A Specific Least General Generalization of Strings and Its Application to Example-Based Machine Translation

Ilyas Cicekli

ilyas@cs.bilkent.edu.tr

Department of Computer Engineering, Bilkent University, 06533 Bilkent, Ankara, Turkey

Abstract

Since the least general generalization (LGG) of strings may cause an over-generalization in the generalization process of clauses, we propose a specific least general generalization (SLGG) of strings to reduce over-generalization. To create a SLGG of two strings, first a minimal match sequence between these strings is found. A minimal match sequence of two strings consists of similarities and differences to represent similar parts and differing parts between those strings. The differences in the minimal match sequence are replaced to create a SLGG of those strings. We also introduce a learning heuristic based on SLGGs of strings to be used in example-based machine translation.

1 Introduction

Example-Based Machine Translation (EBMT), originally proposed by Nagao [15], is one of the main approaches of corpus-based machine translation in which the required knowledge resources are automatically acquired from corpora. The main idea behind EBMT is that a given input sentence in the source language is compared with the example translations in the given bilingual parallel text to find the closest matching examples so that these examples can be used in the translation of that input sentence. After finding the closest matchings for the sentence in the source language, parts of the corresponding target language sentence are constructed using structural equivalences and deviances in those matches. Following Nagoa's original proposal, several machine translation methods that utilize bilingual corpora have been studied [5, 9, 20, 21, 22, 23]. Some researchers [3, 24] only utilize bilingual corpora to create a bilingual dictionary and use it during the translation process. In other words, they aligned bilingual corpora at word level to figure out corresponding words in languages. Bilingual corpora is also aligned at phrase level by some other researchers [1, 2, 16]. But these correspondences between two languages are only accomplished at atomic level, and they are used in the translation of portions of sentences. Kaji [8] tried to learn correspondences of English and Japanese syntactic structures from bilingual corpora. This is similar to the work in [6] and it needs reliable parsers for both source and target languages. The technique described here not only learns atomic correspondences between two languages, it also learns general templates describing the structural correspondence (not syntactic structure) from bilingual corpora.

Researchers in Machine Learning (ML) community have widely used exemplar-based representation. Medin and Schaffer [10] were the first researchers who proposed exemplar-based learning as a model of human learning. The characteristic examples stored in the memory are called *exemplars*. In EBMT, translation examples should be available prior to the translation of an input sentence. In most of the EBMT systems, these translation examples are directly used without any generalization. Kitano [9] manually encoded translation rules, however this is a difficult and error-prone task for a large corpus. In this paper, we formulate the acquisition of translation rules, which are similar to exemplars, as a machine learning problem in order to automate this task.

Inductive Logic Programming (ILP) deals with the induction of predicate definitions from examples and background knowledge. Recently, researchers have been trying to apply ILP techniques to the construction of natural language processing (NLP) systems. Mooney and Califf [11] have applied ILP techniques to learning the past tense of English verbs, and showed that ILP techniques are more effective than neural-network and decision-tree methods. Mooney [12] has also worked a system which learns a parser from a training corpus of parsed sentences. In [14], Muggleton presents how to use ILP techniques in NLP systems. We also believe that ILP techniques will find good application domains in the construction of NLP systems. In this paper, we present how to use ILP techniques in the creation of an example-based machine translation system.

In EBMT area, learning must be done just from positive translation examples because negative examples will not be available most of the time. Just learning from positive examples may cause over-generalization of examples because there are no restrictions imposed by negative examples. For example, just using Plotkin's [18, 19] the relative least general generalization (RLGG) schema in learning of translation templates from given translation examples between two natural languages will be disastrous. From two unrelated translation examples, an over-generalized clause will be created, and this clause will satisfy almost any given goal. In this paper, we suggest a specific least general generalization (SLGG) for strings which will reduce this over-generalization.

Let us assume that, we have an ILP system (for example, the GOLEM system [13]) which uses only RLGG schema to generalize two given clauses, and we want to generalize the following clauses.

```
p([a,b],[x,y]).
p([c,d,b],[z,w,y]).
```

Although these two clauses have a common property, this will not be captured by this ILP system. This common property is that the first arguments end with atom `b`, and the second arguments end with atom `y`. This ILP system will generalize these clauses with the following clause

```
p([A,B|C],[D,E|F]).
```

without capturing that common property. We believe that this clause is an over-generalization, and it will accept any lists whose lengths are more than 1 for both arguments. In fact, if they were two translation examples between two natural languages, this generalized clause will say that anything can be a translation of anything. On the other hand, our proposed mechanism will generalize these clauses with the following clause.

```
p(L1,L2) :- append(X,[b],L1),append(Y,[y],L2),p(X,Y).
```

This generalized clause means that any list ending with atom `b` can be a translation of a list ending with atom `y` if the prefixes of these lists are translations of each other. In addition to this general clause, our mechanism also infers the following two unit clauses from these examples.

```
p([a],[x]).
p([c,d],[z,w]).
```

Here we assume that `append` predicate is given as background knowledge. In the generalization process, we use SLGGs of strings to create that general clause and those two unit clauses.

The translation template learning framework presented in this paper is based on a learning heuristic which uses the SLGG schema for strings. This learning schema infers the correspondences between the patterns in the source and target languages from two given translation pairs. According to this heuristic, given two translation examples, if the sentences in the source language exhibit some similarities, then the corresponding sentences in the target language must have similar parts, and they must be translations of the similar parts of the sentences in the source language. Further, the remaining differing constituents of the source sentences should also match the corresponding differences of the target sentences. However, if the sentences do not exhibit any similarities, then no correspondences are inferred. Given a corpus of translation examples, our learning heuristic infers the correspondences between the source and target languages in the form of templates. These templates can be used for translation in both directions.

The rest of the paper is organized as follows. First, we explain how a minimal match sequence, which represents similarities and differences in a pair of strings, can be found. In Section 3, we propose a specific least general generalization of two strings which is created from the minimal match sequence of those strings. We describe our learning heuristic based on SLGGs of strings in Section 4. In Section 5, the usage of our learning heuristic in an example-based machine translation system is described. Finally, we conclude the paper with pointers for further research.

2 Minimal Match Sequence

In this section, we formally describe a minimal match sequence between two non-empty strings of atoms. The minimal match sequence of two strings will be used in the creation of the SLGG of those strings. Before we define the minimal match sequence, we give other required definitions used in its definition.

A *similarity* between α_1 and α_2, where α_1 and α_2 are two non-empty strings of atoms, is a non-empty string β such that $\alpha_1 = \alpha_{1,1}\beta\alpha_{1,2}$ and $\alpha_2 = \alpha_{2,1}\beta\alpha_{2,2}$. A similarity represents a similar part between two strings.

A *difference* between α_1 and α_2, where α_1 and α_2 are two non-empty strings of atoms, is a pair of two strings (β_1, β_2) where β_1 is a substring of α_1 and β_2 is a substring of α_2, the same atom cannot occur in both β_1 and β_2, and at least one of them is not empty (When we use a minimal match sequence in example-based machine translation domain, we will insist that both constituents of its differences are not empty.). A difference represents a pair of differing parts between two strings.

A *minimal match sequence* between two strings α_1 and α_2 is a sequence of similarities and differences between α_1 and α_2 such that the following conditions must be satisfied by this match sequence:

1. Concatenation of similarities and the first constituents of differences must be equal to α_1.
2. Concatenation of similarities and the second constituents of differences must be equal to α_2.
3. A minimal match sequence should contain at least one similarity and one difference.
4. A similarity cannot follow another similarity, and a difference cannot follow another difference in a minimal match sequence.
5. If an atom occurs in a similarity, it cannot occur in any difference.
6. If an atom occurs in the first constituent of a difference, it cannot occur in the second constituent of a prior difference.
7. If an atom occurs in the second constituent of a difference, it cannot occur in the first constituent of a prior difference.

The order of operands in the concatenation operations are same as their order in the match sequence. So, a minimal match sequence M between α_1 and α_2 is in the following form:

$M = P_1...P_n$ where each P_i is a similarity S_i or a difference $D_i = (D_{i,1}, D_{i,2})$, and $n \geq 2$.

If we define two constituent functions as follows:

$$C_{i,1} = \begin{cases} S_i & \text{if } P_i \text{ is a similarity } S_i \\ D_{i,1} & \text{if } P_i \text{ is a difference } (D_{i,1}, D_{i,2}) \end{cases} \qquad C_{i,2} = \begin{cases} S_i & \text{if } P_i \text{ is a similarity } S_i \\ D_{i,2} & \text{if } P_i \text{ is a difference } (D_{i,1}, D_{i,2}) \end{cases}$$

then

$$\alpha_1 = C_{1,1}...C_{n,1} \qquad \alpha_2 = C_{1,2}...C_{n,2}$$

The conditions for the minimal match sequence guarantee that there is a unique match sequence, or there is no match sequence between two strings. The conditions 1 and 2 reflect the main characteristic of a match sequence. The condition 3 is introduced to guarantee that the strings are not exactly the same, or they are not completely different (i.e. they do not have any common part). The conditions from 4 to 7 are introduced to guarantee the uniqueness of the minimal match sequence.

Although an atom can appear in more than one similarity according to the conditions above, we can observe the following facts for minimal match sequences.

1. If an atom appears in both strings α_1 and α_2, it must be appear n times, where $n \geq 1$, in both of those strings. Otherwise, they cannot have a minimal match sequence.
2. If an atom appears more than once in both strings α_1 and α_2, its i^{th} occurrence in α_1 and its i^{th} occurrence in α_2 must end up in the same similarity of their minimal match sequence.

Some examples for minimal match sequences:

- The minimal match sequence of $abcbd$ and $ebfbg$ will be $(a,e)b(c,f)b(d,g)$.
- The strings $abcbd$ and ebf cannot have a minimal match sequence because b occurs twice in the first string and it occurs only once in the second string.
- The minimal match sequence of abc and $dbef$ is $(a,d)b(c,ef)$.

- The minimal match sequence of bc and $dbef$ is $(a, \epsilon)b(c, ef)$.
- The minimal match sequence of $abcde$ and $fbcg$ is $(a, f)bc(de, g)$.
- The minimal match sequence of $abdbce$ and $fbgbch$ is $(a, f)b(d, g)bc(e, h)$.
- The minimal match sequence of $abdbce$ and $fbbch$ is $(a, f)b(d, \epsilon)bc(e, h)$.

The definition of the minimal match sequence can be extended for strings of atoms and variables by assuming a variable as a new atom. A variable in a string represents its substring, and that variable can be replaceable with any string. If two strings are going to have a minimal match sequence, they cannot contain the same variables. This can be easily satisfied by renaming variables in one of those strings. Then each variable is treated as a new atom in the creation of the minimal match sequence. For example, the match sequence of $aXbcYd$ and $efbcZ$ will be $(aX, ef)bc(Yd, Z)$. The reader can observe that a variable cannot appear in the similarity of a minimal match sequence.

3 A Specific Least General Generalization of Strings

Plotkin's relative least general generalization (RLGG) technique [18, 19] is used by many ILP systems[13]. In that technique, least general generalization (LGG) of terms are used in the generalization process of clauses. For example, the GOLEM system, which uses RLGG schema, generalizes following two clauses

```
p([b,a]).
p([c,d,a]).
```

by creating the following general clause

```
p([A,B|C]).
```

In this generalization, the LGG of terms `[b,a]` and `[c,d,a]` is the term `[A,B|C]`. This generated clause covers given two example clauses, but we think that it is an over-generalization of those clauses. Although both lists in these examples end with atom `a`, this common characteristic has not been captured by the LGG of those lists. The generalized clause does not reflect this fact because of over-generalization.

Here, we propose a new specific least general generalization (SLGG) of strings. Our proposed technique will generalize the examples above as

```
p(L) :- append(L1,[a],L).
```

by assuming that we have the `append` predicate for lists as background knowledge.

In our technique, to generalize two clause examples of a single-arity predicate with string arguments, we use a SLGG of two strings. The SLGG of two strings exists only if they have a minimal match sequence. If they do not have a minimal match sequence, they do not have the SLGG, and we do not generalize those clauses. In other words, two clauses are only generalized if their arguments have a minimal match sequence.

The SLGG of two strings α_1 and α_2 is found as follows.

- First the minimal match sequence of α_1 and α_2 is found.
- Then all differences in this match sequence are replaced with new variables to create a SLGG. The minimal match sequence in which differences are replaced with variables is the SLGG of those strings. The same differences are replaced with the same variables.

For example, the SLGG of the strings $abcd$ and $ecfg$ is found as follows.

- Their minimal match sequence is $(ab, e)c(d, fg)$.
- Two differences are replaced with two new variables to create their SLGG, and it will be XcY.

Another example can be the SLGG of the strings $abcdeaf$ and $gbchegf$. Their minimal match sequence is $(a, g)bc(d, h)e(a, g)f$, and their SLGG will be $XbcYeXf$. In this example, the same difference is replaced with the same variable.

Example 1: In this example, we will show how our technique is used in the generalization of single-arity predicates whose arguments are strings. Let us assume that the following clauses are given as positive examples.

```
1. p(ba).
2. p(cda).
3. p(a).
```

These clauses will be represented in Prolog as follows.

```
1. p([b,a]).
2. p([c,d,a]).
3. p([a]).
```

To generalize clauses 1 and 2, we will find the SLGG of the strings `ba` and `cda`. Since their minimal match sequence is `(b,cd)a`, their SLGG will be `Xa`. Thus, the generalization of these two clauses will be `p(Xa)`. Then, we will try to generalize this new clause with the third clause. Since the minimal match sequence of strings `Xa` and `a` is `(X,`ϵ`)a`, their SLGG will be `Ya`. And, their generalization will be `p(Ya)`. In fact, this clause is the generalization of these three positive examples. This clause will be represented in Prolog as follows

```
p(L) :- append(L1,[a],L).
```

by assuming that we have the `append` predicate for lists as background knowledge.

On the other hand, these three clauses could have been generalized by the GOLEM system as the following clause

```
p([A|B]).
```

without capturing the common property of these examples.

Example 2: Now, let us assume that the following clauses are given as positive examples.

```
1. p(ca).
2. p(dea).
3. p(b).
4. p(fgb).
```

The generalization of clauses 1 and 2 will be `p(Xa)` since the SLGG of `ca` and `dea` is `Xa`. The generalization of clauses 3 and 4 will be `p(Yb)` since the SLGG of `b` and `fgb` is `Yb`. Now, the given four clauses are generalized as two general clauses. We do not generalize these two clauses because the strings `Xa` and `Yb` do not have a minimal match sequence. As result, we induce these two clauses as the generalization of the given examples, and they are represented in Prolog as follows.

```
p(L) :- append(L1,[a],L).
p(L) :- append(L1,[b],L).
```

These two clauses capture the fact that the argument of this learned predicate should end with atom `a` or `b`.

The reader should notice that it does not matter which clause pair is tried first for the generalization. After all possible combinations are tried, we will reach to the same result. This example demonstrates that we only generalize if two strings have a minimal match sequence; otherwise we leave clauses as it is without a generalization.

The GOLEM system will again generalize these four clauses with the same predicate `p([A|B])`.

4 A Learning Heuristic Based on SLGGs of Strings

In this section, we describe a learning heuristic based on SLGGs of strings. Here we assume that we have positive examples for a 2-arity predicate whose both arguments are strings. We also assume that the alphabet of strings in the first argument position is different from the alphabet of strings in the second argument position. In fact, these positive examples are translation examples between two natural languages. The learning heuristic presented in this section is used in the generalization of translation examples to create general translation templates between two natural languages.

A translation template is an *atomic* or *general* translation template. An *atomic translation template* between languages $\mathcal{L}^a$ and $\mathcal{L}^b$ is a pair of two non-empty strings $\alpha \leftrightarrow \beta$ where $\alpha \in \mathcal{L}^a$ and $\beta \in \mathcal{L}^b$. A given *translation example* will be an atomic translation template.

A *general translation template* between languages $\mathcal{L}^a$ and $\mathcal{L}^b$ is an if-then rule in the following form:

$T^a \leftrightarrow T^b$ **if** $X_1 \leftrightarrow Y_1$ **and** ... **and** $X_n \leftrightarrow Y_n$

where $n \geq 1$, T^a is a string of atoms in the alphabet of the language $\mathcal{L}^a$ and variables $X_1, ..., X_n$, T^b is a string of atoms in the alphabet of the language $\mathcal{L}^b$ and variables $Y_1, ..., Y_n$, and both T^a and T^b must contain at least one atom.

For example, if the alphabet of $\mathcal{L}^a$ is $\mathcal{A} = \{a, b, c, d, e, f, g, h\}$ and the alphabet of $\mathcal{L}^b$ is $\mathcal{B} = \{t, u, v, w, x, y, z\}$, the followings are some examples of general templates between $\mathcal{L}^a$ and $\mathcal{L}^b$.

- $abX_1c \leftrightarrow uY_1$ **if** $X_1 \leftrightarrow Y_1$
- $aX_1bX_2c \leftrightarrow Y_2uvY_1$ **if** $X_1 \leftrightarrow Y_1$ **and** $X_2 \leftrightarrow Y_2$
- $aX_1X_2b \leftrightarrow Y_2vY_1$ **if** $X_1 \leftrightarrow Y_1$ **and** $X_2 \leftrightarrow Y_2$

A general template is a generalization of translation examples, where certain components are generalized by replacing them with variables and establishing bindings between these variables. For example, in the first example above, abX_1c represents all sentences of $\mathcal{L}^a$ starting with ab and ending with c where X_1 represents a non-empty string on $\mathcal{A}$, and uY_1 represents all sentences of $\mathcal{L}^b$ starting with u where Y_1 represents a non-empty string on $\mathcal{B}$. That general template says that a sentence of $\mathcal{L}^a$ in the form of abX_1c corresponds to a sentence of $\mathcal{L}^b$ in the form of uY_1 given that X_1 corresponds to Y_1. If we know that the correspondence $de \leftrightarrow vyz$, the correspondence $abdec \leftrightarrow uvyz$ can be inferred from that general template.

A *minimal match sequence between two translation examples* $\alpha_1 \leftrightarrow \beta_1$ and $\alpha_2 \leftrightarrow \beta_2$ is a pair of two minimal match sequences $M^a \leftrightarrow M^b$ where M^a is the minimal match sequence of α_1 and α_2, and M^b is the minimal match sequence of β_1 and β_2. If a minimal match sequence exists for a pair of translation examples, it will be unique.

Our learning heuristic learns translation templates from two given translation examples. In order to learn translation templates from two given examples, first a minimal match sequence of these examples is found; then the learning heuristic is applied to this minimal match sequence. The learning heuristic learns a general template by replacing differences with variables in a match sequence, and establishing bindings between these variables. In addition, a learning heuristic can also learn atomic templates. Of course, if there is no minimal match sequence for the examples, the learning heuristic cannot be applied to them. The learning heuristic will also insist extra conditions on minimal match sequences. The learning heuristic can learn new templates from minimal match sequence $M^a \leftrightarrow M^b$ of translation examples $E_1 = \alpha_1 \leftrightarrow \beta_2$ and $E_2 = \alpha_2 \leftrightarrow \beta_2$, if this match sequence satisfies the following conditions:

1. Both M^a and M^b must contain at least one similarity and one difference. This condition will be automatically satisfied, because we insist that a minimal match sequence must satisfy this condition.

2. Both M^a and M^b cannot contain a difference with empty constituent.

3. Both M^a and M^b must contain n differences where $n \geq 1$. In other words, they must contain equal number of differences.

4. Each difference in M^a must correspond to a difference in M^b, and a difference cannot correspond to more than one difference on the other side. Thus, we will have n corresponding differences.

If the minimal match sequence satisfies the first three conditions, the n corresponding differences must be found to satisfy the fourth condition. If there is only one difference on both sides, they should correspond to each other (i.e. the fourth condition is trivially satisfied). But, if there is more than one difference on both sides, we need to look at previously learned translation templates to determine the corresponding differences. For example, if there are two differences D_1^a and D_2^a in M^a, and two differences D_1^b and D_2^b in M^b; we cannot determine whether D_1^a corresponds to D_1^b or D_2^b without using prior knowledge. Now, let us assume that the corresponding between the differences D_1^a and D_1^b has been learned earlier; in this case D_2^a must correspond to D_2^b. In general, if the $n-1$ corresponding differences have been learned earlier, the last two differences must correspond to each other. We say that the corresponding difference between the differences $D^a = (D_1^a, D_2^a)$ and $D^b = (D_1^b, D_2^b)$ has been learned, if the following two atomic translation templates have been learned earlier.

$D_1^a \leftrightarrow D_1^b$
$D_2^a \leftrightarrow D_2^b$

Now, let us assume that the differences in M^a are D_1^a,....,D_n^a and the differences in M^b are D_1^b,....,D_n^b where D_i^a corresponds to D_i^b. In this case, the first $n-1$ corresponding differences have been learned earlier, and the corresponding difference between the differences D_n^a and D_n^b is inferred now. The learning heuristic replaces each D_i^a with the variable X_i to create $SLGG^a$ from M^a, and each D_i^b with the variable Y_i to create $SLGG^b$ from M^b. In fact, $SLGG^a$ is the SLGG of the strings α_1 and α_2, and $SLGG^b$ is the SLGG of strings β_1 and β_2. Then, the following general template is induced by the learning heuristic.

$SLGG^a \leftrightarrow SLGG^b$ **if** $X_1 \leftrightarrow Y_1$ **and** ... **and** $X_n \leftrightarrow Y_n$

In addition, the following two atomic templates are learned from the inferred corresponding difference between $D_n^a = (D_{n,1}^a, D_{n,2}^a)$ and $D_n^b = (D_{n,1}^b, D_{n,2}^b)$.

$D_{n,1}^a \leftrightarrow D_{n,1}^b$
$D_{n,2}^a \leftrightarrow D_{n,2}^b$

Example 3.

Let us assume that

$abc \leftrightarrow vwxyz$
$abef \leftrightarrow tuxyz$

are two translation examples. The minimal match sequence for these examples will be

$ab\,(c, ef) \leftrightarrow (vw, tu)\,xyz.$

Since the minimal match sequence immediately satisfies the fourth condition, the following translation templates can be learned from that match sequence.

$ab\,X_1 \leftrightarrow Y_1\,xyz$ **if** $X_1 \leftrightarrow Y_1$
$c \leftrightarrow vw$
$ef \leftrightarrow tu$

where abX_1 is the SLGG of abc and $abef$, and Y_1xyz is the SLGG of $vwxyz$ and $tuxyz$.

The learned translation templates will be represented in Prolog as follows.

```
tt(S1,S2) :- append([a,b],X1,S1), append(Y1,[x,y,z],S2), tt(X1,Y1).
tt([c],[v,w]).
tt([e,f],[t,u]).
```

The order of literals in the body of general translation templates can be changed to increase the efficiency of this translation template in the translation phase.

Example 4.

Let us assume that

$bac \leftrightarrow vwxy$
$daef \leftrightarrow tuxz$

are two translation examples. The minimal match sequence for these examples will be

$(b, d)\,a\,(c, ef) \leftrightarrow (vw, tu)\,x\,(y, z).$

This minimal match sequence satisfies the first three conditions because it has two differences on both sides. But we do not know whether it satisfies the fourth condition. We cannot know whether the difference (b, d) on the left side corresponds to the difference (vw, tu) or the difference (y, z) on the other side without using prior knowledge. Since we have learned that the difference (c, ef) corresponds to the difference (vw, tu) in Example 3, the difference (b, d) must correspond to the difference (y, z). Thus, all difference correspondings are found in our minimal match sequence. The learning heuristic infers the following general template by generalizing the given examples, and the next two atomic templates from the corresponding difference between (b, d) and (y, z).

$X_1\, a\, X_2 \leftrightarrow Y_2\, x\, Y_1$ **if** $X_1 \leftrightarrow Y_1$ **and** $X_2 \leftrightarrow Y_2$
$b \leftrightarrow y$
$d \leftrightarrow z$

where $X_1 a X_2$ is the SLGG of bac and $daef$, and $Y_2 x Y_1$ is the SLGG of $vwxy$ and $tuxz$.

The learned translation templates will be represented in Prolog as follows.

```
tt(S1,S2) :- append(X1,[a],L1), append(L1,X2,S1), append(Y2,[x],L2),
             append(L2,Y1,S2), tt(X1,Y1), tt(X2,Y2).
tt([b],[y]).
tt([d],[z]).
```

5 Application to Example-Based Machine Translation

Our learning heuristic based SLGGs of strings can be used in the learning of translation templates from a given bilingual corpus for two natural languages. In order to learn translation templates, the learning heuristic should be applied to every pair of atomic translation templates in the system. The given translation examples are also treated as atomic translation templates, in fact learning starts from these examples. Learning should continue until no more new templates can be learned from atomic translation templates. The learned translation templates can be used in the translation of other sentences in both directions.

The learning heuristic can work on the surface level representation of sentences. However, in order to generate useful templates, it is helpful to use the lexical representation. In this case, the set of all root words, all prefixes, and all suffixes in a natural language are treated as the alphabet of that language for our purposes. So, a natural language is treated as the set of all meaningful strings on that alphabet. Normally, the given translation examples should be sentences of those natural languages, but they can also be phrases in those languages. Of course, morphological analyzers will be needed for both languages to compose the lexical forms of sentences.

Example 5. *Learning Between English and Turkish*

To explain the behavior of our learning heuristic on the actual natural language sentences, we give a simple learning example for translation examples between English and Turkish. Assume that we have the translation examples '`I will drink water` ↔ `su içeceğim`' and '`I will drink tea` ↔ `çay içeceğim`' between English and Turkish. Their lexical representations are '`I will drink water` ↔ `su iç`+FUT+1SG' and '`I will drink tea` ↔ `çay iç`+FUT+1SG' where +FUT and +1SG denote future tense and first singular agreement morphemes in Turkish, respectively. For these two examples, the minimal match sequence will be '`I will drink (water,tea)` ↔ `(su,çay) iç`+FUT+1SG'. From this match sequence the learning heuristic learns the following three templates by creating SLGGs of the given sentences.

`I will drink` $X_1 \leftrightarrow Y_1$ `iç`+FUT+1SG **if** $X_1 \leftrightarrow Y_1$
`water` ↔ `su`
`tea` ↔ `çay`

In this example, we do not only learn the general pattern in the first clause between English and Turkish; we also learn that `water` corresponds to `su` in Turkish, and `tea` corresponds to `çay`. These clauses will be represented as follows in Prolog.

```
tt(S1,S2) :- append([i,will,drink],X1,L1),
             append(Y1,['iC','+FUT','+1SG'],S2), tt(X1,Y1).
tt([water],[su]).
tt([tea],['Cay']).
```

The learned translation templates can be used in translations in both directions.

6 Conclusion

In this paper, we presented a model for learning translation templates between two languages. The most important part of this model is the learning heuristic based on SLGGs of strings. Translation templates are directly learned from sets of translation examples without using other knowledge resources

such as lexicons, grammars, and ontology. The knowledge resources required for this technique are sets of translation examples and morphological processors for the languages.

We introduced SLGGs of strings to be used instead of LGGs of strings to reduce over-generalization of clauses. The over-generalization can be a serious problem when the learning is done just from positive examples. For example, just using LGGs of strings in example-based machine translation will not be acceptable because of over-generalization problem. We should find learning heuristics which do not cause over-generalization and still perform good generalizations of the given positive examples. In this paper, we have presented one of these techniques to be used in an example-based machine translation system.

We believe that humans learn general sentence patters using similarities and differences between many different example sentences that they are exposed to. This observation led us to the idea that general sentence patterns can be taught to a computer using learning heuristics based on similarities and differences in sentence pairs. In the sense that our mechanism is close to how the humans learn languages from examples.

The learning heuristic presented in this paper is used in an example-based machine translation system [4, 7, 17] between English and Turkish. In this work, we got very promising results. Although in that system, the examples are between English and Turkish, we believe that our techniques are also applicable for other language pairs. In fact, to test this claim we have also applied this technique between English and French using a small set of translation examples. We got comparable results to what we have achieved for the system of English and Turkish.

References

[1] Brona, C., and Padraig, C., Translating Software Documentation by Example: An EBMT Approach to Machine Translation, in: *Proceedings of Int. ECAI Workshop: Multilinguality in the Software Industry*, 1996.

[2] Brown, R. D., Example-Based Machine Translation in the Pangloss System, in: *Proceedings of COLING-96*, 1996.

[3] Brown, R. D., Automated Dictionary Extraction for "Knowledge-Free" Example-Based Translation, in: *Proceedings of TMI'97*, 1997.

[4] Cicekli, I., and Güvenir, H. A., Learning Translation Templates From Bilingual Translation Examples, in: *Applied Intelligence*, Vol 15., No. 1, 2001, pp:57-76.

[5] Furuse, O., and Iida, H., Cooperation between Transfer and Analysis in Example-Based Framework, in: *Proceedings of COLING-92*, Nantes, France, 1992, pp:645-651.

[6] Güvenir, H.A., and Tunç, A., Corpus-Based Learning of Generalized Parse Tree Rules for Translation, in: Gord McCalla (Ed). *New Directions in Artificial Intelligence: Proceedings of the 11th Biennial Conference of the Canadian Society for Computational Studies of Intelligence.* Springer-Verlag, LNCS 1081, Toronto, Ontario, Canada, May 1996, pp:121-131.

[7] Güvenir, H. A, and Cicekli, I., Learning Translation Templates from Examples, in: *Information Systems*, Vol. 23, No. 6, 1998, pp: 353-363

[8] Kaji, H., Kida Y., and Morimoto, Y., Learning Translation Templates from Bilingual Text, in: *Proceedings of COLING-92*, 1992, pp:672-678.

[9] Kitano, H., A Comprehensive and Practical Model of Memory-Based Machine Translation, in: *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, 1993, pp:1276-1282.

[10] Medin, D.L., and Schaffer, M.M., Context Theory of Classification Learning, *Psychological Review, 85*, 1978, pp:207-238.

[11] Mooney, R. J., and Califf M.E., Induction of First-Order decision Lists: Results on Learning The Past Tense of English Verbs, *Journal of Artificial Intelligence Research, 3*, 1995, pp:1-24.

[12] Mooney, R. J., Inductive Logic Programming for Natural Language Processing, in: *Proceedings of the 6th International Workshop on Inductive Logic Programming*, S. Muggleton (ed.), Springer-Verlag, Berlin, 1997, pp:3-21.

[13] Muggleton, S., and Feng, C., Efficient Induction of Logic Programs, in: *Inductive Logic Programming*, S. Muggleton (ed.), Academic Press, London, 1992, pp:281-298.

[14] Muggleton, S., Inductive Logic Programming: issues, results and the challenge of Learning Language in Logic, *Artificial Intelligence 114(1-2)*, 1999, pp:283-296.

[15] Nagao, M. A., Framework of a Mechanical Translation between Japanese and English by Analogy Principle, in: *Artificial and Human Intelligence*, A. Elithorn and R Banerji (eds.), NATO Publications, 1984.

[16] Nirenburg, S., Beale, S., and Domashnev, C., A Full-Text Experiment in Example-Based Machine Translation, in: *Proceedings of the International Conference on New Methods in Language Processing, NeMLap*, Manchester, UK, 1994, pp:78-87.

[17] Öz, Z., and Cicekli, I., Ordering Translation Templates by Assigning Confidence Factors, in: *Lecture Notes in Computer Science 1529*, Springer Verlag, 1998, pp:51-61.

[18] Plotkin, G. D., A Note on Inductive Generalisation, *Machine Intelligence 5*, M. Meltzer and D. Michie (eds.), Elsevier North-Holland, New York, 1970, pp:153-163.

[19] Plotkin, G. D., *Automatic Methods of Inductive Inference*, Ph.D. Thesis, Edinburgh University, Edinburgh, 1971.

[20] Sato, S., and Nagao, M., The Memory-Based Translation, in: *Proceedings of COLING-90*, 1990.

[21] Sato, S., MBT2: A Method for Combining Fragments of Examples in Example-Based Translation, *Artificial Intelligence*, Vol 75, Elsevier Science, 1995, pp: 31-50.

[22] Smadja, F., McKeown, K. R., and Hatzivassiloglou, V., Translating Collocation for Bilingual Lexicons: A Statistical Approach in: *Computational Linguistics*, Vol 22(1), The MIT Press, 1996, pp:1-38.

[23] Sumita, E., and Iida, H., Experiments and Prospects of Example-Based Machine Translation, in: *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, 1991.

[24] Wu, D., Xia, X., Large-Scale Automatic Extraction of an English-Chinese Translation Lexicon in: *Machine Translation*, Vol 9, Kluwer Academic Publishers, 1995, pp:285-313.