from *Turkish* to *English*:

- Example pair 1, matches the translation template for both languages so N1 increases by 1
- Example pair 2,does not match the translation template
- Example pair 3,the substring *at least two notebook* and *en az iki defter* matches the template so N1 increases by 1
- Example pair 4,does not match the translation template

$$confidence\ factor_2 = \frac{2}{2+0} = 1.0 \tag{4.10}$$

The major point of the confidence factor assignment for translation templates is to find a substring in the example pairs that can infer the given translation template.

For types containing *nullor* the type patterns are checked for the all possibilities of the type patterns. Assume that the translation template is as in 4.11. So the type patterns declared in 4.12 will be searched in the examples for this translation template's English part.

$$\text{I } X^1_{nullor(am)\,Verb\,Tense} \text{ to } X^2_{nullor(Det)\,Noun} \leftrightarrow Y^2_{Noun}\text{+YE } Y^1_{Verb\,Tense} \text{ +1SG} \tag{4.11}$$

$$\begin{aligned} \text{am Verb Tense} &\ \ \text{...} \ \ \text{Det Noun} \\ \text{Verb Tense} &\ \ \text{...} \ \ \text{Det Noun} \\ \text{am Verb Tense} &\ \ \text{...} \ \ \text{Noun} \\ \text{Verb Tense} &\ \ \text{...} \ \ \text{Noun} \end{aligned} \tag{4.12}$$

For the completeness of the example that I have used to describe learning algorithm, I will assign confidence factors to the translation templates shown in 4.13 and 4.14.

$$\begin{aligned} \text{come+Verb} &\leftrightarrow \text{gel+Verb} \\ \text{not+Adv go+Verb} &\leftrightarrow \text{git+Verb +Neg} \end{aligned} \tag{4.13}$$

$$\begin{aligned} &X^1_{Noun} \text{ +Pl be+Verb +Pres +Pl } X^2_{nullor(Adv)\,Verb} \text{ +Prog} \leftrightarrow \\ &\quad Y^1_{Noun} \text{ +A3pl +Pnon +Nom } Y^2_{Verb\,VERB-SUFFIX-SENSE} \text{ +Prog1 +A3pl} \end{aligned} \tag{4.14}$$

As stated earlier, the main resource for confidence factors of translation templates is the training set. For the sake of the example, suppose that the training set contains the following examples:

$$\begin{aligned} &\text{girl+Noun +Pl be+Verb +Pres +Pl come+Verb +Prog} \leftrightarrow \\ &\quad \text{kız+Noun +A3pl +Pnon +Nom gel+Verb +Pos +Prog1 +A3pl} \\ &\text{girl+Noun +Pl be+Verb +Pres +Pl not+Adv go+Verb +Prog} \leftrightarrow \\ &\quad \text{kız+Noun +A3pl +Pnon +Nom git+Verb +Neg +Prog1 +A3pl} \\ &\text{boy+Noun +Pl be+Verb +Pres +Pl come+Verb +Prog} \leftrightarrow \\ &\quad \text{oğlan+Noun +A3pl +Pnon +Nom gel+Verb +Pos +Prog1 +A3pl} \\ &\text{boy+Noun +Pl be+Verb +Pres +Pl not+Adv go+Verb +Prog} \leftrightarrow \\ &\quad \text{oğlan+Noun +A3pl +Pnon +Nom git+Verb +Neg +Prog1 +A3pl} \end{aligned} \tag{4.15}$$

The template in 4.13 is an atomic template (fact). As stated earlier the method for assigning confidence factors to facts is naive, the template is searched as a substring in the examples and if both sides contain the template's both sides confidence raises, but if the template is contained only in one side of the example confidence deteriorates.

Using the example set in 4.15, the confidence factors for fact in 4.13 is as follows:

$$confidencefactor_{en-tr} = \frac{N1}{N1+N2} = \frac{2}{2+0} = 1.0 \tag{4.16}$$

Confidence factor from English to Turkish is calculated as 1.0, the used source sentences are the examples 1 and 3 in the example set. The same things are done for the other direction (Turkish to English):

$$confidencefactor_{tr-en} = \frac{N1}{N1+N2} = \frac{2}{2+0} = 1.0 \tag{4.17}$$

Now I will demonstrate how the confidence factor of 4.14 will be calculated. To compute the confidence factor of a translation template, the same logic is applied as in the fact confidence factor assignment but this time the substring match algorithm is different. For 4.14 we will search for the pattern in 4.18.

$$\begin{aligned} &\textit{Noun} \text{ +Pl be+Verb +Pres +Pl } \textit{nullor(Adv) Verb} \text{ +Prog} \leftrightarrow \\ &\quad \textit{Noun} \text{ +A3pl +Pnon +Nom } \textit{Verb VERB-SUFFIX-SENSE} \text{ +Prog1 +A3pl} \end{aligned} \tag{4.18}$$

As mentioned in the previous chapter, the *nullor* types in the type sequence, increases the number of the patterns to be searched, as there are two versions of type sequence (one with the type written next to nullor, the other one is the empty type) for each nullor type. As we have only one *nullor* in the pattern, the patterns to be searched will be as in 4.19.

$$\begin{array}{l} \textit{Noun} \text{ +Pl be+Verb +Pres +Pl } \textit{Adv Verb} \text{ +Prog} \leftrightarrow \\ \quad \textit{Noun} \text{ +A3pl +Pnon +Nom } \textit{Verb VERB-SUFFIX-SENSE} \text{ +Prog1 +A3pl} \\ \textit{Noun} \text{ +Pl be+Verb +Pres +Pl } \textit{Verb} \text{ +Prog} \leftrightarrow \\ \quad \textit{Noun} \text{ +A3pl +Pnon +Nom } \textit{Verb VERB-SUFFIX-SENSE} \text{ +Prog1 +A3pl} \end{array} \tag{4.19}$$

So while looking for a pattern in an example, the main point is the translation template's variable or constant information. In other words, while iterating over an example, the suitability with the translation template is decided whether they have the same string(s) or they have the same type of string(s), in constant and variable case, respectively.

So when seeking for the first pattern in 4.19 in first example of 4.15, as the first token is a variable we will search for a token in the example with type *Noun*. Token is found in the first position, so the process goes on with the next token of the template, as the strings *+Pl be+Verb +Pres +Pl* are all constants, in the example next to the constituent with the type *Noun* these strings are tried to be matched and the example qualifies our expectation. Similarly for the second variable we seek the type sequence *Adv Verb*, but this time the example does not meet our expectation as it has the constituent with type *Verb* next to the constant strings instead of *Adv*. The type matching algorithm returns false for the English part, but if we apply the same steps to the Turkish part the algorithm returns true. For the second pattern, both sides of the example will match the template so this example is said to prove the translation template. When searching for sentences to match the templates containing *nullor* types, the sentence is said to match, when one of the patterns fits the sentence.

So if we calculate the confidence factors for 4.14, the results are:

$$confidencefactor_{en-tr} = \frac{N1}{N1 + N2} = \frac{4}{4 + 0} = 1.0 \tag{4.20}$$

$$confidencefactor_{tr-en} = \frac{N1}{N1 + N2} = \frac{4}{4 + 0} = 1.0 \tag{4.21}$$

The whole confidence factor assignment algorithm is defined in Table 4.1.

So far, I have put the extraction of type associated translation templates across. After the templates are learned, the calculation of confidence factors of learned templates are described. So the system now yields the floor to translate sentences using these templates. In the next chapter translation process within system architecture will be given.

```
procedure confidenceFactor(template)
    foreach(example pair in training set)
    {
        valueForLang1 = matches(example's $lang_1$ part, template's $lang_1$ part);
        valueForLang2 = matches(example's $lang_2$ part, template's $lang_2$ part);
        if(valueForLang1 = true and valueForLang2 = true)
            //only the $lang_1$ and $lang_2$ parts match
            N1 = N1+1;
        if(valueForLang1 = true and valueForLang2 = false)
            //only the $lang_1$ part matches
            $N2_{lang_1} = N2_{lang_1} + 1;$
        if(valueForLang1 = false and valueForLang2 = true)
            //only the $lang_2$ part matches
            $N2_{lang_2} = N2_{lang_2} + 1;$
    }
    $confidence\ factor_{lang_1} = N1\ /\ (N1 + N2_{lang_1})$
    $confidence\ factor_{lang_2} = N1\ /\ (N1 + N2_{lang_2})$
end


procedure matches(sentence, template)
    foreach(pattern of template ($template_{length} = n$))
    {
        foreach(substring S of length n)
        {
            flag = true;
            if(pattern[i] is variable)
            {
                if(type of S[i] and pattern[i] are different)
                    flag = false;
            }
            else if(S[i] and pattern[i] are different)
                flag = false;
        }
        if(flag = true)
            return flag;
        i = i+1;
    }
    return flag;
```

Table 4.1: Confidence Factor Assignment to Type Associated Translation Templates

Chapter 5

System Architecture

In this chapter, the whole architecture of the system will be described in detail. The system is developed with Java programming language (JDK 1.5) using Eclipse 3.1 integrated development environment.

The components of the system is shown in Figure 5.1. The system is composed of two main components **learning** and **translation**. Firstly the learning component is fed with bilingual translation examples and lattice structures for languages. The learning component produces type associated translation templates.

The translation component takes this learned templates as rules. When a sentence is given in $language_1$ that sentence is fed to the morphological analyzer belonging to $language_1$, and combinations of all results are generated and given as input to the translation component, here the matching templates that are to be used in translation, are found and sentence is generated in lexical form in $language_2$, the lexical results are fed into morphological generator of $language_2$ and all possible results are composed. Results are given in decreasing order of confidence factors. The same process is valid for reverse direction.

Our system uses English as $language_1$ and Turkish as $language_2$. We make use of training files containing Turkish-English translation pairs. These pairs are

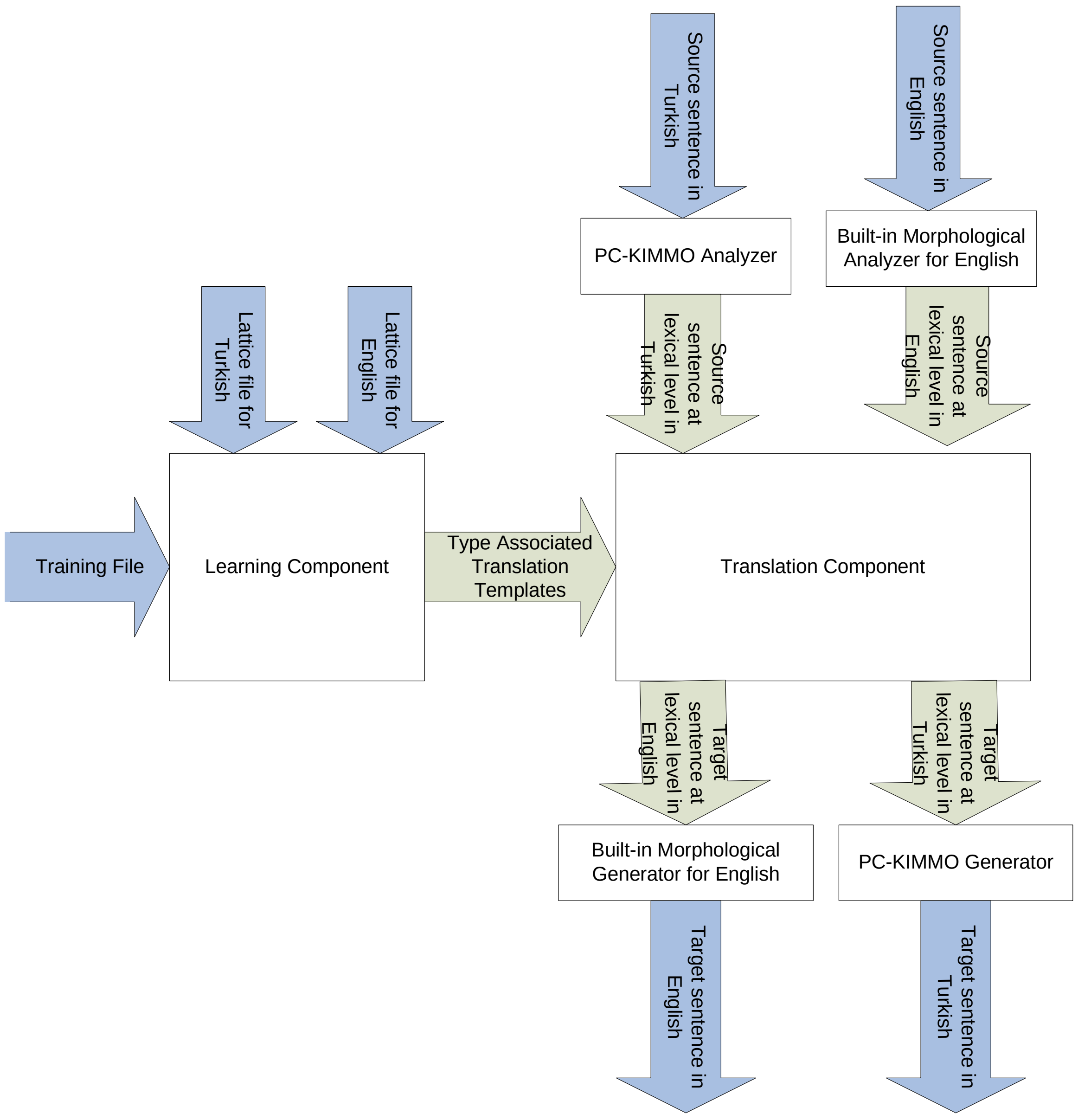
Source sentence in Turkish
Source sentence in English
PC-KIMMO Analyzer
Built-in Morphological Analyzer for English
Lattice file for Turkish
Lattice file for English
Source sentence at lexical level in Turkish
Source sentence at lexical level in English
Training File
Learning Component
Type Associated Translation Templates
Translation Component
Target sentence at lexical level in English
Target sentence at lexical level in Turkish
Built-in Morphological Generator for English
PC-KIMMO Generator
Target sentence in English
Target sentence in Turkish

Figure 5.1: Components of the System

stored in their lexical levels. So before giving the sample run for the learning component, I will describe the storage of examples in the next section.

5.1 Storage of Examples

In 5.1 the storage of the examples *boys are coming* ↔ *oğlanlar geliyorlar* and *boys are not going* ↔ *oğlanlar gitmiyorlar* is shown. For English sentences the sentences are analyzed using a built-in morphological analyzer whose category list is taken from Xerox's English Morphological Analysis, and for Turkish ones sentences are analyzed using Turkish morphological analyzer implemented in PC-KIMMO environment.

As you can see in 5.1, the affixes are separated, this is done in order to learn more specific templates from the examples. Some kind of affixes and some part-of-speech tag of the words are not separated among to the specification level.

boy+Noun +Pl be+Verb +Pres +Pl come+Verb +Prog ↔
  oğlan+Noun +A3pl +Pnon +Nom gel+Verb +Pos +Prog1 +A3pl
boy+Noun +Pl be+Verb +Pres +Pl not+Adv go+Verb +Prog ↔
  oğlan+Noun +A3pl +Pnon +Nom git+Verb +Neg +Prog1 +A3pl (5.1)

So we can say that examples are stored with their type information so whenever we need type of a word we extract it from the word itself like in 5.2.

I+Pron+Pers ↔ type is: *Pron* (5.2)
+Nom ↔ type is: *Nom*

5.2 Lattice Structure for Languages

The lattice structure for Turkish and English are given in Appendix A and B, respectively. In these structures, the root is always the type *ANY*.

The Turkish lattice contains total number of 101 categories, and to arrange these categories in a lattice structure 39 main categories are added. A big part of these main categories arranges the structure for affixes. The affixes are categorized in 4 levels, so the deepest level of the lattice is 4.

The English lattice is not so crowded as Turkish lattice, as this language does not need so many categories for affixes required by an agglutinative language. This lattice contains 46 categories and 19 main categories to arrange the lattice. And the deepest level in the lattice is 3.

5.3 Performance Analysis of Learning Component

Before giving a sample run for the learning component, I will give the time measurements of the learning component for several training sets.

The running time measures of the learning components with different number of translation pairs (N) is given in Table 5.1. In this table first column gives the time measure per iteration, learning component iterates over the example set till it learns no more translation templates. Iterating till no extra translation template learning provides to opportunity to the new templates to be learned. Last three columns show the number of templates learned from examples, learned from learned translation templates and total number of translation templates learned, respectively.

Learning component consumes most of the time where computations are done according to the possible combinations like in feeding match sequences that do not have equal number of differences on both sides, calculating the string that matches with empty string and lastly computing confidence factor of type sequences including *nullor*.

5.4 Sample Run of the Learning Component

After giving the example storage and lattice details of the system. I can now go on with giving a sample run of the whole learning component.

The training file seen in Figure 5.2, is fed with the lattices described in the appendices.

With these translation pairs, the learning component produces the translation templates as in Figures 5.3, 5.4 and 5.5.

As seen from the learned templates, the structure of a translation template is like as:

tt([English translation template],[Turkish translation template],[correspondence list of variables],[type sequence for English variable, type sequence for corresponding Turkish variable],[confidence factor for English, confidence factor for Turkish])

For atomic templates, fields related with variables (correspondence and type) are left empty. For translation templates the number of fields related with variables are related with variable count, in other words if there are two variables in a translation template, there should be two correspondence lists and two type sequence lists.

5.4.1 Incremental Learning

So far we have learned from a single training file, after the learning process has been completed we translate the incoming sentences according to the templates generated by that training file. But what if we want to expand the scope of the

Table 5.1: Performance Measures of Learning Component

N	Time per Iteration	Number Of Iterations	Learning Time	Re-learning Time	Confidence Factor Assignment Time	# Templates from examples	# Templates from templates	# Templates (total)
970	554	4	2218.020	243.920	1015.430	2940	169	3109
100	1.776	3	5.328	0.591	42.341	316	18	334
50	0.445	2	0.891	0.15	13.059	167	9	176
20	0.111	2	0.221	0.1	1.712	44	0	44

1 train_pair([it+Pron+Pers,+NomObl,+3P,+Sg,be+Verb,+Pres,+3sg,brown+Adj,car+Noun,+Sg],[o+Pron,+A3sg,+Pnon,+Nom, kahverengi+Adj,araba+Noun,+A3sg,+Pnon,+Nom]). **[it is brown car , o kahverengi araba]**
2 train_pair([it+Pron+Pers,+NomObl,+3P,+Sg,be+Verb,+Pres,+3sg,green+Adj,car+Noun,+Sg],[o+Pron,+A3sg,+Pnon,+Nom,yesil+Adj,araba+Noun,+A3sg,+Pnon,+Nom]). **[it is green car , o yesil araba]**
3 train_pair([it+Pron+Pers,+NomObl,+3P,+Sg,be+Verb,+Pres,+3sg,I+Pron+Pers,+Gen,+1P,+Sg,car+Noun,+Sg],[o+Pron,+A3sg,+Pnon, +Nom,ben+Pron,+A1sg,+Pnon,+Gen,araba+Noun,+A3sg,+P1sg,+Nom]). **[it is my car , o benim arabam]**
4 train_pair([at+Prep,least+Adv,one+Num+Card,letter+Noun,+Sg],[en+Adverb,+AdjMdfy,az+Adj,bir+Num+Card,mektup+Noun,+A3sg, +Pnon,+Nom]). **[at least one letter , en az bir mektup]**
5 train_pair([at+Prep,least+Adv,two+Num+Card,notebook+Noun,+Pl],[en+Adverb,+AdjMdfy,az+Adj,iki+Num+Card,defter+Noun, +A3sg,+Pnon,+Nom]). **[at least two notebooks , en az iki defter]**
6 train_pair([at+Prep,most+Adv,one+Num+Card,letter+Noun,+Sg],[en+Adverb,+AdjMdfy,çok+Adj,bir+Num+Card,harf+Noun,+A3sg, +Pnon,+Nom]). **[at most one letter , en çok bir harf]**
7 train_pair([at+Prep,most+Adv,one+Num+Card,letter+Noun,+Sg],[en+Adverb,+AdjMdfy,çok+Adj,bir+Num+Card,mektup+Noun, +A3sg,+Pnon,+Nom]). **[at most one letter , en çok bir mektup]**
8 train_pair([at+Prep,most+Adv,three+Num+Card,notebook+Noun,+Pl],[en+Adverb,+AdjMdfy,çok+Adj,üç+Num+Card, defter+Noun,+A3sg,+Pnon,+Nom]). **[at most three notebooks , en çok üç defter]**
9 train_pair([blue+Adj],[mavi+Adj]). **[blue , mavi]**
10 train_pair([I+Pron+Pers,+Gen,+1P,+Sg],[ben+Pron,+A1sg,+Pnon,+Gen]). **[my , benim]**

Figure 5.2: Sample Training File

```
tt([[it+Pron+Pers,+NomObl,+3P,+Sg,be+Verb,+Pres,+3sg,brown+Adj,car+Noun,+Sg]],[[o+Pron,+A3sg,+Pnon,+Nom,kahverengi+Adj,
        araba+Noun,+A3sg,+Pnon,+Nom]],[],[]).[ 1.0 , 1.0 ] learned from 1-2
tt([[it+Pron+Pers,+NomObl,+3P,+Sg,be+Verb,+Pres,+3sg,green+Adj,car+Noun,+Sg]],[[o+Pron,+A3sg,+Pnon,+Nom,yesl+Adj,araba+Noun
        ,+A3sg,+Pnon,+Nom]],[],[] ).[1.0 , 1.0 ] learned from 1-2
tt([[brown+Adj]],[[kahverengi+Adj]],[],[]).[ 1.0 , 1.0 ] learned from 1-2
tt([[green+Adj]],[[yesil+Adj]],[],[]).[ 1.0 , 1.0 ] learned from 1-2
tt([[it+Pron+Pers,+NomObl,+3P,+Sg,be+Verb,+Pres,+3sg,I+Pron+Pers,+Gen,+1P,+Sg,car+Noun,+Sg]],[[o+Pron,+A3sg,+Pnon,+Nom,
        ben+Pron,+A1sg,+Pnon,+Gen,araba+Noun,+A3sg,+P1sg,+Nom]],[],[]).[ 1.0 , 1.0 ] learned from 1-3
tt([[at+Prep,least+Adv,one+Num+Card,letter+Noun,+Sg]],[[en+Adverb,+AdjMdfy,az+Adj,bir+Num+Card,mektup+Noun,+A3sg,
        +Pnon,+Nom]],[],[]).[ 1.0 , 1.0 ] learned   from 1-4
tt([[at+Prep,least+Adv,two+Num+Card,notebook+Noun,+Pl]],[[en+Adverb,+AdjMdfy,az+Adj,iki+Num+Card,defter+Noun,+A3sg,
        +Pnon,+Nom]],[],[]).[ 1.0 , 1.0 ] learned from 1-5
tt([[at+Prep,most+Adv,one+Num+Card,letter+Noun,+Sg]],[[en+Adverb,+AdjMdfy,çok+Adj,bir+Num+Card,harf+Noun,+A3sg,
        +Pnon,+Nom]],[],[]).[ 0.5 , 1.0 ] learned   from 1-6
tt([[at+Prep,most+Adv,one+Num+Card,letter+Noun,+Sg]],[[en+Adverb,+AdjMdfy,çok+Adj,bir+Num+Card,mektup+Noun,+A3sg,
        +Pnon,+Nom]],[],[]).[ 0.5 , 1.0 ] learned from 1-7
tt([[at+Prep,most+Adv,three+Num+Card,notebook+Noun,+Pl]],[[en+Adverb,+AdjMdfy,çok+Adj,üç+Num+Card,defter+Noun,
        +A3sg,+Pnon,+Nom]],[],[]).[ 1.0 , 1.0 ] learned from 1-8
tt([[blue+Adj]],[[mavi+Adj]],[],[]).[ 1.0 , 1.0 ] learned from 1-9
```

Figure 5.3: Translation Templates Learned from Sample Training File 1 of 3

```
tt([[I+Pron+Pers,+Gen,+1P,+Sg]],[[ben+Pron,+A1sg,+Pnon,+Gen]],[],[]).[ 1.0 , 1.0 ] learned from 1-10
tt([[one+Num+Card,letter+Noun,+Sg]],[[bir+Num+Card,mektup+Noun]],[],[]).[ 0.6666667 , 1.0 ] learned from 4-5
tt([[one+Num+Card,letter+Noun,+Sg]],[[bir+Num+Card harf+Noun]],[],[]).[ 0.33333334 , 1.0 ] learned from 6-8
tt([[two+Num+Card,notebook+Noun,+Pl]],[[iki+Num+Card,defter+Noun]],[],[]).[ 1.0 , 1.0 ]learned from 4-5
tt([[least+Adv]],[[az+Adj]],[],[]).[ 1.0 , 1.0 ] learned from 4-7
tt([[most+Adv]],[[çok+Adj]],[],[]).[ 1.0 , 1.0 ] learned from 4-7
tt([[least+Adv,one+Num+Card,letter+Noun,+Sg]],[[az+Adj,bir+Num+Card,mektup+Noun]],[],[]).[ 1.0 , 1.0 ] learned from 4-8
tt([[most+Adv,three+Num+Card,notebook+Noun,+Pl]],[[çok+Adj,üç+Num+Card,defter+Noun]],[],[]).[ 1.0 , 1.0 ] learned from 4-8
tt([[least+Adv,two+Num+Card,notebook+Noun,+Pl]],[[az+Adj,iki+Num+Card,defter+Noun]],[],[]).[ 1.0 , 1.0 ] learned from 5-6
tt([[most+Adv,one+Num+Card,letter+Noun,+Sg]],[[çok+Adj,bir+Num+Card,harf+Noun]],[],[]).[ 0.5 , 1.0 ] learned from 5-6
tt([[most+Adv,one+Num+Card,letter+Noun,+Sg]],[[çok+Adj bir+Num+Card mektup+Noun]],[],[]).[ 0.5 , 1.0 ] learned from 5-7
```

Figure 5.4: Translation Templates Learned from Sample Training File 2 of 3

```
tt([[least+Adv,two+Num+Card]],[[az+Adj,iki+Num+Card]],[],[]).[ 1.0 , 1.0 ] learned from 5-8
tt([[most+Adv,three+Num+Card]],[[çok+Adj,üç+Num+Card]],[],[]).[ 1.0 , 1.0 ] learned from 5-8
tt([[three+Num+Card,notebook+Noun,+Pl]],[[üç+Num+Card,defter+Noun]],[],[]).[ 1.0 , 1.0 ] learned from 6-8
tt([[it+Pron+Pers,+NomObl,+3P,+Sg,be+Verb,+Pres,+3sg],1,[car+Noun,+Sg]],[[o+Pron,+A3sg,+Pnon,+Nom],1,
            [araba+Noun,+A3sg,+Pnon,+Nom]],[1,1],[Adj ,Adj ]).[1.0 , 1.0] learned from 1-2
tt([[at+Prep,least+Adv],1],[[en+Adverb,+AdjMdfy,az+Adj],1,[+A3sg,+Pnon,+Nom]],[1,1],[Num Noun VERB-SUF-COUNT ,Num Noun ],).
            [ 1.0 , 1.0]learned from 4-5
tt([[at+Prep],1,[one+Num+Card,letter+Noun,+Sg]],[[en+Adverb,+AdjMdfy],1,[bir+Num+Card,mektup+Noun,+A3sg,+Pnon,+Nom]],[1,1],
            [Adv ,Adj ],).[ 0.6666667 ,1.0] learned from 4-7
tt([[at+Prep],1],[[en+Adverb,+AdjMdfy],1,[+A3sg,+Pnon,+Nom]],[1,1],[Adv Num Noun VERB-SUF-COUNT ,Adj Num Noun ]).
            [ 1.0 , 1.0] learned from 4-8
tt([[at+Prep],1,[notebook+Noun,+Pl]],[[en+Adverb,+AdjMdfy],1,[defter+Noun,+A3sg,+Pnon,+Nom]],[1,1],[Adv Num ,Adj Num ]).
            [ 1.0 , 1.0] learned from 5-8
tt([[at+Prep,most+Adv],1],[[en+Adverb,+AdjMdfy,çok+Adj],1,[+A3sg,+Pnon,+Nom]],[1,1],[Num Noun VERB-SUF-COUNT ,Num Noun
            ]). [ 1.0 , 1.0] learned from 6-8
tt([[at+Prep],1,2],[[en+Adverb,+AdjMdfy],1,2,[+A3sg,+Pnon,+Nom]],[1,1],[2,2],[Adv ,Adj ],[Num Noun VERB-SUF-COUNT ,Num Noun
            ]). [ 1.0 , 1.0] learned again from translation template 2-6
```

Figure 5.5: Translation Templates Learned from Sample Training File 3 of 3

templates. In the older version, the training file is expanded with the proper examples and the learning process is repeated, this would be time consuming.

If we want to add new examples to the learned file, we execute the learning component with *merge* option. Merge option loads the learned templates (as previous knowledge) and learns from the new file with that prior knowledge and produces a single rule file which contains the previous learned translation templates and the new ones.

We have used an example set of 970 examples for learning and after merged this learned templates with nearly 1000 words containing dictionary like training file. This option makes the system to translate the sentence templates in the first training set with the words included in the dictionary training file.

Now we have acquired the output of learning component as translation templates, in the next section I will exemplify the translation process via these learned pairs.

5.5 Translation Component

So far I have explained the translation template the extraction and confidence factor assignment algorithms in detail, the translation component is the other main component of the system along with learning.

As mentioned in the Example Storage part, the examples are stored therefore learned in their lexical forms, in order to translate the sentence we have to obtain the lexical level representation of the sentence. In the following section, I will explain the details of the morphological analysis operations.

5.5.1 Morphological Analysis Operations

Morphological analysis and generation arranges the input and output of the translation component. As our system is making translations between Turkish and English, we have used the morphological analyzer for these languages.

For Turkish, we have used the PC-KIMMO system integrated into our system via interprocess communication. The rule and lexicon files written for Turkish are loaded at the beginning and when a sentence is wanted to be translated, the sentence is recognized by the PC-KIMMO system and all combinations of lexical representations are listed to the user, to choose the correct combination (disambiguate the different types of a word). After the translation, again the system is wanted to synthesize the generated sentence and the result is given again in the surface form.

For English, the lexical level representations of Online Xerox Morphological Analyzer tool is used, all lexical level representations of encountered words are kept and this data is used for both recognition and synthesize processes as in Turkish part.

5.5.2 Matching Phase

The heart of the translation component is the matching (finding appropriate templates for a sentence) part.

In order to translate the sentence, we have used a modified version of the Earley Parsing algorithm.

5.5.2.1 Earley Parser

The Earley parser is a type of parser mainly used for parsing in computational linguistics. The algorithm uses dynamic programming. Since Earley parser makes use of a chart to eliminate backtracking, it is classified as a chart parser. This feature of the parser makes it distinguished from the other conventional parsing algorithms as it eliminates the reparsing problem [17]. It parses the sentence, according to the grammar given as input.

Before going on with the details of the Earley parser, let me briefly describe the notation that it uses called *dot notation.* The notation in 5.3 means that in the sentence A B C, the parser has already parsed A, now waits for B and C.

$$\mathrm{S} \rightarrow \mathrm{A} \; \textbf{.} \; \mathrm{B} \; \mathrm{C} \tag{5.3}$$

The main operation of an Earley parser is seeking through the N + *1* sets of states (where N is the number of words in the input) and processing the state in its set in order. [17]

The algorithm is composed of three main steps:

- **Prediction:** Adds new states to the chart, if the examined state contains a variable (non-terminal) just after its dot position.
- **Scanning:** Enhances the dot position if the expected terminal has been detected
- **Completion:** If dot has reached to the end of the state (S $\rightarrow$ A b.) in $chart_k$, then this step goes to $chart_{dotPosition_k}$ and enhances the dot position of states having dots just before S.

It is really hard to understand the parsing algorithm without an example ,so for the sake of clarity I will explain the algorithm using the simple grammar given in 5.4.

Chart 0
$State_0$ = . John comes
(1) γ → . S 0 (starter rule)
(2) S → . NP VP 0 (predicted from 1)
(3) S → . VP 0 (predicted from 1)
(4) NP → . Noun 0 (predicted from 2)
(5) VP → . Verb 0 (predicted from 3)
(6) Noun → . John 0 (predicted from 4)
(7) Verb → . comes 0(predicted from 5)
Chart 1
$State_1$ = John. comes
(8) Noun → John . 0 (scanner from chart(0)(6))
(9) NP → Noun . 0 (completed from chart(0)(4) by 8)
(10) S → NP . VP 0 (completed from chart(0)(2) by 9)
(11) VP → . Verb 1 (predicted by 10)
(12) Verb → . comes 1 (predicted by 11)
Chart 2
$State_2$ = John comes.
(13) Verb → comes . 1(scanner from chart(1)(12))
(14) VP → Verb . 1(completed from chart(1)(11) by 13)
(15) S → NP VP . 0 (completed from chart(1)(10) by 14)
(16) γ → S . 0 (completed from chart(0)(1) by 15)

Table 5.2: Earley Parser Demo

$$\begin{aligned} \text{S} &\rightarrow \text{NP VP} \\ \text{S} &\rightarrow \text{VP} \\ \text{NP} &\rightarrow \text{Noun} \\ \text{Noun} &\rightarrow \text{John} \\ \text{VP} &\rightarrow \text{Verb} \\ \text{Verb} &\rightarrow \text{comes} \end{aligned} \tag{5.4}$$

Suppose that we want to parse the sentence *John comes* using these grammar rules. Firstly the length of the chart will be 3 since the sentence has 2 words.

If the chart(n+1) contains at least one rule with the dot at the end of the rule and encloses the whole sentence in other words start position is 0. As you can

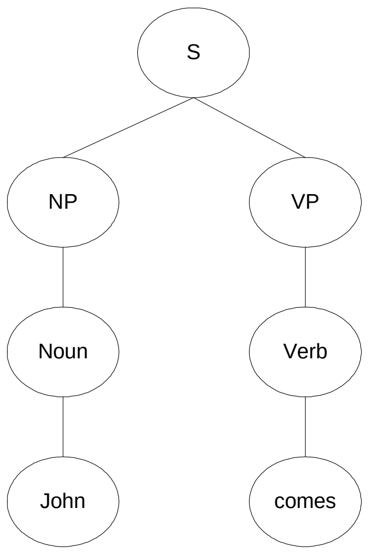

Figure 5.6: Generated Parse Tree for the sentence *John comes*

see from the example that the sentence *John comes* is a legal sentence according to the grammar given in 5.4. Along with the information that the sentence is an appropriate sentence for the grammar, we are able to extract the parse tree as we have the trace information from the completer. The chart generated by the Earley parser is seen in Table 5.2. The parse tree for the sentence is seen in Figure 5.6.

5.5.2.2 Modification of Earley Parser

In this system the Earley parser is used to find the appropriate templates for a given source sentence (matching). To accomplish the matching part some modifications are needed in the Earley parser.

As stated in the previous section the grammatical rules are needed for an Earley parser, in our system the grammatical rules correspond to the translation templates.

First modification is that the structure of the rule that is kept in the chart. As translation is bidirectional process, translation template structure is embedded into the rule structure. For example a translation template as shown in 5.5 is represented as a rules as seen in 5.5.

$$X^1_{Noun} \text{ +Pl be+Verb +Pres +Pl } X^2_{nullor(Adv)\,Verb} \text{ +Prog} \leftrightarrow$$
$$Y^1_{Noun} \text{ +A3pl +Pnon +Nom } Y^2_{Verb\,VERB-SUFFIX-SENSE} \text{ +Prog1 +A3pl}$$

S → S +Pl be+Verb +Pres +Pl S +Prog
[S +A3pl +Pnon +Nom S +Prog1 +A3pl]
correspondence [1,1] [2,2] (5.5)
types [Noun,nullor(Adv) Verb] [Noun,Verb VERB-SUFFIX-SENSE]
confidence factor [1.0,1.0]

These rules are generated in both languages, the example shows a rule that can be used for translation of a sentence from English to Turkish, so the main rule (that the dot will march through) is in English, beyond that the Turkish component of the rule is to be kept with the information which variable corresponds which one, type information of both languages and the confidence factors. For atomic templates the correspondence and type areas are kept empty.

Actually at this level the parser is ready to be used for matching. Let me give an example for the usage. Assume that there are rules as shown in 5.6.

S → S +Pl be+Verb +Pres +Pl S +Prog
[S +A3pl +Pnon +Nom S +Prog1 +A3pl]
correspondence [1,1] [2,2]
types [Noun,nullor(Adv) Verb] [Noun,Verb VERB-SUFFIX-SENSE]
confidence factor [1.0,1.0]
S → man+Noun
[adam+Noun]
correspondence (5.6)
types
confidence factor [1.0,1.0]
S → go+Verb
[git+Verb +Pos]
correspondence
types
confidence factor [1.0,1.0]

We want to translate sentence *men are going* whose lexical representation is given at 5.7.

man+Noun +Pl be+Verb +Pres +Pl go+Verb +Prog (5.7)

So the resulting chart for this example is given in Table 5.3.

To complete the translation process the enclosing rules (with the dot at the end) parse tree is extracted.

To extract the parse tree we simply follow the pointers (data given in parenthesis) of the enclosing rule, and get the rules added by completer:

Rule number *16* is the enclosing rule, it is scanned by rule 14 and this rule is completed by *13*, 13 is scanned by 12 and 12 is predicted by 9, 9 is scanned by 8, 8 is scanned by 7 and 7 is scanned by 6, 6 is scanned by 5 and lastly 5 is completed by *4*. So our parse tree is composed of rules 16, 13 and 4.

Chart 0
$State_0$ = . man+Noun +Pl be+Verb +Pres +Pl go+Verb +Prog
(1) S $\rightarrow$ .S +Pl +be+Verb +Pres +Pl S +Prog 0 (predicted from main)
(2) S $\rightarrow$. man+Noun 0 (predicted from main)
(3) S $\rightarrow$. go+Verb 0 (predicted from main)
Chart 1
$State_1$ = man+Noun . +Pl be+Verb +Pres +Pl go+Verb +Prog
(4) S $\rightarrow$ man+Noun . 0 (scanner from chart(0)(2))
(5) S $\rightarrow$ S . +Pl +be+Verb +Pres +Pl S +Prog 0 (completed from chart(0)(2) by 4)
Chart 2
$State_2$ = man+Noun +Pl . be+Verb +Pres +Pl go+Verb +Prog
(6) S $\rightarrow$ S +Pl . +be+Verb +Pres +Pl S +Prog 0 (scanner from chart(1)(5))
Chart 3
$State_3$ = man+Noun +Pl be+Verb . +Pres +Pl go+Verb +Prog
(7) S $\rightarrow$ S +Pl +be+Verb . +Pres +Pl S +Prog 0 (scanner from chart(2)(6))
Chart 4
$State_4$ = man+Noun +Pl be+Verb +Pres . +Pl go+Verb +Prog
(8) S $\rightarrow$ S +Pl +be+Verb +Pres . +Pl S +Prog 0 (scanner from chart(3)(7))
Chart 5
$State_5$ = man+Noun +Pl be+Verb +Pres +Pl . go+Verb +Prog
(9) S $\rightarrow$ S +Pl +be+Verb +Pres +Pl . S +Prog 0 (scanner from chart(4)(8))
(10) S $\rightarrow$ .S +Pl +be+Verb +Pres +Pl S +Prog 5 (predicted from 9)
(11) S $\rightarrow$. man+Noun 5 (predicted from 9)
(12) S $\rightarrow$. go+Verb 5 (predicted from 9)
Chart 6
$State_6$ = man+Noun +Pl be+Verb +Pres +Pl go+Verb . +Prog
(13) S $\rightarrow$ go+Verb . (scanner from chart(5)(12))
(14) S $\rightarrow$ S +Pl +be+Verb +Pres +Pl S . +Prog 0 (completed chart(5)(12) by 13)
(15) S $\rightarrow$ S . +Pl +be+Verb +Pres +Pl S +Prog 5 (completed chart(5)(12) by 13)
Chart 7
$State_6$ = man+Noun +Pl be+Verb +Pres +Pl go+Verb +Prog .
(16) S $\rightarrow$ S +Pl +be+Verb +Pres +Pl S +Prog . 0 (scanner by chart(6)(14))

Table 5.3: Earley Parser for Matching

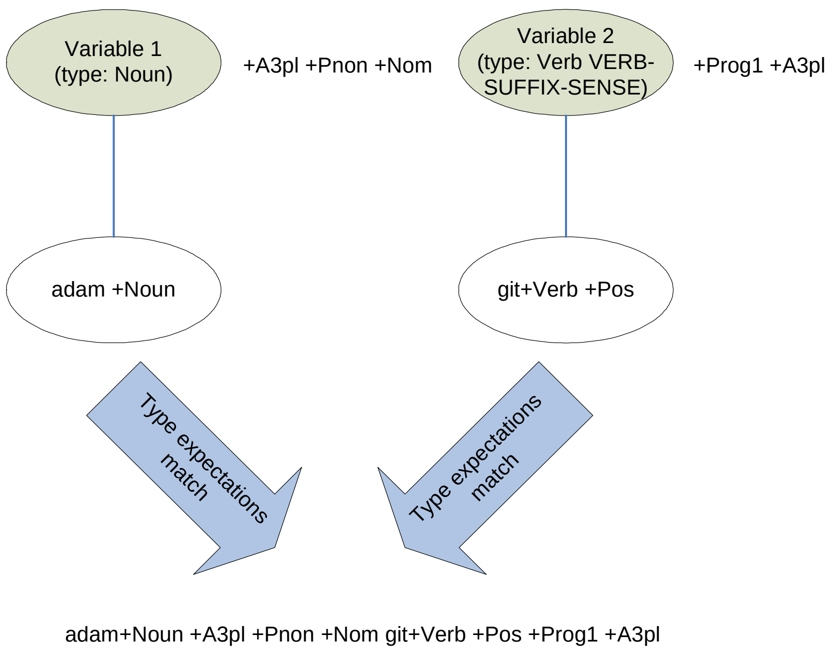

Figure 5.7: Translation Process from Parse Tree

After finding the parse tree, the Turkish correspondent of the rules are filled by the element in the same hierarchy. the remaining process is shown in Figure 5.7.

Confidence factor of the resulting sentence is equal to the multiplication of $confidence\ factor_{src-dest}$ of all rules acted in the translation process, for this example the confidence factor is:

$$confidencefactor_{result} = 1.0 \times 1.0 \times 1.0 = 1.0 \tag{5.8}$$

At the last step type compliance is checked. At this point, to underline the significance of the type association for the templates, suppose that among the rules there exists another rule like in 5.9 [1].

[1]assume the word *go* stands for the game GO in Turkish

$$
\begin{array}{l}
\text{S} \rightarrow \text{go+Verb} \\
\text{[go+Noun]} \\
\text{correspondence} \\
\text{types} \\
\text{confidence factor [1.0,1.0]}
\end{array}
\tag{5.9}
$$

At this point the previous version of the system will locate the template in place of the second variable, but now we are able to eliminate this wrong result.

So far the Earley algorithm is adequate for translation, but when this system is operated on a realistic dataset (containing templates more than 3000), the operation time increases dramatically. In order to get response from the system in a short time, we have to somehow decrease the rules that are enqueued in the chart. We have used two modifications to eliminate useless rules:

- **Look forward and eliminate:** In prediction phase, if the rules are starting with a terminal, the system looks-a-head (looks a head for two strings if not applicable looks a head for one string) and if the following terminals are not equal eliminate the rule. The look-a-head elimination is bounded with 2 because adding a 3 look-a-head elimination did not improve the performance very much and the elimination stopped at 2 look-a-head in order to prevent non-promising elimination controls.
- **Grouping the rules:** Before the Earley parser starts, a pool is composed of the rules starting with the words of the source sentence. And the rules are added as a group to the $Chart0$ and $Chart1$, so the iteration over the first two elements of the chart decreases.

The whole modified Earley parse algorithm is given in Table 5.4.

The chart in Table 5.3 is generated according to the classical Earley parser, Table 5.6 gives the results according to the modified version of the algorithm as you can see among only four grammar rules addition of two useless rules

```
procedure EarleyTranslate(rulePool, sentence)
    rule pool contains rule groups starting with the words of
        the sentence and S
    put all rules starting with S and sentence[0]
    foreach(word in sentence)
    {
        foreach(state in chart[wordIndex])
        {
            if(state is grouped)
            {
                if(rule starts with S)
                    predictor(wordIndex);
                else
                    ungroupIfApplicable(state, wordIndex);
            }
            else if(state is incomplete and element after dot is S)
                predictor(wordIndex);
            else if(state is incomplete and element after dot is a terminal)
                scanner(state,wordIndex);
            else
                completer(state);
        }
    }


procedure predictor(i)
    enqueue(i,rules starting with S as a group);
    enqueue(i,rules starting with sentence[i] as a group);


procedure scanner(state,wordIndex)
    if(sentence[wordIndex] and element after dot in state are equal)
        if(1 or 2 elements further from the wordIndex in state match
            with sentence from point wordIndex)
        enqueue(state's dot position, state dot enhanced by 1);


procedure completer(state)
    foreach(rule in chart[state's start position])
        if(element after dot in rule is S)
            enqueue(state dot position, rule with dot position
                enhanced by 1)
```

Table 5.4: Modified Earley Parser 1 of 2

```
procedure enqueue(index,state)
      add the state to the chart[index] if not previously added


procedure ungroupIfApplicable(index,state)
     if(state's group header and the question's word at the state's
           index are the same)
     {
          scanner(state,index);
     }
```

Table 5.5: Modified Earley Parser 2 of 2 (extra functions)

are prevented, thinking among thousands of rules these modifications makes the system usable as the working time decreases.

In order to show system's output assume that we have fed the unseen example *it is my car* in the training set given in Figure 5.2.

Firstly all possibilities of the output of English morphological analyzer is composed:

it+Pron+Pers +NomObl +3P +Sg be+Verb +Pres +3sg I+Pron+Pers +Gen +1P +Sg car+Noun +Sg

At this point there can be different combinations for the lexical level representation of the sentence, we can send all of the possibilities to the translating component and get the best result.

The system produces only one result:

o+Pron +A3sg +Pnon +Nom ben+Pron +A1sg +Pnon +Gen araba+Noun +A3sg +P1sg +Nom CONFIDENCE: 1.0

o benim arabam

But when type compliance is not checked system produces one more result which is grammatically wrong:

o+Pron +A3sg +Pnon +Nom ben+Pron +A1sg +Pnon +Gen araba+Noun +A3sg +Pnon +Nom CONFIDENCE: 1.0

o benim araba

As you can see both translations have 100% confidence, but the second result is wrong, and its wrongness can only be understood by type checking.

As stated before the confidence of a translation result is calculated by multiplication of all of the translation templates confidence factors used in translation.

At the last step the produced lexical level sentence is fed into the Turkish generator giving the surface form of the result.

Chart 0
$State_0$ = . man+Noun +Pl be+Verb +Pres +Pl go+Verb +Prog
(1) S $\rightarrow$ .S α 0 (predicted from main)
(2) S $\rightarrow$. man+Noun α 0 (predicted from main)
Chart 1
$State_1$ = man+Noun . +Pl be+Verb +Pres +Pl go+Verb +Prog
(3) S $\rightarrow$ man+Noun . 0 (scanner from chart(0)(2))
(4) S $\rightarrow$ S . +Pl +be+Verb +Pres +Pl S +Prog 0 (completed from chart(0)(2) by 3)
Chart 2
$State_2$ = man+Noun +Pl . be+Verb +Pres +Pl go+Verb +Prog
(5) S $\rightarrow$ S +Pl . +be+Verb +Pres +Pl S +Prog 0(scanner from chart(1)(4))
Chart 3
$State_3$ = man+Noun +Pl be+Verb . +Pres +Pl go+Verb +Prog
(6) S $\rightarrow$ S +Pl +be+Verb . +Pres +Pl S +Prog 0(scanner from chart(2)(5))
Chart 4
$State_4$ = man+Noun +Pl be+Verb +Pres . +Pl go+Verb +Prog
(7) S $\rightarrow$ S +Pl +be+Verb +Pres . +Pl S +Prog 0(scanner from chart(3)(6))
Chart 5
$State_5$ = man+Noun +Pl be+Verb +Pres +Pl . go+Verb +Prog
(8) S $\rightarrow$ S +Pl +be+Verb +Pres +Pl . S +Prog 0(scanner from chart(4)(7))
(9) S $\rightarrow$ .S α 5 (predicted from 8)
(10) S $\rightarrow$. go+Verb α 5 (predicted from 8)
Chart 6
$State_6$ = man+Noun +Pl be+Verb +Pres +Pl go+Verb . +Prog
(11) S $\rightarrow$ go+Verb . (scanner from chart(5)(10))
(12) S $\rightarrow$ S +Pl +be+Verb +Pres +Pl S . +Prog 0 (completed chart(5)(10) by 11)
(13) S $\rightarrow$ S . +Pl +be+Verb +Pres +Pl S +Prog 5 (completed chart(5)(10) by 11)
Chart 7
$State_6$ = man+Noun +Pl be+Verb +Pres +Pl go+Verb +Prog .
(14) S $\rightarrow$ S +Pl +be+Verb +Pres +Pl S +Prog . 0 (scanner by chart(6)(12))

Table 5.6: Modified Earley Parser for Matching

Chapter 6

Tests and Evaluation

So far I have explained the system and its components in detail. The performance analysis of the system comes next.

Human intervention was one the most popular techniques to verify the results of machine translation systems. But as we can guess, human verification is very expensive and time-consuming. In [23] Papineni et. al. offers an automatic method for evaluation of these systems: BiLingual Evaluation Understudy (BLEU) method.

6.1 BLEU Method

Bleu is an evaluation method that gives scores to the results that are generated by machine translation systems. It states for BiLingual Evaluation Understudy, and it is capable of evaluating of any bilingual machine translation system independent from language.

As stated in [23], *the closer a machine translation is to a professional human translation, the better it is.* The quality of translation is a number between 0 and 1, and this measurement is a statistical closeness to a set of perfect translations.

The first computational phase of BLEU method is *n-gram* comparison of the candidate sentences. If we take the example from [23]:

Candidate: the the the the the the the.

Reference 1: The cat is on the mat.

Reference 2: There is a cat on the mat.

The machine translation system has produced the absurd result in Candidate. If we compute the n-gram value (taking n as 1) for the candidate according to Reference 1 the unigram value will be 7 / 7, which leads to a high score for that wrong candidate, so we must somehow modify the computation of n-grams.

The best method for modification of n-gram computation is to a word in the reference is exhausted after it has been matched with a word in candidate sentence. To compute the *modified unigram precision*, first the maximum number of times a word occurs in any single reference translation is counted, then the total count of each candidate word is clipped by its maximum reference count. Finally, these clipped counts are added and divided by total number of candidate words.

According to the formula, the modified unigram precision will be 2 / 7 (the clipped words are underlined in the candidate sentence).

The problem is to decide the value n to be used. The experiments in [23] show that the best results are taken when $n = 4$, so when the value of n is greater than 1 the n-gram precisions are calculated for $n = 1..n$ and the geometric mean of these precisions is taken as resulting n-gram precision:

$$p_n = \frac{\sum_{C\in\{Candidates\}} \sum_{n-gram\in C} Count_{clip}(n-gram)}{\sum_{C'\in\{Candidates\}\{} \sum_{n-gram'\in C'} Count(n-gram')} \tag{6.1}$$

Experiments show that the modified n-gram precision for most cases can distinguish a bad translation and a good translation. The n-gram precision penalizes the non-existing words in the reference sentence, at this point Papineni et. al. has added another penalty for the candidate sentences that are longer than the reference sentences. This penalty is called Brevity Penalty and calculated as:

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases} \tag{6.2}$$

Finally the BLEU score is calculated as shown in Equation 6.3, where the positive weights of n-grams w_n sums up to 1.

$$\text{BLEU} = \text{BP} \cdot \exp\,(\sum_{n=1}^{N} w_n \log p_n) \tag{6.3}$$

Let me explain the algorithm with an example, assume that our system has generated the result *the cat is on mat* and the reference sentence is *the cat is on the mat*.

I have chosen n to be 4, so the modified n-gram precisions of the candidate sentence will be:

Precision 1-gram: 1.0

Precision 2-gram: 0.75

Precision 3-gram: 0.6666666666666666

Precision 4-gram: 0.5

And now the weighted precision is to be calculated, with equal weights for $n = 4$ so $w = 0.25$, for i from 1 to n.

$$\text{weighted precision} = (\sum_{n=1}^{N} w_n \log p_n) \tag{6.4}$$

So applying the formula in 6.4:

Weighted Precision: 0.7071067811865475

After this point the Brevity Penalty is calculated according to the equation 6.5.

$$\text{BP} \ = e^{(1-6/5)} = 0.8187307530779819 \tag{6.5}$$

Lastly the BLEU score is the multiplication of the brevity penalty and the weighted precision:

$$\text{BLEU Score} = \text{WP} \times \text{BP} = 0.5789300674674098 \tag{6.6}$$

6.2 Tests

As stated earlier, we have made use a 970 pairs training set for learning, and for testing, 100 unseen examples in the training set is used. The English sentence and its correct Turkish correspondent are given in the test file.

The BLEU score calculation is implemented in Java. I have used $n = 4$ for n-gram precision where the sentence length allows me to do, if the sentence length is shorter than 4 then the total length of the sentence is taken as n. The weight for each n is taken equal, $w_n = 1/n$. The candidate sentence is the first result that the system returns (the sentence having the biggest confidence factor).

The system is tested for the following cases:

- **Experiment 1-Translation negating type constraints:** This experiment is done to underline the enhancement of the system by addition of type constraints. After translation no type checking is done.
- **Experiment 2-Translation using main lattice:** In this experiment, the system is tested in its normal case. The type constraints are composed by the main lattice structure given in appendices.

Experiment	Average BLEU Score
Experiment 1	76 %
Experiment 2	93 %
Experiment 3	88 %
Experiment 4	93 %
Experiment 5	93 %

Table 6.1: BLEU Score Results for Experiments from English to Turkish

- **Experiment 3-Translation using a single node lattice:** This experiment makes use of a lattice containing only *ANY* as root node and all of the lexical categories are bound to directly to that root without any subcategorization. Actually learning from this type of lattice, constraints the number of constituents that a variable can be replaced, instead of the types.

- **Experiment 4-Translation using a single node for affix subcategorization in lattice:** As stated earlier, while designing the lattice especially for Turkish we have taken the affix categorization into account. In this experiment all kinds of suffixes are bound to a single *SUFFIX* node. The remaining part of the categorization is left the same.

- **Experiment 5-Translation using a single node for type specific affix sub-categorization in lattice:** In this lattice structure, the type specific suffixes (adjective suffixes, noun suffixes etc.) are not sub-categorized as in the main lattice structure. Instead only one type-suffix node is used as a parent for all suffixes belonging to this type. For example in the main lattice verb suffixes were divided into two groups like VERB-TENSE and VERB-SENSE, but in the lattice that this experiment has used, does no divide the verb suffixes, instead it binds all of the verb suffixes to VERB-SUFFIX.

The enhancement that the type association has incorporated to the system can be seen from the Table 6.1. Without type constraints the performance of the system is 76%, with the addition of type constraints using the main lattice the system's performance is 93%. By looking at the results of Experiment 2 after the addition of types, the system does not produce wrong results, the deviation from 100% comes from the different translations of the sentences like: *it is a pencil*

Experiment	Average BLEU Score
Experiment 1	75 %
Experiment 2	92.5 %
Experiment 3	85 %
Experiment 4	92.5 %
Experiment 5	92.5 %

Table 6.2: BLEU Score Results for Experiments from English to Turkish without Confidence Factors

can be translated as *o bir kalemdir* and *bir kalemdir*.

In Experiment 3, we have shrank the lattice structure into one node *ANY*, and bound all of the categories to that node. So all the type patterns do not contain a type different from ANY. The difference of this setup from a typeless fashion is that the number of constituents that a variable can replace is restricted by he count of ANYs in the type sequence. Count restriction causes the system performance enhance to 88% from 76%.

As stated above, in experiments 4 and 5, we have changed the categorization of affixes and learned the templates according to these lattices. The result does not deviate from the main structure of the lattice, but these results depends upon the translation pairs. It is obvious that the system performs better under a detailed structure of lattice.

Experiments are repeated for the same experiment sets without sorting the results according to the confidence factors, in other words the first candidate sentence given is the first sentence is the sentence that the translation component has firstly produces. The results are given in Table 6.2.

From the results we can see that scores decrease about 1% for each experiment set, so usage of confidence factors enhances the systems overall performance.

In Table 6.3, the results of the experiments repeated over the same test set in reverse direction, from Turkish to English. Again the deviation of the type associated system from 100% is sourced from equivalent translations like *bir siyah araba* is translated as *one black car* instead of *a black car*.

Experiment	Average BLEU Score
Experiment 1	70.2 %
Experiment 2	86.78 %
Experiment 3	80.3 %
Experiment 4	86.78 %
Experiment 5	86.78 %

Table 6.3: BLEU Score Results for Experiments from Turkish to English

Experiment	Average BLEU Score
Experiment 1	63.3 %
Experiment 2	76.4 %
Experiment 3	70.5 %
Experiment 4	76.4 %
Experiment 5	76.4 %

Table 6.4: BLEU Score Results for Experiments from Turkish to English without Confidence Factors

Table 6.4 shows the performance of the system over the same test from Turkish to English when the confidence factors of the results are ignored. In Turkish to English case confidence factor of a result becomes significant as there is a decrease about 10%.

The same tests are repeated with 10 fold cross validation. The test set is added to the training set and the resulting set is pieced into 10. The results for English to Turkish translations can be seen in Tables 6.5 and 6.6. The results are parallel with the main test set results.

The results for Turkish to English are given in Tables 6.7 and 6.8, again the significance of confidence factor usage is seen as the results decrease without

Experiment	Average BLEU Score
Experiment 1	83.64 %
Experiment 2	95.07 %
Experiment 3	93.47 %
Experiment 4	95.07 %
Experiment 5	95.07 %

Table 6.5: Average BLEU Score Results for Experiments from English to Turkish with Confidence Factors (10-fold cross validation)

Experiment	Average BLEU Score
Experiment 1	82.52 %
Experiment 2	92.91 %
Experiment 3	92.23 %
Experiment 4	92.91 %
Experiment 5	92.91 %

Table 6.6: Average BLEU Score Results for Experiments from English to Turkish without Confidence Factors (10-fold cross validation)

Experiment	Average BLEU Score
Experiment 1	73.48 %
Experiment 2	96.30 %
Experiment 3	89.58 %
Experiment 4	96.30 %
Experiment 5	96.30 %

Table 6.7: Average BLEU Score Results for Experiments from Turkish to English with Confidence Factors (10-fold cross validation)

Experiment	Average BLEU Score
Experiment 1	65.72 %
Experiment 2	93.28 %
Experiment 3	88.01 %
Experiment 4	93.28 %
Experiment 5	93.28 %

Table 6.8: Average BLEU Score Results for Experiments from Turkish to English without Confidence Factors (10-fold cross validation)

Experiment	First 1	First 3	First 5	First 10
Experiment 1	47.2 %	41.1%	11.7%	0%
Experiment 2	86 %	13%	1%	0%
Experiment 3	78 %	20%	2%	0%
Experiment 4	86 %	13%	1%	0%
Experiment 5	86 %	13%	1%	0%

Table 6.9: Correct Result Position for Translations from English to Turkish

Experiment	First 1	First 3	First 5	First 10
Experiment 1	35.4 %	52%	8%	4.6%
Experiment 2	86 %	14%	0%	0%
Experiment 3	61.2 %	26.5%	12.3%	0%
Experiment 4	86 %	14%	0%	0%
Experiment 5	86 %	14%	0%	0%

Table 6.10: Correct Result Position for Translations from Turkish to English

confidence factors.

The performance results from Turkish to English is not as good as the translation performance from English to Turkish, the reason beyond this deterioration is mainly sourced from pronoun differences in Turkish and English like, *o gitti* is translated as *it went*, this translation is correct but the test set waits the result as *he went*, so this type of translations have yielded poor BLEU scores.

Tables 6.9 and 6.10 show the distribution of the correct result position among the generated results. First column means the frequency of the correct result generated at the first location, the second columns stands for the frequencies that the correct result is among the first three results but not in the first position. Third column shows the correct result is in the first five results but not in first three, similarly the last column points that the correct result is in the first ten results but not in first five.

As you can see from the tables 6.9 and 6.10 the addition of the type information has also increased the probability that the system generates the waited result at the top of all the results. These tests are run on the same test set with confidence factors taken into account. Actually the generated results from the system are all correct but not same with the waited one, so the divergence from a 100% comes from this difference.

As a result, the performance results of the system is satisfactory, as the system is capable of eliminating nearly all of the wrong results.

Chapter 7

Conclusion and Future Work

In this thesis, I have presented modified version of translation template extraction algorithm developed by Çiçekli&Güvenir [9]. In this version type information of the constituents are associated with variables in the translation templates. Also I have presented modification of the confidence factor assignment methods given in Öz&Çiçekli [22].

In the previous version of the system, the results were given according to their confidence factors, adding this kind of metric gives the appropriate result at the top of the results, but it does not eliminate the incorrect results. In order to give the best result among the correct results, the confidence factor assignment is re-implemented.

I have used English-Turkish pairs to exemplify the system, but keep in mind that the system is language independent. As I have reported in Chapter 1, there are multiple means for a corpus based machine translation system to use as a bilingual corpus like World Wide Web etc. I have tagged the existing large corpus in their lexical levels and built up another corpus in a fashion of a dictionary to enlarge the scope of the sentences that can be generated by the system. The learning time for this large corpus is considerable but as this process is done once for the system it can be tolerated.

The test results are satisfactory, as we can see that for some translations nearly 90% of the results are eliminated and the remaining ones were all correct.

The system is accurate enough, that it does not generate a wrong result, as long as a semantic behavior does not come into the scene. Assume that we have the following translation template extracted from the examples *I ate the orange* $\leftrightarrow$ *Portakalı yedim* and *I ate the apple* $\leftrightarrow$ *Elmayı yedim*:

$$\text{I eat+PAST the } X^1_{Noun} \;\leftrightarrow\; Y^1_{Noun} \text{ +ACC ye+PAST+1sg}$$

Whenever we want to translate the Turkish sentence *kafa+Noun +ACC yedim* to its English correspondent, the system will generate *I ate the head.* The generated result is *grammatically* correct but *semantically* it is not.

In the future along with types the variables in the translation templates can also be associated with semantic information to eliminate the sentences that are not valid in the language. So the system will not produce any result grammatically and semantically incorrect.

Bibliography

[1] Carabao do-it-yourself machine translation kit. http://ai-depot.com/nolimits/aboutcarabao.htm.

[2] European association for machine translation. http://www.eamt.org/mt.html.

[3] A. Bech. MT from an Everyday User's Point of View. *Proceedings of MT Summit*, pages 98–105, 1997.

[4] P. Brown, S. Della Pietra, V. Della Pietra, F. Jelinek, J. Lafferty, R. Mercer, and P. Roossin. A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85, 1990.

[5] R. Brown. Automated generalization of translation examples. *Proceedings of the 17th conference on Computational linguistics-Volume 1*, pages 125–131, 2000.

[6] R. Brown. Transfer-rule induction for example-based translation. *Proceedings of the Workshop on Example-Based Machine Translation*, 2001.

[7] M. Carl, C. Pease, L. Iomdin, and O. Streiter. Towards a Dynamic Linkage of Example-based and Rule-based Machine Translation. *Machine Translation*, 15(3):223–257, 2000.

[8] I. Cicekli. Learning Translation Templates with Type Constraints. *Machine Translation*, 2007 (in press).

[9] I. Cicekli and H. Güvenir. Learning Translation Templates from Bilingual Translation Examples. *Applied Intelligence*, 15(1):57–76, 2001.

[10] A. Corbí-Bellot, M. Forcada, S. Ortiz-Rojas, J. Pérez-Ortiz, G. Ramírez-Sánchez, F. Sánchez-Martínez, I. Alegria, A. Mayor, and K. Sarasola. An Open-Source Shallow-Transfer Machine Translation Engine for the Romance Languages of Spain. *Proceedings of EAMT*, 2005.

[11] L. Cranias, H. Papageorgiou, and S. Piperidis. A matching technique in Example-Based Machine Translation. *Proceedings of the 15th conference on Computational linguistics-Volume 1*, pages 100–104, 1994.

[12] L. CRANIAS, H. PAPAGEORGIOU, and S. PIPERIDIS. Example retrieval from a translation memory. *Natural Language Engineering*, 3(04):255–277, 2000.

[13] M. Forcada and F. Nolla. TEFATE: Finite-state technologies applied to specialized translation TIC2003-08681-C02.

[14] G. Grefenstette. The World Wide Web as a Resource for Example-Based Machine Translation Tasks. *prostate*, 28:40772.

[15] W. Hutchins. *Machine translation.* Ellis Horwood.

[16] W. Hutchins. The Georgetown-IBM experiment demonstrated in January 1954. *Machine translation: from real users to research. 6th Conference of the Association for Machine Translation in the Americas, AMTA*, pages 102–114, 2004.

[17] D. Jurafsky and J. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* MIT Press, 2000.

[18] K. Mahesh and S. Nirenburg. Meaning Representation for Knowledge Sharing in Practical Machine Translation. *Proc. of the FLAIRS-96 track on information interchange*, 1996.

[19] T. Mitchell. *Machine Learning.* McGraw-Hill Higher Education, 1997.

[20] M. Nagao. A framework of a mechanical translation between Japanese and English by analogy principle. *Proc. of the international NATO symposium on Artificial and human intelligence table of contents*, pages 173–180, 1984.

[21] B. Ørsnes, B. Music, and B. Maegaard. PaTrans: a patent translation system. *Proceedings of the 16th conference on Computational linguistics-Volume 2*, pages 1115–1118, 1996.

[22] Z. Öz and I. Cicekli. Ordering Translation Templates by Assigning Confidence Factors. *Proceedings of AMTA*, 98:51–61, 1998.

[23] K. Papineni, S. Roukos, T. Ward, and W. Zhu. BLEU: a method for automatic evaluation of machine translation. *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 311–318, 2001.

[24] A. Poutsma. Data-Oriented Translation. *Proceedings of the 17th conference on Computational linguistics-Volume 2*, pages 635–641, 2000.

[25] V. Sadler. The Textual Knowledge Bank: Design, Construction, Applications. *Proc. hlernational Workshop on Fundamental Research for the Future Generation of Natural Language Processing (FGNLP)*, pages 17–32.

[26] S. Sato. CTM: an example-based translation aid system. *Proceedings of the 14th conference on Computational linguistics-Volume 4*, pages 1259–1263, 1992.

[27] S. Sato and M. Nagao. Toward memory-based translation. *Proceedings of the 13th conference on Computational linguistics-Volume 3*, pages 247–252, 1990.

[28] H. Somers. EBMT seen as case-based reasoning. *Proceedings of the Workshop on Example-Based Machine Translation, at the MT-Summit, Association for Computational Linguistics, Santiago de Compostela, Spain*, 2001.

[29] H. Somers. An overview of EBMT. *Recent advances in example-based machine translation, Kluwer Academic Publishers, Dordrecht*, 2003.

[30] E. Sumita and H. Iida. Experiments and prospects of Example-Based Machine Translation. *Proceedings of the 29th conference on Association for Computational Linguistics*, pages 185–192, 1991.

[31] E. Sumita, H. Iida, and H. Kohyama. Translating with Examples: A New Approach to Machine Translation. *The Third International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Language*, pages 203–212, 1990.

[32] T. Veale and A. Way. Gaijin: A Bootstrapping Approach to Example-Based Machine Translation. *International Conference, Recent Advances in Natural Language Processing*, pages 239–244, 1997.

[33] H. Watanabe. A similarity-driven transfer system. *Proceedings of the 14th conference on Computational linguistics-Volume 2*, pages 770–776, 1992.

[34] G. Zhao and J. Tsujii. Transfer in Experience-Guided Machine Translation. *Machine Translation Summit VII*, pages 501–508, 1999.

Appendix A

Lattice Structure for Turkish

The lattice structure for Turkish is given in Table A.1, the main categories that come from Turkish PC-KIMMO system is written in normal case whereas the subcategories that are added to arrange the lattice is written in capitals.

Category Name	Parent Category in Lattice
ANY	
AdjMdfy	ANY
Pron	ANY
Noun	ANY
Prop	Noun
ADJ-SUF	ANY
PRON-SUF	ANY
ADVERB-SUF	ANY
VERB-SUF	ANY
VERB-TENSE	VERB-SUF
NOUN-SUF	ADJ-SUF ANY
NOUN-AGREEMENT	NOUN-SUF
VERB-AGREEMENT	VERB-SUF
A3pl	NOUN-AGREEMENT

	VERB-AGREEMENT
A3sg	NOUN-AGREEMENT VERB-AGREEMENT
NOUN-POSSESIVE	NOUN-SUF
Pnon	NOUN-POSSESIVE
P1sg	NOUN-POSSESIVE
P2sg	NOUN-POSSESIVE
P3sg	NOUN-POSSESIVE
P1pl	NOUN-POSSESIVE
P2pl	NOUN-POSSESIVE
P3pl	NOUN-POSSESIVE
NOUN-CASE	NOUN-SUF
Acc	NOUN-CASE
Dat	NOUN-CASE
Loc	NOUN-CASE
Abl	NOUN-CASE
Gen	NOUN-CASE
Ins	NOUN-CASE
Nom	NOUN-CASE
Adj	ANY
Conj	ANY
Verb	ANY
Pres	VERB-TENSE
Past	VERB-TENSE
Aor	VERB-TENSE
Fut	VERB-TENSE
Narr	VERB-TENSE
Prog1	VERB-TENSE
Prog2	VERB-TENSE
Neces	VERB-TENSE
Opt	VERB-TENSE
Imp	VERB-TENSE

Desr	VERB-TENSE
Cond	VERB-TENSE
Cop	VERB-TENSE
ASPECT*PR-CONT	VERB-TENSE
A1sg	VERB-AGREEMENT
A2sg	VERB-AGREEMENT
A1pl	VERB-AGREEMENT
A2pl	VERB-AGREEMENT
VERB-SENSE	VERB-SUF
Pos	VERB-SENSE
Neg	VERB-SENSE
Punc	ANY
Interj	ANY
NUMBER	ANY
Num	NUMBER
Num+Card	NUMBER
Num+Ord	NUMBER
Num+Dist	NUMBER
NOUN-DB	ANY
POSTP	ANY
Postp+PCNom	POSTP
Postp+PCAbl	POSTP
Postp+PCDat	POSTP
Postp+PCIns	POSTP
Postp+PCGen	POSTP
Adverb	ANY
PRON-DB	ANY
PRON-DB-ADJ	PRON-DB
Ques	ANY
ADJ-DB	ANY
ADJ-DB-NOUN	ADJ-DB

NOUN-DB	ANY
NOUN-DB-NOUN	NOUN-DB
NOUN-DB-PRON	NOUN-DB
NOUN-DB-ADJ	NOUN-DB
NOUN-DB-ADVERB	NOUN-DB
NOUN-DB-VERB	NOUN-DB
PRON-DB-NOUN	PRON-DB
PRON-DB-PRON	PRON-DB
PRON-DB-VERB	PRON-DB
ADVERB-DB	ANY
ADVERB-DB-ADVERB	ADVERB-DB
VERB-DB	ANY
VERB-DB-ADVERB	VERB-DB
VERB-DB-VERB	VERB-DB
VERB-DB-NOUN	VERB-DB
VERB-DB-ADJ	VERB-DB
ADJ-DB-ADJ	ADJ-DB
ADJ-DB-ADVERB	ADJ-DB
ADJ-DB-VERB	ADJ-DB
D̂B+Noun+Zero	ADJ-DB-NOUN
D̂B+Adverb+Ly	ADJ-DB-ADVERB
D̂B+Verb+Zero	ADJ-DB-VERB NOUN-DB-VERB PRON-DB-VERB
D̂B+Adj+Rel	PRON-DB-ADJ NOUN-DB-ADJ PRON-DB-ADJ
D̂B+Adj+FitFor	ADJ-DB-ADJ NOUN-DB-ADJ
D̂B+Noun+Ness	NOUN-DB-NOUN ADJ-DB-NOUN
D̂B+Noun+Agt	ADJ-DB-NOUN

	NOUN-DB-NOUN
D̂B+Adj+Agt	ADJ-DB-ADJ NOUN-DB-ADJ VERB-DB-ADJ
D̂B+Noun+Dim	ADJ-DB-NOUN
D̂B+Verb+Become	ADJ-DB-VERB NOUN-DB-VERB
D̂B+Verb+Acquire	ADJ-DB-VERB NOUN-DB-VERB
D̂B+Adj+Without	ADJ-DB-ADJ NOUN-DB-ADJ PRON-DB-ADJ
D̂B+Adj+With	ADJ-DB-ADJ NOUN-DB-ADJ PRON-DB-ADJ
D̂B+Adverb+AsIf	NOUN-DB-ADVERB VERB-DB-ADVERB
D̂B+Adverb+Ly	NOUN-DB-ADVERB
D̂B+Noun+Dim	NOUN-DB-NOUN
D̂B+Pron+Rel	PRON-DB-PRON NOUN-DB-PRON
D̂B+Adverb+While	ADVERB-DB-ADVERB VERB-DB-ADVERB
D̂B+Verb+Caus	VERB-DB-VERB
D̂B+Verb+Pass	VERB-DB-VERB
D̂B+Verb+Recip	VERB-DB-VERB
D̂B+Verb+AbleNeg	VERB-DB-VERB
D̂B+Verb+Repeat	VERB-DB-VERB
D̂B+Verb+Hastily	VERB-DB-VERB
D̂B+Verb+EverSince	VERB-DB-VERB
D̂B+Verb+Able	VERB-DB-VERB
D̂B+Verb+Almost	VERB-DB-VERB

D̂B+Verb+Stay	VERB-DB-VERB
D̂B+Verb+Start	VERB-DB-VERB
D̂B+Noun+NotState	VERB-DB-NOUN
D̂B+Adverb+”maksizin”	VERB-DB-ADVERB
D̂B+Adverb+WithoutHavingDone	VERB-DB-ADVERB
D̂B+Noun+Inf1	VERB-DB-NOUN
D̂B+Noun+Inf2	VERB-DB-NOUN
D̂B+Noun+Inf3	VERB-DB-NOUN
D̂B+Noun+FeelLike	VERB-DB-NOUN
D̂B+Adj+Zero	VERB-DB-ADJ
D̂B+Adj+PresPart	VERB-DB-ADJ
D̂B+Adj+FutPart	VERB-DB-ADJ
D̂B+Adj+FeelLike	VERB-DB-ADJ
D̂B+Adj+PastPart	VERB-DB-ADJ
D̂B+Adverb+AsLongAs	VERB-DB-ADVERB
D̂B+Adverb+When	VERB-DB-ADVERB
D̂B+Adverb+ByDoingSo	VERB-DB-ADVERB
D̂B+Adverb+AfterDoingSo	VERB-DB-ADVERB
D̂B+Adverb+SinceDoingSo	VERB-DB-ADVERB

Table A.1: Lexical Category List for Turkish

Appendix B

Lattice Structure for English

The lattice structure for English is given in Table B.1, the main categories that have been used by Xerox morphological analyzer is written in normal case whereas the subcategories that are added to arrange the lattice is written in capitals.

Category Name	Parent Category in Lattice
ANY	
Verb	ANY
Quant	ANY
Prep	ANY
NOUN-SUF	ANY
NOUN-SUF-COUNT	NOUN-SUF
ADJ-SUF	ANY
VERB-SUF	ANY
VERB-SUF-AGGR	VERB-SUF
VERB-SUF-TENSE	VERB-SUF
VERB-SUF-COUNT	VERB-SUF
VERB-SUF-COUNT-AGR	VERB-SUF
Pl	NOUN-SUF-COUNT VERB-SUF-COUNT
Sg	NOUN-SUF-COUNT

	VERB-SUF-COUNT
SP	NOUN-SUF-COUNT VERB-SUF-COUNT
VProg	NOUN-SUF ADJ-SUF
Pron	ANY
Part	ANY
Meas	ANY
Let	ANY
Interj	ANY
Non3sg	VERB-SUF-AGGR
1P	VERB-SUF-AGGR
2P	VERB-SUF-AGGR
3P	VERB-SUF-AGGR
Inf	VERB-SUF-TENSE
PastBoth	VERB-SUF-TENSE
PastPerf	VERB-SUF-TENSE
PastTense	VERB-SUF-TENSE
Pres	VERB-SUF-TENSE
Prog	VERB-SUF-TENSE
1sg	VERB-SUF-COUNT-AGR
123SP	VERB-SUF-COUNT-AGR
2sg	VERB-SUF-COUNT-AGR
3sg	VERB-SUF-COUNT-AGR
PRON-SUF	ANY
PRON-SUF-CASE	PRON-SUF
Gen	PRON-SUF-CASE
Nom	PRON-SUF-CASE
Obl	PRON-SUF-CASE
NomObl	PRON-SUF-CASE
Det	ANY
DET-SUF	ANY

Def	DET-SUF
Indef	DET-SUF
Conj	ANY
Comp	ADJ-SUF
Sup	ADJ-SUF
VPap	ADJ-SUF
NUMBER	ANY
Num	NUMBER
Dec	NUMBER
Dig	NUMBER
Noun	ANY
Prop	Noun
City	Noun
Bus	Noun
Continent	Noun
Country	Noun
Deg	Noun
Init	Noun
Title	Noun
Adj	ANY
Adv	ANY
Aux	ANY

Table B.1: Lexical Category List for English

Appendix C

Sample Training File Subset

A subset of the training set used in this system is given:

train_pair([a+Det +Indef +Sg brown+Adj car+Noun +Sg] [bir+Num+Card kahverengi+Adj araba+Noun +A3sg +Pnon +Nom]).

train_pair([a+Det +Indef +Sg cat+Noun +Sg come+Verb +Pres +3sg] [bir+Num +Card kedi+Noun +A3sg +Pnon +Nom gel+Verb +Pos +Aor +A3sg]).

train_pair([a+Det +Indef +Sg cat+Noun +Sg go+Verb +Pres +3sg] [bir+Num +Card kedi+Noun +A3sg +Pnon +Nom git+Verb +Pos +Aor +A3sg]).

train_pair([a+Det +Indef +Sg green+Adj apple+Noun +Sg] [bir+Num+Card yeşil+Adj elma+Noun +A3sg +Pnon +Nom]).

train_pair([a+Det +Indef +Sg pig+Noun +Sg go+Verb +Pres +3sg] [bir+Num +Card domuz+Noun +A3sg +Pnon +Nom git+Verb +Pos +Aor +A3sg]).

train_pair([a+Det +Indef +Sg red+Adj apple+Noun +Sg] [bir+Num+Card kırmızı+Adj elma+Noun +A3sg +Pnon +Nom]).

train_pair([a+Det +Indef +Sg white+Adj car+Noun +Sg] [bir+Num+Card beyaz+Adj araba+Noun +A3sg +Pnon +Nom]).

train_pair([a+Det +Indef +Sg yellow+Adj apple+Noun +Sg] [bir+Num+Card sarı+Adj elma+Noun +A3sg +Pnon +Nom]).

train_pair([ali+Prop+Masc +Sg go+Verb +Pres +3sg] [ali+Noun +Prop +A3sg +Pnon +Nom git+Verb +Pos +Aor +A3sg]).

train_pair([all+Det +Pl book+Noun +Pl] [bütün+Adj kitap+Noun +A3pl +Pnon +Nom]).

train_pair([all+Det +Pl house+Noun +Pl] [bütün+Adj ev+Noun +A3pl +Pnon +Nom]).

train_pair([all+Det +Pl notebook+Noun +Pl] [bütn+Adj defter+Noun +A3pl +Pnon +Nom]).

train_pair([apple+Noun +Pl be+Verb +Pres +Pl fruit+Noun +Sg] [elma+Noun +A3pl +Pnon +Nom meyve+Noun +A3sg +Pnon +Nom D̂B+Verb+Zero +Pres +Cop +A3sg]).

train_pair([apple+Noun +Pl be+Verb +Pres +Pl not+Adv fruit+Noun +Sg] [elma+Noun +A3pl +Pnon +Nom meyve+Noun +A3sg +Pnon +Nom değil+Noun +A3sg +Pnon +Nom D̂B+Verb+Zero +Pres +Cop +A3sg]).

train_pair([apple+Noun +Sg be+Verb +Pres +3sg a+Det +Indef +Sg fruit+Noun +Sg] [elma+Noun +A3sg +Pnon +Nom bir+Num+Card meyve+Noun +A3sg +Pnon +Nom D̂B+Verb+Zero +Pres +Cop +A3sg]).

train_pair([approximately+Adv two+Num+Card liter+Noun +Pl] [yaklaşık+Noun +A3sg +Pnon +Nom iki+Num+Card litre+Noun +A3sg +Pnon +Nom]).

train_pair([boy+Noun +Pl be+Verb +Pres +Pl not+Adv go+Verb +Prog] [oğlan+Noun +A3pl +Pnon +Nom git+Verb +Neg +Prog1 +A3pl]).

train_pair([that+Det +Sg man+Noun +Sg be+Verb +Pres +3sg a+Det

+Indef +Sg tailor+Noun +Sg] [şu+Adj adam+Noun +A3sg +Pnon +Nom bir+Num+Card terzi+Noun +A3sg +Pnon +Nom D̂B+Verb+Zero +Pres +Cop +A3sg]).

train_pair([the+Det +Def +SP funny+Noun +Sg boy+Noun +Pl do+Aux +PastTense +123SP not+Adv go+Verb +Pres +Non3sg to+Prep the+Det +Def +SP mountain+Noun +Sg yesterday+Adv] [komik+Adj oğlan+Noun +A3pl +Pnon +Nom dün+Adverb dağ+Noun +A3sg Pnon +Dat git+Verb +Neg +Past +A3pl]).

train_pair([I+Pron+Pers +Nom +1P +Sg do+Aux +PastTense +123SP not+Adv write+Verb +Pres +Non3sg a+Det +Indef +Sg letter+Noun +Sg] [bir+Num+Card mektup+Noun +A3sg +Pnon +Nom yaz+Verb +Neg +Past +A1sg]).

train_pair([Mary+Prop+Fem +Sg do+Aux +PastTense +123SP not+Adv go+Verb +Pres +Non3sg] [mary+Noun +Prop +A3sg +Pnon +Nom git+Verb +Neg +Past +A3sg]).

train_pair([it+Pron+Pers +NomObl +3P +Sg be+Verb +Pres +3sg rain+Verb +Prog] [yağmur+Noun +A3sg +Pnon +Nom yağ+Verb +Pos +Prog1 +A3sg]).

train_pair([if+Conj+Sub a+Det +Indef +Sg pen+Noun +Sg be+Verb +Pres +3sg drop+Verb +PastBoth +123SP then+Adv it+Pron+Pers +NomObl +3P +Sg fall+Verb +Pres +3sg] [bir+Num+Card kalem+Noun +A3sg +Pnon +Nom bırak+Verb D̂B+Verb+Pass +Pos +Aor +Cond +A3sg düş+Verb +Pos +Aor +A3sg]).

train_pair([girl+Noun +Pl will+Aux write+Verb +Pres +Non3sg a+Det +Indef +Sg message+Noun +Sg tomorrow+Adv] [kız+Noun +A3pl +Pnon +Nom yarın+Adverb bir+Num+Card mesaj+Noun +A3sg +Pnon +Nom yaz+Verb +Pos +Fut +A3pl]).

Appendix D

Sample Test File Subset

A subset of the test set used in this system is given:

train_pair([a+Det +Indef +Sg black+Adj car+Noun +Sg],[bir+Num+Card siyah+Adj araba+Noun +A3sg +Pnon +Nom]).

train_pair([interesting+Adj red+Adj book+Noun +Sg],[ilgin+Adj kırmızı+Adj kitap+Noun +A3sg +Pnon +Nom]).

train_pair([it+Pron+Pers +NomObl +3P +Sg be+Verb +Pres +3sg a+Det +Indef +Sg red+Adj car+Noun +Sg],[o+Pron +A3sg +Pnon +Nom bir+Num+Card kırmızı+Adj araba+Noun +A3sg +Pnon +Nom D̂B+Verb+Zero +Pres +Cop +A3sg]).

train_pair([I+Pron+Pers +Nom +1P +Sg be+Verb +Pres +1sg a+Det +Indef +Sg teacher+Noun +Sg],[bir+Num+Card öğretmen+Noun +A3sg +Pnon +Nom D̂B+Verb+Zero +Pres +A1sg +Cop]).

train_pair([I+Pron+Pers +Nom +1P +Sg do+Aux +PastTense +123SP not+Adv come+Verb +Pres +Non3sg],[gel+Verb +Neg +Past +A1sg]).

train_pair([I+Pron+Pers +Nom +1P +Sg will+Aux write+Verb +Pres +Non3sg a+Det +Indef +Sg message+Noun +Sg],[bir+Num+Card mesaj+Noun

+A3sg +Pnon +Nom yaz+Verb +Pos +Fut +A1sg]).

train_pair([it+Pron+Pers +NomObl +3P +Sg be+Verb +Pres +3sg not+Adv an+Det +Indef +Sg artificial+Adj tree+Noun +Sg],[o+Pron +A3sg +Pnon +Nom bir+Num+Card yapay+Adj ağa+Noun +A3sg +Pnon +Nom değil+Noun +A3sg +Pnon +Nom D̂B+Verb+Zero +Pres +Cop +A3sg]).

train_pair([four+Num+Card brown+Adj car+Noun +Pl],[dört+Num+Card kahverengi+Adj araba+Noun +A3sg +Pnon +Nom]).

train_pair([boy+Noun +Pl be+Verb +Pres +Pl go+Verb +Prog],[oğlan+Noun +A3pl +Pnon +Nom git+Verb +Pos +Prog1 +A3pl]).

train_pair([ayşe+Noun +Sg +Part +Gen book+Noun +Sg],[ayşe+Noun +Prop +A3sg +P2sg +Gen kitap+Noun +A3sg +P3sg +Nom]).

train_pair([she+Pron+Pers +Nom +3P +Sg be+Verb +Pres +3sg a+Det +Indef +Sg tailor+Noun +Sg],[bir+Num+Card terzi+Noun +A3sg +Pnon +Nom D̂B+Verb+Zero +Pres +Cop +A3sg]).

train_pair([I+Pron+Pers +Nom +1P +Sg will+Aux come+Verb +Pres +Non3sg tomorrow+Adv],[yarın+Adverb gel+Verb +Pos +Fut +A1sg]).

train_pair([ayşe+Noun +Sg be+Verb +Pres +3sg not+Adv a+Det +Indef +Sg teacher+Noun +Sg],[ayşe+Noun +Prop +A3sg +Pnon +Nom bir+Num+Card retmen+Noun +A3sg +Pnon +Nom deil+Noun +A3sg +Pnon +Nom D̂B+Verb+Zero +Pres +Cop +A3sg]).