# Inducing Translation Templates with Type Constraints

ILYAS CICEKLI
*Dept. of Computer Eng., Bilkent University, Bilkent 06800, Ankara, TURKEY.*
*E-mail: ilyas@cs.bilkent.edu.tr*

**Abstract.** This paper presents a generalization technique that induces translation templates from a given set of translation examples by replacing differing parts in the examples with typed variables. Since the type of each variable is inferred during the learning process, each induced template is also associated with a set of type constraints. The type constraints that are associated with a translation template restrict the usage of the translation template in certain contexts in order to avoid some of the wrong translations. The types of variables are induced using type lattices designed for both the source language and the target language. The proposed generalization technique has been implemented as a part of an EBMT system.

**Keywords:** EBMT, Machine Learning

## 1. Introduction

An example-based machine translation (EBMT) system uses a bilingual corpus to translate a given sentence in a source language into a target language (Nagao, 1984; Somers, 2003). Some EBMT systems use a bilingual corpus to find translations of the parts of a given sentence, and combine these partial solutions to get the translation of the whole sentence. On the other hand, some other EBMT systems (Kaji et al., 1992; Cicekli and Güvenir, 2001; Brown, 2003; Carl, 2003; Cicekli and Güvenir, 2003; McTait, 2003) extract translation templates from example sentences in a given bilingual corpus and use these translation templates in the translation of other sentences. The main differences between these EBMT systems are the assumptions made on the structure of the bilingual corpus and their generalization techniques. The EBMT translation system that uses the generalization technique described in this paper also extracts translation templates from a set of translation examples.

In the EBMT system presented in (Cicekli and Güvenir, 2001; Cicekli and Güvenir, 2003), a translation template is induced from two given 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. Later, any string can replace that variable during the translation process without putting any restriction on the possible replacements.

Although the learned translation template works correctly in certain environments, it can lead to wrong translations in some other unrelated environments because that variable replacement cannot be appropriate in those unrelated environments. In this paper, we propose a generalization heuristic that replaces the differences with variables as well as inducing the *types* of these variables from the differences. Since the types of variables disallow some possible replacements for the variables, the generation of some of the wrong translation results in the unrelated contexts can be avoided.

The type of a variable that replaces a difference is found by using a type lattice for the language of the symbols appearing in the difference. Since the generalization technique described in this paper is used as a part of an EBMT system between English and Turkish, the type lattices for English and Turkish have been developed manually and they are used in the EBMT system. The variables in the induced translation templates are associated with the type names in the type lattices during learning phase. Although the type lattices are created manually, the associations of the variables with the type names in the type lattices are done automatically during the induction of the translation templates. The quality of the induced translation templates also depends on the quality of the type lattices, and the quality of type lattices can be measured experimentally.

The rest of the paper is organized as follows. The structure of translation templates without type constraints is discussed in Section 2. Section 3 introduces the structure of translation templates with type constraints. The generalization process that learns the translation templates with type constraints is presented in Section 4. The systems with and without type constraints are compared in Section 5 by giving the results of some experiments. After the presentation of related work in Section 6, the concluding remarks and possible future extensions are given in Section 7.

## 2. Translation Templates Without Type Constraints

A *language* is a set of strings in the alphabet of that language, and the *alphabet of a language* is a finite set of symbols. For example, a string in a natural language, such as English or Turkish, is a sequence of tokens in that natural language. Each token in a natural language can be a root word or a morpheme. The set of all root words and morphemes in a natural language is treated as its alphabet in our discussions. We also associate each language with a finite set of variables. A *generalized string* is a string of the symbols of the alphabet of the language and

the variables in the set of variables associated with the language. This means that a generalized string is a string that contains at least one variable. We assume that each language is associated with a different set of variables. A string without variables is called a *ground string*.

A translation template can be an *atomic* or *general translation* template. An *atomic translation template* $T_a \leftrightarrow T_b$ between languages $L_a$ and $L_b$ is a pair of two nonempty strings $T_a$ and $T_b$ where $T_a$ is a ground string in $L_a$ and $T_b$ is a ground string in $L_b$. An atomic translation template $T_a \leftrightarrow T_b$ means that the strings $T_a$ and $T_b$ correspond to each other. A given *translation example* is an atomic translation template.

A *general translation template* between languages $L_a$ and $L_b$ is an if-then rule in the following form:

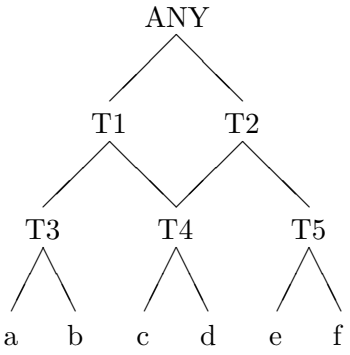
$T_a \leftrightarrow T_b$ **if** $X_1 \leftrightarrow Y_1$ **and** ... **and** $X_n \leftrightarrow Y_n$

where $n \geq 1$, $T_a$ is a generalized string of the language $L_a$, and Tb is a generalized string of the language $L_b$. Both $T_a$ and $T_b$ must contain $n$ unique variables. The variables in $T_a$ are $X_1...X_n$, and the variables in $T_b$ are $Y_1...Y_n$. Each generalized string ($T_a$ and $T_b$) in a general translation template should contain at least one symbol from the alphabet of the language of that string.

For example, if the alphabet of $L_a$ is $A = \{a, b, c, d, e, f, g, h\}$ and the alphabet of $L_b$ is $B = \{t, u, v, w, x, y, z\}$, the examples in (1) are well-formed translation templates between $L_a$ and $L_b$.

(1) $de \leftrightarrow vyz$
$abX_1c \leftrightarrow uY_1$ **if** $X_1 \leftrightarrow Y_1$
$aX_1X_2b \leftrightarrow Y_2vY_1$ **if** $X_1 \leftrightarrow Y_1$ **and** $X_2 \leftrightarrow Y_2$

The first translation template is an atomic translation template, and the last two are general translation templates. The first atomic translation template means that $de$ in the language $L_a$ and $vyz$ in the language $L_b$ correspond to each other. A general translation 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, the generalized string $abX_1c$ in the second example in (1) represents all sentences of $L_a$ starting with $ab$ and ending with $c$ where $X_1$ represents a non-empty string in $A$, and the generalized string $uY_1$ represents all sentences of $L_b$ starting with $u$ where $Y_1$ represents a non-empty string in $B$. That general template says that a sentence of $L_a$ in the form of $abX_1c$ corresponds to a sentence of $L_b$ in the form of $uY_1$ given that $X_1$ corresponds to $Y_1$. If we know the correspondence $de \leftrightarrow vyz$, the correspondence $abdec \leftrightarrow uvyz$ can be inferred from that general template.




- Ground Type Names = {a,b,c,d,e,f}
- The set of ground type names is also the alphabet of this simple language.
- The sets of ground type names represented by some type names.
  $GT_a$ = {a}
  $GT_{T3}$ = {a,b}
  $GT_{T1}$ = {a,b,c,d}
  $GT_{T2}$ = {c,d,e,f}
  $GT_{ANY}$ = {a,b,c,d,e,f}

*Figure 1.* A Type Lattice for A Simple Language

A well-formed general translation template contains $n$ unique variables in both sides of the translation template, and each variable in one side of the translation template must correspond to a variable in the other side. For example, "$abX_1c \leftrightarrow uY_1vY_2\ if\ X_1 \leftrightarrow Y_1$" is not a well-formed translation template because the left side contains one variable, and the right side contains two variables. Another ill-formed translation template is "$aX_1bX_2c \leftrightarrow uY_1vY_2\ if\ X_1 \leftrightarrow Y_1\ and\ X_1 \leftrightarrow Y_2$" because the variable $X_1$ in the left side corresponds to two different variables, and the variable $X_2$ does not correspond to any variable.

## 3. Translation Templates With Type Constraints

### 3.1. TYPE EXPRESSIONS

All symbols in the alphabet of a language are organized as a *type lattice*. The symbols in the alphabet of the language appear at the bottom of this type lattice. In fact, each symbol is treated as a *ground type name* that represents itself in the type lattice. Inner nodes in the lattice are *type names* that are used for the language, and each type name represents a set of ground type names. Thus, a ground type name represents a singleton set containing that ground type name. At the top of the lattice, there is a special type name, called ANY. The type name ANY represents the set of all ground type names in the language. If $t$ is a type name, we say that $GT_t$ is the set of the ground type names that are covered by $t$. Each node in the lattice, except ANY, can have one or more parents. If node $P$ is a parent of node $C$ in the type lattice, $GT_P \supset GT_C$ holds. Figure 1 gives a type lattice for a simple language.

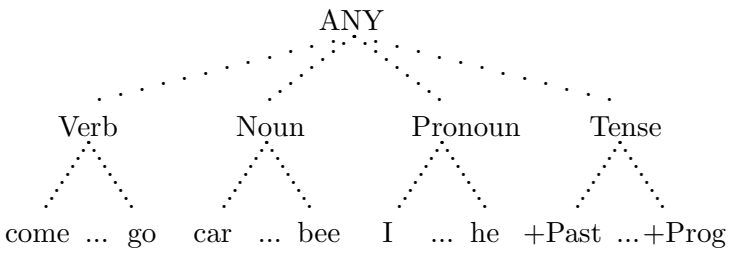



*Figure 2.* A Simplified Type Lattice for English

Since type name T1 is the parent of type name T3, $GT_{T1} \supset GT_{T3}$ holds true for the type lattice in Figure 1.

Each variable of a generalized string in *a general translation template with type constraints* is associated with a type expression, and that type expression is called the type of that variable. *The type of a variable* indicates the possible ground strings that can replace the variable during the translation process. A *type expression* is a non-empty sequence of atomic type expressions. An *atomic type expression* can be either $T$ or $nullor(T)$ where $T$ is a type name from the type lattice. If the type of a variable is a type name $T$, this means that the variable can be replaced by a ground type name from $GT_T$. In the second case where the type of a variable is $nullor(T)$, the variable is replaceable with an empty string in addition to a ground type name from $GT_T$. In other words, $GT_{nullor(T)}$ is equal to $GT_T \cup \{\epsilon\}$.

The definition of $GT$ can be extended for the type expressions that consist of more than one atomic type expression. If a type expression $T$ is an atomic type sequence $T_1...T_n$, $GT_T$ is equal to the concatenation of the sets $GT_{T_1}$ through $GT_{T_n}$. In general, a variable of type $T$ is replaceable with a ground string from $GT_T$. For example, let us consider the simple language and its type lattice in Figure 1. If the type of a variable is type T3, this means that it can be replaced with a ground string from $GT_{T3}$={a,b}. When the type of a variable is $nullor(T3)$, it can be replaced with an empty string or a string from $GT_{T3}$. A variable of the type ANY can be replaced with any ground type name. If a type expression $T$ is an atomic type sequence "T3 T4", $GT_T$ is equal to {ac,ad,bc,bd}.

The type lattices for English and Turkish are manually created, and they are be used in the developed EBMT system. Simplified partial type lattices for these languages can be seen in Figure 2 and Figure 3. Major type names in each type lattice are the part of speech tags used for the language. The affixes used in a language are also considered as major type names. For example, the major part of speech tags such as

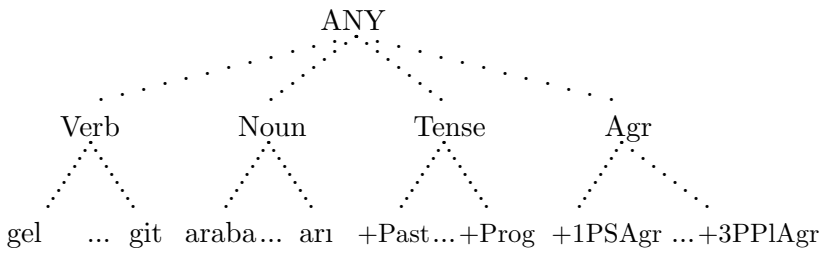



*Figure 3.* A Simplified Type Lattice for Turkish

noun, verb, pronoun and adjective are major type names in the English type lattice, and they appear as the children of ANY. The type names between major type names and ground type names generally represent the subgroups of part of speech tags. The affixes are grouped according to where they can be used. For example, the set of suffixes that can be added to verbs is considered as a major type name. The arcs in the figures are given as dotted lines, because there can be other nodes on those paths. These simplified type lattices are used in the examples in the rest of the paper, and we treat the dotted lines in the figures as the straight lines.

The English type lattice that we created is similar to the morphological type hierarchy used in HPSG (Pollard and Sag, 1994). Our English type lattice can be seen as a simplified morphological type hierarchy. At the bottom of the type lattice, there are stems and affixes, and they are treated as ground type names since they cannot be a parent of another type name in the type lattice. The stems are organized as a morphological type hierarchy mainly based on their part of speech tags. The affixes are also organized as a type hierarchy based on their functionalities. The Turkish type lattice is also a morphological type hierarchy for Turkish. In the Turkish type lattice, the type hierarchy of the affixes is more complex with respect to the English type lattice because Turkish is a morphologically complex language. For example, the inflectional suffixes that can follow Turkish nouns are grouped with respect to their functionalities such as agreement markers, possessive markers and case markers.

Although the major nodes in the type lattice are part of speech tags, there are also type names to represent smaller and larger groups. For example, there are type names for numbers and ordinals. The type names representing small groups can help avoiding over generalization. For example, "+Past" and "+Prog" are tense morphemes, and they can only follow the verbs in Turkish. In the Turkish type lattice, their immediate parent is the type name "Tense", the parent of "Tense"

is "VerbSuffix", and the parent of "VerbSuffix" is "Suffix". The morphemes "+Past" and "+Prog" are generalized as "Tense" according to this type hierarchy, because "Tense" is their immediate parent. Thus, this finer type hierarchy can avoid the over generalization of these symbols.

## 3.2. TRANSLATION TEMPLATES WITH TYPE CONSTRAINTS

A *translation template with type constraints* is a general translation template where all variables are associated with type expressions. A translation template with type constraints is a translation template in the following form:

$$T_a \leftrightarrow T_b \textbf{ if } X_1^{T_{A_1}} \leftrightarrow Y_1^{T_{B_1}} \textbf{ and } ... \textbf{ and } X_n^{T_{A_n}} \leftrightarrow Y_n^{T_{B_n}}$$

where each of $T_{A_1}, ..., T_{A_n}$ and $T_{B_1}, ..., T_{B_n}$ is a type expression. A translation template with type constraints also puts a restriction on the possible replacements of variables during the translation process. For example, the template in (2) is a translation template with type constraints.

(2) I $X^{Verb}$ +Past $\leftrightarrow$ $Y^{Verb}$ +Past +1PSAgr **if** $X^{Verb} \leftrightarrow Y^{Verb}$

This general template represents the fact that an English sentence in the form of "I $X^{Verb}$ +Past" corresponds to a Turkish sentence in the form of "$Y^{Verb}$ +Past +1PSAgr" given that $X$ and $Y$ are translations of each other with respect to the translation templates. This template also specifies that $X$ can only be replaced by a verb on English side, and $Y$ can only be replaced by a verb on Turkish side. In this example, "+Past" means the past tense suffix on both English and Turkish sides, and "+1PSAgr" on Turkish side is the first person singular agreement suffix.

The translation template in (2) can be used in the translation of the Turkish sentence into the English sentence in (3) if the correspondence "gel↔come" is available with respect to the translation templates. During the translation process, both variables are replaced by English and Turkish verbs without violating type constraints in the translation template.

(3) geldim ⇒ I came
gel+Past+1PSAgr I come+Past

Type constraints in the translation templates restrict wrong usages of templates in certain circumstances. For example, if we try to use the

translation template in (2) without using type constraints, it may lead to wrong translation results. Let us assume that we want to translate the Turkish sentence in (4) into English using the translation template in (2) without any type constraints.

(4) utangaçtım (I was shy)
utangaç+Past+1PSAgr

Without using the type restrictions, variable Y on Turkish side can match with "utangaç" which is an adjective (not a verb). If the correspondence "shy↔utangaç" is available, variable X on English side can match with "shy" (not a verb). Thus, it can lead to the meaningless translation result "I shy +Past" at the lexical level. Type constraints in the translation template will avoid this wrong translation by rejecting the binding of Y with "utangaç" which is an adjective.

During the translation process, the variables in the source language portion of a translation template are bound to the parts of the given sentence that will be translated. The string that a variable is bound to must satisfy the type constraint that is imposed by the variable. Otherwise, the translation template cannot be used in the translation of the sentence. Then, the string that a variable is bound to is translated, and the translation result must satisfy the type constraint that the corresponding variable in the target language portion of the translation template imposes. Otherwise, that translation result is rejected. For example, if we use the translation template in (2) to translate the English sentence "I come+Past" into Turkish, the variable $X^{Verb}$ is bound to the string "come". Before the string "come" is translated, it must satisfy the type constraint "Verb" that is imposed by the type of the variable $X^{Verb}$. Then, the string "come" is translated into Turkish. The translation results must satisfy the type constraint "Verb" that is imposed by the corresponding variable $Y^{Verb}$. In other words, we only accept the translation results that are Turkish verbs, and reject all other translations of "come".

Every word in a given source language sentence is morphologically analyzed by the source language morphological analyzer in order to create the lexical level representation of the input sentence. There can be more than one lexical level representation of the sentence because of the morphological ambiguity. Then, the translation results are found using the translation templates in the system for all lexical representations of the input sentence. The produced translation results are in the lexical representation, and the target language morphological generator finds the surface level representations of the translation results. We use our own versions of morphological processors for Turkish and English.

## 4. Learning Translation Templates

In the EBMT system described in (Cicekli and Güvenir, 2001; Cicekli and Güvenir, 2003), translation templates are inferred without type constraints from the given translation examples. Each translation example consists of an English sentence and a Turkish sentence, and their lexical level representations are used for the sentences.

In order to induce a translation template from given two translation examples $E_a^1 \leftrightarrow E_b^1$ and $E_a^2 \leftrightarrow E_b^2$, we first find the match sequence $M_a \leftrightarrow M_b$ where the match sequence $M_a$ is a match sequence between $E_a^1$ and $E_a^2$, and the match sequence $M_b$ is a match sequence between $E_b^1$ and $E_b^2$. A match sequence between two sentences is a sequence of similarities and differences between those sentences. A similarity between two sentences is a non-empty sequence of common items in both sentences. A difference between two sentences is a pair of two sequences $(D_1, D_2)$ where $D_1$ is a sub-sequence of the first sentence and $D_2$ is a sub-sequence of the second sentence, and $D_1$ and $D_2$ do not contain any common item.

For example, the two examples in (5) are translation examples between English and Turkish sentences. The lexical level representations of the sentences are used, and common parts in the sentences are underlined.

(5) I come +Past ↔ gel +Past +1PSAgr
I go +Past ↔ git +Past +1PSAgr

From the two examples in (5), the match sequence in (6) is found. In the match sequence in (6), (come,go) is a difference on English side, (gel,git) is a difference on Turkish side, and the remaining parts of the match sequence are similarities.

(6) (come,go) +Past ↔ (gel,git) +Past +1PSAgr

One of the learning heuristics described in (Cicekli and Güvenir, 2001; Cicekli and Güvenir, 2003) infers a translation template by replacing differences by variables and establishing bindings between these variables. This learning heuristic can create a translation template if both sides of the match sequences contain $n$ fferences where $n \geq 1$ and the correspondences of $n-1$ difference pairs have been already learned. For example, for the match sequence in (6), this learning heuristic infers the three translation templates in (7).

(7) I $X$ +Past ↔ $Y$ +Past +1PSAgr **if** $X \leftrightarrow Y$
come ↔ gel
go ↔ git

The first translation template in (7) is a general translation template created by replacing differences with variables $X$ and $Y$. The last two translation templates are atomic translation templates and they are inferred from the correspondence of the differences (come,go) and (gel,git).

Variables $X$ and $Y$ in the translation template in (7) do not have any type constraints, and they are replaceable with any ground strings as long as they are translations of each other during the translation process. As we discussed in Section 3, this can lead to wrong translation results in unrelated environments. In order to reduce the amount of wrong translation results, translation templates are associated with type constraints. In the rest of this section, we describe how translation templates with type constraints are inferred from given translation examples.

### 4.1. INFERRING A TYPE EXPRESSION FOR TWO SYMBOLS

When we replace a difference with a variable, we should also find a type expression for that variable. If both constituents of a difference are symbols (strings with length 1), the type expression for those symbols is found using the type lattice of that language, and the found type expression is used as a type constraint for the variable replacing that difference. For example, when we infer a translation template from the match sequence in (6), we also infer types of the variables replacing the differences (come,go) and (gel,git). Of course, we use English type lattice for the difference (come,go), and Turkish type lattice for the difference (gel,git).

If we have two symbols in a difference, they should be ground type names in the type lattice of the language of those symbols. For example, the symbols *come* and *go* in the difference (come,go) are ground type names in English type lattice. Since the variable replacing the difference (come,go) represents the symbols *come* and *go*, the type of this variable should cover both of these symbols. We say that a ground type $gt$ is covered by a type $t$, if $gt \in GT_t$. Thus, if there is a type $T$ that covers both symbols *come* and *go*, both $come \in GT_T$ and $go \in GT_T$. In the worst case, type ANY covers any given two ground type names in a language.

In general, there can be more than one type covering any given two type names. Since we do not want to over generalize, we select the most specific type covering both of them. We say that type $T_2$ is more specific than type $T_1$ if $GT_{T_1} \supset GT_{T_2}$ holds. This means that $T_1$ is one of the ancestors of $T_2$. So, if both $T_1$ and $T_2$ cover given type names

and $T_2$ is more specific than $T_1$, $T_2$ is selected as a type expression for the given type names.

In some cases, there can be two ancestors $T_1$ and $T_2$ of a given pair of type names $A$ and $B$, and neither $GT_{T_1} \supset GT_{T_2}$ nor $GT_{T_2} \supset GT_{T_1}$ holds. So, the youngest one of $T_1$ and $T_2$ is selected to represent $A$ and $B$. In order to find a youngest ancestor of two given types, the shortest path containing one of their ancestors is found and the ancestor on that shortest path is the youngest ancestor of them. A type is also considered as an ancestor of itself. Thus, the youngest ancestor of types $T_1$ and $T_2$ will be $T_1$ if $T_1$ is an ancestor of $T_2$.

According to English type lattice in Figure 2, the youngest ancestor of *come* and *go* is type "Verb", and the youngest ancestor of *gel* and *git* is type "Verb" according to Turkish type lattice in Figure 3. So, the translation template with type constraints in (8) is induced from the match sequence in (6). In addition to this template, two atomic translation templates in (8) are also induced.

(8) I $X^{Verb}$ +Past $\leftrightarrow$ $Y^{Verb}$ +Past +1PSAgr **if** $X^{Verb} \leftrightarrow Y^{Verb}$
come $\leftrightarrow$ gel
go $\leftrightarrow$ git

A difference $(t_1, t_2)$, where $t_1$ and $t_2$ are two different type names in the type lattice, is generalized as a type name $t_3$ if $t_3$ is the youngest ancestor of $t_1$ and $t_2$. Each generalization has a generalization score to indicate the amount of the generalization. We use the length of the shortest path between $t_1$ and $t_2$ as a generalization score. For example, the generalization score of the difference (come,go) as "Verb" is 2 because the length of the shortest path between *come* and *go* is 2 according to the simplified English type lattice in Figure 2. In fact, when a difference is generalized, the generalization with the smallest generalization score is selected as its generalization. We say that $gen(come, go)$ is the generalization of the difference (come,go), and $genscore(come, go)$ is the generalization score of this generalization.

Because of homonym and the structure of the type lattice, a type name can have multiple parents in the type lattice. For example, the word $fly$ has "Verb" and "Noun" as its parents in the English type lattice. The difference (fly,swim) is generalized as "Verb" because "Verb" is the youngest ancestor of $fly$ and *swim*. On other hand, the difference (fly,eagle) is generalized as "Noun" because "Noun" is the youngest ancestor of $fly$ and *eagle*.

## 4.2. INFERRING A TYPE EXPRESSION FOR TWO STRINGS

If a difference has a constituent whose length is greater than one, the generalization of that difference cannot be an atomic type expression. If $n$ is the length of the longest constituent of a difference, its generalization is a type expression consisting of $n$ atomic type expressions. If a difference is $(a_1...a_n, b_1...b_n)$ where the lengths of the constituents are equal, the generalization $gen(a_1...a_n, b_1...b_n)$ is equal to $gen(a_1, b_1)...gen(a_n, b_n)$. The generalization score $genscore(a_1...a_n, b_1...b_n)$ for this generalization is equal to $genscore(a_1, b_1) + ... + genscore(a_n, b_n)$.

If the lengths of constituents are different, we have to consider different possibilities and some symbols have to be generalized with empty strings. For example, we have to consider the three generalizations in (9) for the difference $(abc, de)$.

(9) $gen(a, d)gen(b, e)gen(c, \epsilon)$
$gen(a, d)gen(b, \epsilon)gen(c, e)$
$gen(a, \epsilon)gen(b, d)gen(c, e)$

When there is more than one possible generalization for a difference, we select the one with the smallest generalization score. Since we assume that we have an imaginary type for each type name in the type lattice such that it is a parent of that type name and the empty string, the score of the generalization of a symbol with the empty string is assumed to be 2. The generalization of a symbol $a$ and the empty string is represented by $nullor(a)$.

Let us consider the two translation examples in (10). For these examples, the match sequence in (11) is found.

(10) I come +Past ↔ gel +Past +1PSAgr
I am go +Prog ↔ git +Prog +1PSAgr

(11) I (come +Past, am go +Prog) ↔
(gel +Past, git +Prog) +1PSAgr

In order to select the generalization for the difference (come +Past, am go +Prog), we have to consider the three generalizations in (12).

(12) gen(come,am) gen(+Past,go) gen($\epsilon$,+Prog)
gen(come,am) gen($\epsilon$,go) gen(+Past,+Prog)
gen($\epsilon$,am) gen(come,go) gen(+Past,+Prog)

Since the last generalization has the smallest generalization score, it is selected as the generalization for this difference. So, the generalization for this difference is the type expression "nullor(am) Verb Tense".

Similarly, the difference (gel +Past, git +Prog) has only one possible generalization that is "gen(gel,git) gen(+Past,+Prog)". Thus, the generalization for the difference (gel +Past, git +Prog) will be the type expression "Verb Tense". As a result, the translation template with type constraints in (13) is inferred from these two translation examples. In addition to this translation template, two more atomic translation templates in (13) are also inferred.

(13) I $X^{nullor(am)\ Verb\ Tense} \leftrightarrow Y^{Verb\ Tense}$ +1PSAgr
**if** $X^{nullor(am)\ Verb\ Tense} \leftrightarrow Y^{Verb\ Tense}$
come +Past ↔ gel +Past
am go +Prog ↔ git +Prog

## 4.3. INDUCING TEMPLATES FROM LEARNED TEMPLATES

We learn not only the translation templates from examples, but also learn new translation templates from the previously induced templates by generalizing them. The induced templates are treated as translation examples containing the typed variables and symbols. In order to achieve the induction of new translation templates from these templates, two typed variables are treated as the same symbol if their types are the same. Thus, the typed variables are treated as symbols and we are able to apply our learning technique to the previously induced templates. Let us consider the two translation templates in (14).

(14) I $X^{Verb}$ +Past ↔ $Y^{Verb}$ +Past +1PSAgr
**if** $X^{Verb} \leftrightarrow Y^{Verb}$
You $X^{Verb}$ +Past ↔ $Y^{Verb}$ +Past +2PSAgr
**if** $X^{Verb} \leftrightarrow Y^{Verb}$

Although the variables $X^{Verb}$ in these two templates may represent different symbols in actual translation examples, these two symbols are treated as the same symbol since the type of both is "Verb". For example, the first template may be induced from the first two translation examples in (15), and the second translation template may be induced from the last two translation examples in (15).

(15) I come +Past ↔ gel +Past +1PSAgr
I go +Past ↔ git +Past +1PSAgr
You sleep +Past ↔ uyu +Past +2PSAgr
You come +Past ↔ gel +Past +2PSAgr

As a result of these generalizations, the variable $X^{Verb}$ in the first template represents "come" and "go", but the variable $X^{Verb}$ in the

second template represents "sleep" and "come". Since both of them represent verbs, we treat them as the same symbol during the induction process.

When we try to learn a translation template from two previously induced translation templates, we first find the match sequence of the heads of these two translation templates. For example, the match sequence of the heads of the two translation templates in (14) is given in (16). From the match sequence in (16), we induce the three translation templates in (17). In the first template in (17), $X_1^{Pronoun}$ is the generalization of the difference (I,you), and $Y_1^{Agr}$ is the generalization of the difference (+1PSAgr ,+2PSAgr).

(16) (I,you) $X^{Verb}$ +Past $\leftrightarrow$ $Y^{Verb}$ +Past (+1PSAgr ,+2PSAgr)

(17) $X_1^{Pronoun}$ $X_2^{Verb}$ +Past $\leftrightarrow$ $Y_2^{Verb}$ +Past $Y_1^{Agr}$
**if** $X_1^{Pronoun} \leftrightarrow Y_1^{Agr}$ **and** $X_2^{Verb} \leftrightarrow Y_2^{Verb}$
I $\leftrightarrow$ +1PSAgr
you $\leftrightarrow$ +2PSAgr

In (14), the variables $X^{Verb}$ and $Y^{Verb}$ in both of the translation templates end up in similarities in the match sequence of these translation templates. Their correspondence in the example translation templates in (14) is copied into the body of the newly induced translation template in (17). The first translation template in (17) can be seen as a further generalization of the translation templates in (14).

In general, a variable ends up in a similarity or a difference of a match sequence. Let us assume that the first example translation template has only one corresponding variable pair $X \leftrightarrow Y$ and the second example translation template has only one corresponding variable pair $Z \leftrightarrow W$. If $X$ and $Z$ end up in a similarity (i.e. $X$ and $Z$ are of the same type variable), our learning heuristic insists that $Y$ and $W$ must end up in a similarity too (i.e. $Y$ and $W$ must have the same type too). In this case, the constraint $X \leftrightarrow Y$ (which is equal to $Z \leftrightarrow W$) in the body of the first translation example also appears in the body of the newly learned translation template. If $X$ and $Z$ end up in a difference $(\alpha_1 X \beta_1, \alpha_2 Z \beta_2)$, $Y$ and $W$ must end up in a difference $(\gamma_1 Y \delta_1, \gamma_2 W \delta_2)$ too, and these two differences must be the corresponding differences. In this case, these two differences are replaced with appropriate typed variables $A$ and $B$, and the constraint $A \leftrightarrow B$ appears in the body of the newly induced translation template. The type of $A$ is the generalization of the difference $(\alpha_1 X \beta_1, \alpha_2 Z \beta_2)$, and the type of $B$ is the generalization of the difference $(\gamma_1 Y \delta_1, \gamma_2 W \delta_2)$. In addition to the newly induced translation template, the following two translation templates in (18) are induced from these corresponding differences.

(18) $\alpha_1 X \beta_1 \leftrightarrow \gamma_1 Y \delta_1$ **if** $X \leftrightarrow Y$
$\alpha_2 Z \beta_2 \leftrightarrow \gamma_2 W \delta_2$ **if** $Z \leftrightarrow W$

For example, let us assume that the two translation templates in (19) have been induced previously from the translation examples in (20). The first translation template in (19) can be learned from the first two translation examples in (20), and the second translation template in (19) can be learned from the last two examples in (20).

(19) I am a $X^{Noun} \leftrightarrow$ bir $Y^{Noun}$ +1PSAgr
**if** $X^{Noun} \leftrightarrow Y^{Noun}$
I am $X^{Verb}$ +Prog $\leftrightarrow Y^{Verb}$ +Prog +1PSAgr
**if** $X^{Verb} \leftrightarrow Y^{Verb}$

(20) I am a student ↔ bir öğrenci +1PSAgr
I am a tailor ↔ bir terzi +1PSAgr
I am go +Prog ↔ git +Prog +1PSAgr
I am come +Prog ↔ gel +Prog +1PSAgr

The match sequence of the heads of the translation templates in (19) is the match sequence in (21). From the match sequence in (21), we induce the first translation template in (22) by generalizing the difference (a $X^{Noun}$,$X^{Verb}$ +Prog) as $X^{ANY\ ANY}$ and the difference (bir $Y^{Noun}$,$Y^{Verb}$ +Prog) as $Y^{ANY\ ANY}$. In addition to the first template in (22), the last two translation templates in (22) are also induced from the corresponding differences in (21).

(21) I am (a $X^{Noun}$,$X^{Verb}$ +Prog) ↔
(bir $Y^{Noun}$,$Y^{Verb}$ +Prog) +1PSAgr

(22) I am $X^{ANY\ ANY} \leftrightarrow Y^{ANY\ ANY}$ +1PSAgr
**if** $X^{ANY\ ANY} \leftrightarrow Y^{ANY\ ANY}$
a $X^{Noun} \leftrightarrow$ bir $Y^{Noun}$ **if** $X^{Noun} \leftrightarrow Y^{Noun}$
$X^{Verb}$ +Prog $\leftrightarrow Y^{Verb}$ +Prog **if** $X^{Verb} \leftrightarrow Y^{Verb}$

The first translation template in (22) is a general form of the translation templates in (19). Thus, it can be used in the translation of other sentences in addition to the sentences in the form of the translation templates in (19). For example, the first translation template can be used in the translation of "I am fly +Prog" into "uç +Prog +1PSAgr" if "uç +Prog" is the translation of "fly +Prog". These English and Turkish sentences are in the form of the second translation template in (19). Although the sentence "I am very fast" is not in the form of any of the translation templates in (19), the first translation template in

(22) can also be used in the translation of this English sentence into the Turkish sentence "çok hızlı +1PSAgr" if "çok hızlı" is the translation of "very fast".

The last two translation templates in (22) can be used in the translation of the subparts of the sentences. For example, the third translation template in (22) can be used in the translation of "fly +Prog" into "uç +Prog" if "uç" is the translation of "fly". Of course, the second and third templates in (22) can be used in the translation of the parts of the sentences, and those sentences can be in the form of the first template in (22) or some other translation template.

## 5. Experiments

In order to see the effects of the variables with types, we compare both versions of our system. The first version uses translation templates without type constraints, and the second version uses translation templates with type constraints. They can translate English sentences into Turkish sentences, and Turkish sentences into English sentences. During the translation process, both versions produce a set of translation results for a given sentence.

The translation results are sorted with respect to their specificity factors. Each translation template is associated with a specificity factor in each translation direction (English to Turkish, Turkish to English). A specificity factor of a translation template depends on the number of symbols in the source language part of the translation template. The usage of specificity factors helps the correct solution to appear among the first translation results.

We tested both of our systems with a bilingual corpus between English and Turkish. The training set contains 4152 sentence pairs, and the test set contains 1039 sentence pairs. The sentences in the test set are structurally similar to the sentences in the training set, and they are relatively short sentences. The length of the longest English sentence is 17 symbols, and the average length of English sentences is 7.2 symbols. The length of the longest Turkish sentence is 21 symbols, and the average length of Turkish sentences is 8.4 symbols. Each symbol is either a stem or a morpheme. Since the training data contains the correspondences of some English and Turkish words, the minimum sentence lengths are 1 for both languages. The results of the experiments are given in Tables I and II.

The first columns in the tables indicate the average number of translation results that are produced by the systems per sentence. The numbers for the system without type constraints are much higher

Table I. Translation Results from English to Turkish

| | Avg. # of Results per sentence | Recall Corr.Sol. appears in Results | Corr.Sol. appears in First Pos. of Results | Corr.Sol. appears in First Three Pos. of Results | Corr.Sol. appears in First Five Pos. of Results | BLEU Score |
|---|---|---|---|---|---|---|
| Without Type Constraints | 328 | 93% | 55% | 76% | 88% | 0.79 |
| With Type Constraints | 4.5 | 91% | 66% | 89% | 90% | 0.82 |

than the numbers for the system with type constraints. This means that the system without type constraints produces many incorrect results together with the correct solutions. The main reason is the over generalization in the system without type constraints. Since the type constraints put extra restrictions in the usage of translation templates, the system with type constraints eliminates most of the incorrect translations.

The second columns in the tables indicate whether the correct translations appear in the produced translation results or not. The values in these columns can be seen as *recall* results. For example, the percentage of whether the correct solutions appear in translation results for the system without type constrains is 93 percent. In other words, this means that the correct translations did not occur among the produced translation results for the 7 percent of the sentences in the test set. Recall results are 2-3 points lower for the system with type constraints. This means that the extra restrictions cause to miss some of the correct translations.

The column 3 gives the percentage of the number of correct translations appearing in the first position of the produced translation results. Although the percentage for the system without type constraints is 55 percent, the percentage for the system with type constraints is 66 percent in the English to Turkish direction. This means that some of the top ranked wrong translations are eliminated by the type constraints, and we get this 11 percent increase. The increase in the Turkish to English direction is 14 percent according to the numbers in Table II. If

Table II. Translation Results from Turkish to English

| | Avg. # of Results per sentence | Recall Corr.Sol. appears in Results | Corr.Sol. appears in First Pos. of Results | Corr.Sol. appears in First Three Pos. of Results | Corr.Sol. appears in First Five Pos. of Results | BLEU Score |
|---|---|---|---|---|---|---|
| Without Type Constraints | 413 | 93% | 45% | 72% | 82% | 0.72 |
| With Type Constraints | 5.3 | 90% | 59% | 78% | 89% | 0.78 |

the translation system is used to return only one result, the numbers in the third columns explain the performance of the system.

The columns 4 and 5 give the similar percentages for the cases in which correct solutions appear in the first three or five results, respectively. If the translation system is used to return top translation results and a human selects the actual translation from these top results, the numbers in columns 4-5 explain the performance of the system. In both directions of the translation, the system with type constraints produces more top translation results containing the correct solutions. This means that the type constraints push the correct translation into top translation results by eliminating some incorrect translations among the top translation results.

The last columns of the tables give BLEU scores of both systems. When we evaluate BLEU scores, we assume that each sentence (given in the test set) has only one correct solution and we pick the first translation in the produced translations as the result of the translation. Under these assumptions, we use the same methods described in (Papineni et al., 2002) in the evaluation of BLEU scores. The results in the tables indicate that the system with type constraints gets better BLEU scores. This means that the obtained translation results are much closer to the correct translations.

According to the numbers given in the last four columns of Tables I and II, both of the translation systems perform better in the English to Turkish direction. One of the observed reasons for this performance difference in the translation directions is the usage of the third per-

son singular pronouns in English and Turkish. The three third person singular pronouns (he/she/it) map to a single third person singular agreement morpheme in Turkish. During the translation from Turkish to English, one of these three pronouns is selected and it may not be the correct solution.

The results of the experiments presented in this section validate our intuition that type constraints improve the precision by eliminating the incorrect translation from the produced translation results. This can be observable from the precision results in columns 3-5, and BLEU scores in column 6. On the other hand, the system with type constraints may miss a few of the correct translations because of extra restrictions. The system with type constraints induces more translation templates than the system without type constraints because the same template without type constraints can appear more than once with different type constraints.

## 6. Related Work

The method presented in this paper generalizes the given examples by replacing their differing parts with variables in order to create translation templates. The variables in the induced templates are associated with types, and these types indicate the morphological categories of the strings that can replace those variables in the translation process. The induced translation templates with type constraints are used in the translation of other sentences in the translation process.

The system described in (Furuse and Iida, 1995) generalizes given translation examples as abstract translation templates. The method described in (Kaji et al., 1992) also generalizes examples to create translation templates with variables, and these variables represent the syntactic categories of the possible replacements for those variables. In order to create translation templates from aligned translation pairs in (Kaji et al. 1992), they parse the translation examples and align the syntactic units in the examples. According to the method described in (Carl, 1999; Carl, 2003), the examples are generalized based on their syntactic categories and morphological features. The method described in (Brown, 2003) also induces transfer rules, and the transfer rules can be combined into equivalence classes using word-level clustering. The main difference between our method and the mentioned methods is that we use type lattices in the generalization process in order to find the morphological categories of the variables in the induced translation templates.

The EBMT system in (Matsumoto and Kitamura, 1995) induces the translation rules based on semantic categories. The variables in the generalized rules are associated with semantic categories. The generalizations are performed according to similarities determined by thesauri. The similar methods based on semantic categories are also described in (Nomiyama, 1992; Akiba et al., 1995; Almuallim et al., 1994) . Although the system described in this paper only generalizes the examples according to morphological categories, it can be extended to generalize them according to semantic categories. For example, we may use WordNet in order to find the semantic categories of differing parts.

## 7. Conclusion

In this paper, we have presented a learning technique that induces translation templates from given translation examples, by replacing the differing parts with variables. Types of variables are also learned from the replaced differing parts during the training phase. The types of variables help to reduce the amount of wrong translation results by restricting the usage of the translation templates in unrelated contexts.

The learning heuristic described in this paper has been implemented as a part of an EBMT system between English and Turkish. When the translation results of the EBMT system using translation templates with type constraints are compared with the translation results of the EBMT system using translation templates without type constraints, it can be seen that the type constraints have eliminated more wrong translations from the translation results. The average number of translation results per sentence is approximately 5 sentences for the system with type constraints, and it is approximately 300 sentences for the system without type constraints. This means that there are a lot of wrong translations in these 300 sentences, and most of them are eliminated in the system with type constraints. In addition, the percentage of the number of correct translations in the top positions of the produced translations is also increased because some of the highly ranked wrong translation results are eliminated from the translation results.

The type expression that is inferred for a variable replacing a difference with two symbols depends on the shortest path between those two symbols in their type lattice. The youngest ancestor of the symbols is a generalization of the difference. By selecting the youngest ancestor for the symbols, we hope that we get the most specific generalization for them. The youngest ancestor may not be the most specific generalization depending on the symbols and the structure of the type lattice. Although there can be other techniques to find the most specific gen-

eralization, the shortest path is one of the good techniques. There are also other possible generalization techniques (Resnik, 1995; Budanitsky and Hirst, 2001) that can be used in our problem domain, and some of them are used to measure semantic similarity in a taxonomy such as WordNet (Fellbaum, 1998).

The type of a variable is a sequence of the type names in the type lattice and represents a specific generalization of the strings in the difference that the variable replaced. If we do not use any type constraint for a variable, it will be the most general generalization for those strings. We may prefer an intermediate generalization for them between the specific one and the most general one. In this case, the regular expressions can be a better choice to represent type expressions. We are currently investigating these alternatives.

In this paper, the constraints for the variables are type constraints. The generalization technique described here can also be used in the inference of the semantic constraints if the semantic lattices, which are similar to WordNet, are available for the source and target languages. The quality of translation templates will depend on the quality of the used semantic lattices, and the quality of the lattice can be checked experimentally.

## Acknowledgements

This work is partially supported by The Scientific and Technical Council of Turkey Grant "TUBITAK EEEAG-105E065".

## References

Akiba, Y., M. Ishii, H. Almuallim, and S. Kaneda: 1995, Learning English Verb Selection Rules from Hand-made Rules and Translation Examples. In *Proceedings of the Sixth International Conference on Theoretical and Methodological Issues in Machine Translation (TMI95)*, Leuven, Belgium, pp: 206-220.

Almuallim, H., Y. Akiba, A. Yamazaki, A. Yokoo, and S. Kaneda: 1994, Two Methods for Learning ALT-J/E Translation Rules from Examples and a Semantic Hierarchy. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING-94)*, Kyoto, Japan, pp: 57-63.

Brown, R. D.: 2003, Clustered Transfer Rule Induction for Example-Based Translation. In *Recent Advances in Example-Based Machine Translation*, Carl, M., and Way, A. (eds.), The Kluwer Academic Publishers, Boston, pp: 287-306.

Budanitsky, A., and G. Hirst: 2001, Semantic Distance in WordNet: An Experiment, Application-oriented Evaluation of Five Measures, In *Proceedings of WordNet and Other Lexical Resources Workshop (NAACL 2001)*, Pittsburgh, pp: 29-34.

Carl, M.: 1999, Inducing Translation Templates for Example-Based Machine Translation. In *Machine Translation Summit VII*, Singapore, pp: 250-258.

Carl, M.: 2003, Inducing Translation Grammars from Bracket Alignments. In *Recent Advances in Example-Based Machine Translation*, Carl, M., and Way, A. (eds.), The Kluwer Academic Publishers, Boston, pp: 339-361.

Cicekli, I., and H. A. Güvenir: 2003 Learning Translation Templates from Bilingual Translation Examples. In *Recent Advances in Example-Based Machine Translation*, Carl, M., and Way, A. (eds.), The Kluwer Academic Publishers, Boston, pp: 255-286.

Cicekli, I., and H. A. Güvenir: 2001, Learning Translation Templates from Bilingual Translation Examples. *Applied Intelligence* **15**, 57-76.

Fellbaum, C. (Ed.): 1998, Wordnet: An Electronic Lexical Database, MIT Press.

Furuse, O., and H. Iida: 1995, 'An Example-Based Method for Transfer Driven Machine Translation. In *Proceedings of the Sixth International Conference on Theoretical and Methodological Issues in Machine Translation (TMI95)*, Leuven, Belgium, pp: 139-150.

Kaji, H., Y. Kida, and Y. Morimoto: 1992, Learning Translation Templates from Bilingual Text. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING-92)*, Nantes, France, pp: 672-678.

McTait, K.: 2003, Translation Patterns, Linguistic Knowledge and Complexity in EBMT. In *Recent Advances in Example-Based Machine Translation*, Carl, M., and Way, A. (eds.), The Kluwer Academic Publishers, Boston, pp: 307-338.

Matsumoto, Y., and M. Kitamura: 1995, Acquisition of Translation Rules from Parallel Corpora. In *Recent Advances in Natural Language Processing*, Amsterdam, John Benjamins, pp: 405-416.

Nagao, M.A.: 1984, Framework of a Mechanical Translation between Japanese and English by Analogy Principle. In *Artificial and Human Intelligence*, Elithorn, A., and Banerji, R. (eds.), North Holland, Amsterdam, pp: 173-180.

Nomiyama, H.: 1992, Machine Translation by Case Generalizations. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING-92)*, Nantes, France, pp: 714-720.

Papineni, K., S. Roukos, T. Ward, and W. Zhu: 2002, 'BLEU: a Method for Automatic Evaluation of Machine Translation. In *40th Annual meeting of the Association for Computational Linguistics*, Philadelphia, pp: 311-318.

Pollard, C., and I. Sag: 1994, Head-Driven Phrase Structure Grammar, University of Chicago Press.

Resnik, P.: 1995, Using Information Content to Evaluate Semantic Similarity in a Taxonomy, *Journal of Artificial Intelligence Research* **11**, pp: 448-453.

Somers, H.: 2003, An Overview of EBMT. In *Recent Advances in Example-Based Machine Translation*, Carl, M., and Way, A. (eds), The Kluwer Academic Publishers, Boston, pp: 3-57.