TreeTran: A Tool for Visual Selection and Testing of Transfer Rules for Machine Translation

David A. Bullock

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Arts

University of Washington

2006

Program Authorized to Offer Degree:
Department of Linguistics

University of Washington
Graduate School

This is to certify that I have examined this copy of a master's thesis by

David A. Bullock

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Fei Xia

_____

H. Andrew Black

Date: _____

Signature: _____

Date: _____

TABLE OF CONTENTS

LIST OF FIGURES

Figure Number Page

LIST OF TABLES

1 Introduction

Transfer rules encode the structural changes needed to translate a parse tree from one language into an equivalent parse tree in another language. When machine translation involves two languages with sufficient electronic resources, such as parallel treebank or sentence corpora, statistical machine-learning techniques can be used to identify transfer rules. For minority languages with little or no written text, however, transfer rules will need to be specified by a human translator.

This paper describes the development of a tool that aids a human translator in identifying, testing and applying transfer rules. Chapter 3 explains the transfer methodology and algorithms used by the system, Chapter 4 describes the system design and user interface in greater detail, and in Chapter 5, the system is compared to two other baseline systems in a small-scale translation task.

2 Previous Work

Machine-translation systems can approach sentence data at various levels of analysis. This is illustrated in Figure 2.1, adapted from (Knight, 1997). At the lowest level in the diagram, sentence data is represented as text. Moving higher in the diagram, the data is represented with increasing degrees of syntactic and semantic analysis.

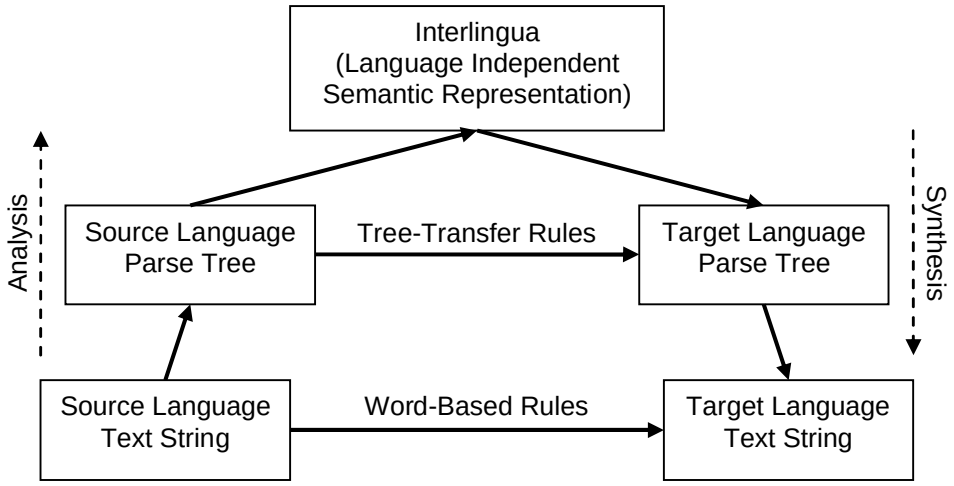

Figure 2.1 Different Strategies for Machine Translation

Some of the translation work can be performed with strictly surface-based methods. For example, the translation system can use a bilingual dictionary to look up each word in the source-language sentence and replace it with an equivalent word in the target language. But word-for-word substitution can yield very poor translations, especially if the source and target languages have different basic word orders, such as SVO vs. SOV. Some systems use surface-based rules to fix some of the word-order issues, swapping the order of nouns and adjectives for example. These methods can work for simple sentences, but fail to address longer embedded phrases and long-distance dependencies.

Structural differences between the source and target languages can be handled more effectively by using the Analysis-Transfer-Synthesis model (Slocum, 1985). In this model, a source sentence is analyzed to produce a parse tree that represents its syntactic structure, and perhaps some semantic features. Transfer rules describe the structural and feature changes needed to produce a grammatical parse-tree representation for an equivalent sentence in the target language. Finally, the synthesis step uses this target parse tree to generate the sentence translation in the target language.

A set of tree-transfer rules, which may be complex and time-consuming to produce, can be used only for a specific pair of source and target languages. If a system needs to translate to and from multiple languages, a set of transfer rules is needed for each pair of source and target languages. This effort might be reduced by using an interlingua, which represents meaning in a language-independent way. If we can output an interlingual representation by parsing sentences, we can then use this interlingual representation to generate sentence translations in multiple other languages. We only need one parser and one generator for each language. Although this technique has been used successfully on a few narrow-domain tasks, such as translating weather reports and product manuals (Knight, 1997), the goal of designing a general-purpose interlingua remains an unsolved problem.

Since a useful system of transfer-based rules can take years of manual effort to produce, corpora-based machine-learning strategies have been employed to automate much of the work in developing machine-translation systems. Example-Based Machine Translation (EBMT) uses sentence-aligned bilingual corpora to extract target words and phrases that are statistically established as correct translations of words and phrases in the source language (Nagao, 1984). Statistical Machine Translation (SMT) learns translation and distortion

probabilities from sentences in bilingual corpora (Brown et al, 1993) and then uses a combination of models, such as a translation model, an alignment model and a language model of the target language, to choose between the multiple potential translations that it generates from the translation and distortion data (Och and Ney, 2002). Other systems learn syntactic tree-transfer rules by comparing the alignment of constituents in the parses of parallel bilingual sentences (Menezes and Richardson, 2001), and from these rules, statistical methods are used to pick the best set (Meyers, Kosaka and Grishman, 2000). In a rule-learning system, the identified transfer rules can be easy for a human to interpret and possibly extend (Kaji, Kida and Morimoto, 1992). Transfer rules may map the source-to-target changes using single-level context-free rules or multi-level sub-tree patterns (Lavoie, White and Korelsky, 2002) or by using paths in dependency trees (Lin, 2004).

The machine-learning strategies mentioned above generally rely on large parallel corpora containing tens of thousands or even millions of sentences. Unfortunately, this kind of data is not available for minority languages, which may have little or no electronic text in existence. Some success has been achieved in learning tree-transfer rules from much smaller corpora, containing only a couple hundred sentences (Probst, 2005), but the corpora must be carefully constructed to show examples of the desired language structures, and the words in the sentences must be manually aligned. Transfer rules are only learned for constituents that are likely to be translated into constituents of the same type (ADJ, ADV, NP, PP, SBAR and S, but not VP), so these rules must be supplemented by additional manually-crafted transfer rules.

Development is underway on linguistic tools for a number of minority languages using the PC-PATR syntactic parser (McConnel, 1995) and the AMPLE morphological parser (McConnel, Black and Doornenbal, 2006). The

parse trees can be produced as XAMPLE files, using XML that conforms to the "xample.dtd" specification, to represent the syntactic structures and features. To facilitate translations between these languages and other related languages, a tree-transfer system, compatible with PC-PATR and AMPLE, was proposed (Bergman, 1996). The proposal, which was never fully implemented, suggested writing tree-editing rules in a text-based format, as described in Figure 2.2, with examples in Figure 2.3.

The text-based tree-editing rules can be one of the following:

```
NodePattern '==>' 'move' number Position number
NodePattern '==>' 'insert' Position number NodePattern
NodePattern '==>' 'delete' number
NodePattern '==>' 'enrich' number '[' Features ']'
NodePattern '==>' 'remove' number '[' Features ']'
```

where:

NodePattern → name ['#' number] ['[' Features ']'] ['(' ChildNodes ')']

Features → name '=' string [Features]

ChildNodes → NodePattern [ChildNodes] | '...' [ChildNodes]

Position → 'before' | 'after' | 'below'

The specified tree editing is performed for each sub-tree that matches the NodePattern on the left-hand side of the rule. The right-hand side of the rule specifies nodes to be moved, inserted or deleted, or features to be enriched or removed. Nodes are referenced by the optional identifying number following the node's name.

Features appear within square brackets. Child nodes appear within parenthesis, and can be nested. A matching sub-tree must match all the specified features and nodes in the pattern. An ellipsis pattern '...' can match zero or more child nodes.

Figure 2.2 Description of Text-Based Tree-Editing Rules

```
\rule Accusative case
    NP#1 [dir_obj=+] ==> enrich 1 [case=ACC]
\rule Remove all articles
    ART#2 ==> delete 2
\rule Move verb to beginning of VP
    VP#2 (... NP ... V#1) ==> move 1 below 2
\rule Move PP after NP
    VP (... PP#1 NP#2 ...) ==> move 1 after 2
```

Figure 2.3 Example Text-Based Tree-Editing Rules

Writing tree-transfer rules in a text editor can be time-consuming and error-prone. As described below, a tree-transfer system for XAMPLE data has been implemented, but it takes a new approach to make the rule-writing process more efficient: Instead of typing a text-based transfer rule, the user looks at example parse trees, visually selects a sub-tree pattern, and then edits a copy of the sub-tree to show what it should look like in the target parse. The pattern and target sub-trees are saved as a transfer rule. In addition, the system provides an environment for debugging transfer rules by stepping through each rule and visually highlighting the changes it makes.

3 TreeTran Methodology

I have developed a computer program, called TreeTran.exe, as the Transfer component of an Analysis-Transfer-Synthesis model of machine translation, shown in Figure 3.1. The system is intended for rough-draft translations between closely-related minority languages, and it is assumed that a human post-editor will be needed to correct the final translation. In addition to an engine that applies the transfer rules, TreeTran also provides a visual user interface for selecting, editing and debugging transfer rules.

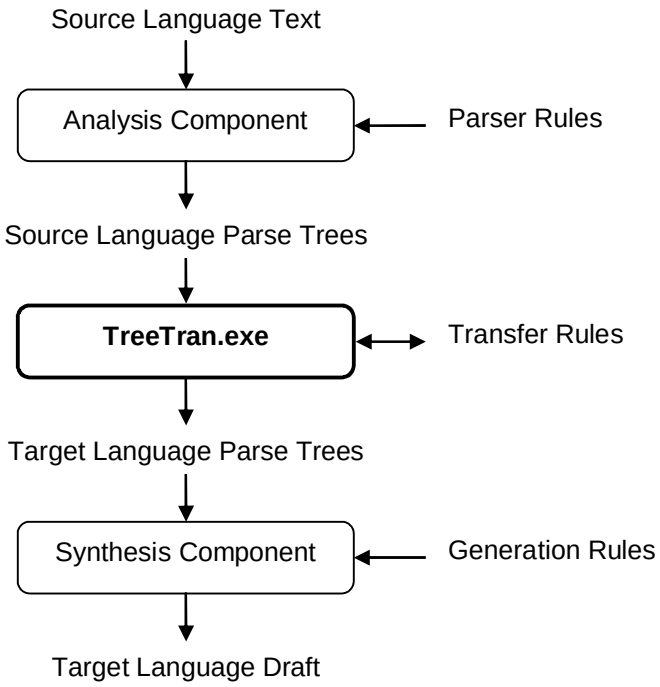

Figure 3.1 Analysis-Transfer-Synthesis Model

The system uses an ordered list of transfer rules that each consist of two sub-trees: a pattern sub-tree to match, and a replacement sub-tree indicating the changes to make to the matching parse-tree branch. The sub-trees can be any depth, and multiple features are allowed on each node. The order of transfer rules

is important, since a branch replaced by one rule may be matched and further modified by a subsequent rule.

3.1 TreeTran Rules

Figure 3.2 describes the tree-transfer rule structure I developed for use by the TreeTran program.

A TreeTran rule consists of a find pattern and a replace pattern, each of which is represented as a sub-tree node pattern:

TreeTranRule → <NodePattern,NodePattern>

Each node in a sub-tree pattern consists of a label string, a set of zero or more features (pairs of name and value strings) and an ordered set of zero or more child nodes (each a sub-tree pattern):

NodePattern → <label,Features,ChildNodes>

Features → { <name,value>, <name,value>, ... }

ChildNodes → { <0,NodePattern>, <1,NodePattern>, ... <n,NodePattern> }

A parse-tree branch is matched if each node in the find-pattern sub-tree matches the corresponding node in the branch. Each pattern-node feature must have the same value as the corresponding parse-node feature (or a non-matching value if the pattern's feature value starts with "!"). If the pattern node has children, all the children of the parse node must match the children of the pattern node (possibly skipping optional pattern nodes, which have labels that start with "("). Each optional node in the find-pattern can match at most one parse-tree node.

A matching parse-tree branch is replaced with a new branch specified by the replace-pattern sub-tree. Additional node features are copied from each corresponding parse-tree node (matched by a find-pattern node with the same label as the replace-pattern node). If a replace-pattern node has no children but there are children on the corresponding parse-tree node (matched by a find-pattern node with the same label), these parse-tree children are recursively copied as sub-trees in the new branch.

Figure 3.2 Description of TreeTran Rules

TreeTran's find and replace patterns are displayed as easy to understand trees, which can specify multiple changes to nodes and features in a single rule. An example of this is shown in Figure 3.3. In contrast, the text-based tree-editing

rules that were described earlier in Figure 2.2 (Bergman, 1996) use bracketed strings to specify sub-trees, and may require multiple rules to make related changes to multiple nodes and features. Figure 3.4 shows a set of text-based rules that are nearly equivalent to the single TreeTran rule in Figure 3.3.

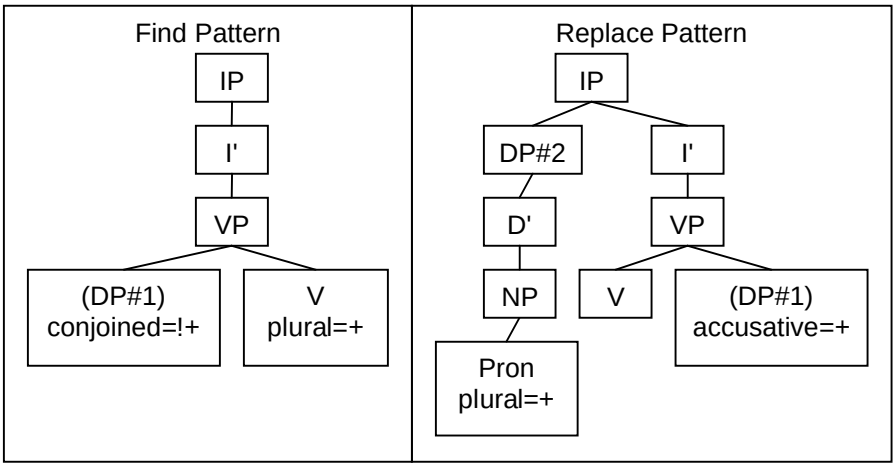

Figure 3.3 Multiple Changes with a Single TreeTran Rule

```
\rule Add Accusative Feature
IP ( I' ( VP ( DP#1 [conjoined=-]
               V [plural=+] ) ) )
==> enrich 1 [accusative=+]

\rule Insert Plural Pronoun
IP ( I'#1 ( VP ( ...
                 V [plural=+] ) ) )
==> insert before 1
DP ( D' ( NP ( Pron [plural=+] ) ) )

\rule Swap Verb and Object
IP ( DP ( D' ( NP ( Pron [plural=+] ) ) )
     I' ( VP ( DP#1 [conjoined=- accusative=+]
               V#2 [plural=+] ) ) )
==> move 1 after 2
```

Figure 3.4 Related Changes Requiring Multiple Text-Based Rules

Figure 3.3 and Figure 3.4 also illustrate other differences between the two systems of tree-transfer rules. Instead of supporting an ellipsis ("..."), TreeTran

supports optional nodes (such as "(DP#1)"). An advantage in doing this is that constraints can be placed on the optional nodes, so we can specify the number, order and kind of optional nodes that can match.

Each TreeTran rule can make multiple structural and feature changes, and can also constrain and manipulate multiple optional nodes, so we can do more with a single rule. There are fewer situations where we need multiple rules to perform a single kind of change. As a result, a set of TreeTran rules can be less susceptible to rule-ordering and rule-interaction problems.

3.2 TreeTran Algorithm

The basic transfer algorithm used by TreeTran is described in Figure 3.5. For each rule, the algorithm traverses every node of the parse tree in post-order (parent after its children), looking for all branches that match the find pattern. Each matching branch is replaced as specified by the replace pattern. The tree is traversed in post-order to prevent infinite recursion that could occur if an XP node was replaced by a node with XP children, and then the rule was applied recursively to these children.

```
for each parse_tree p:
    for each rule <find_pattern_root f,replace_pattern_root r>:
        n = first node of p in post-order.
        while n ≠ null:
            clear MatchingNodes stack.
            if Match(n,f):
                replace branch n with new branch Replace(r).
            n = next node of p in post-order.
```

Figure 3.5 Pseudo-code for Find-and-Replace Algorithm

The algorithm's two main functions, Match() and Replace(), both use the MatchingNodes stack. The Match() function fills this stack with matching node pairs (each consisting of a parse-tree node and the find-pattern node it matched).

The Replace() function uses this stack to determine how to replace the matching branch.

Figure 3.6 summarizes the Match() function, which recursively compares the given parse-tree branch to the indicated find pattern. Matching nodes are pushed onto the MatchingNodes stack. If a non-matching node is reached, the Match() function attempts to backtrack in the MatchingNodes stack to find an optional node that can be skipped.

```
Match(parse_tree_node n,find_pattern_node f)
    // Match features.
    if all the features of f match the features of n:
        push node_pair <n,f> onto MatchingNodes stack.
    else:
        return false.
    // Match children.
    nc = first child of n.
    fc = first child of f.
    while nc ≠ null and fc ≠ null:
        if Match(nc,fc):
            nc = next child of n.
            fc = next child of f.
        else:
            if fc is not an optional node:
                // Backtrack to optional node.
                fc = null.
                if MatchingNodes stack contains a child of f
                    that is an optional node:
                        pop MatchingNodes stack to that node_pair.
                        assign nc and fc to the nodes in that node_pair.
            // Skip optional node.
            fc = next child of f.
    if all children matched:
        return true.
    else:
        pop MatchingNodes stack to <n,f>.
        return false.
```

Figure 3.6 Pseudo-code for Match Function

Figure 3.7 summarizes the Replace() function, which recursively traverses the indicated replace pattern, creating a replacement for the matching branch. When a

replace-pattern node has the same label as a find-pattern node on the MatchingNodes stack, the features and children of the corresponding parse-tree node are copied to create part of the replacement branch.

```
Replace(replace_pattern_node r)
    // Copy features.
    find node_pair <n,f> on the MatchingNodes stack
        where f has the same label as r.
    if this <n,f> was found:
        new node p = copy of node n.
        copy features from r to p.
    else:
        new node p = copy of node r.
    // Copy children.
    if r has children or f has children:
        for each child rc of r:
            rc = Replace(rc).
            add rc as a child of p.
    else
        for each child nc of n:
            pc = Replace(nc).
            add pc as a child of p.
    return p.
```

Figure 3.7 Pseudo-code for Replace Function

4 TreeTran System Design

This chapter describes the design and implementation of the TreeTran program in more detail.

4.1 User Interface

The TreeTran program provides a visual user interface that allows the user to view parse trees, copy and paste branches to create and edit transfer rules, and then apply transfer rules to highlight the changes they make to example parse trees. The TreeTran program also provides a batch mode to apply transfer rules to an entire file of parse trees.

4.1.1 Command Line

The TreeTran program can be run in interactive mode to create, edit and debug a file of tree-transfer rules, using the following command-line arguments:

TreeTran.exe [rules.xml [parses.xml]]

where:

rules.xml = the file of tree-transfer rules to edit.

parses.xml = an input file of XAMPLE parse trees produced by the Analysis Component. This optional file is loaded so the user can copy parts of the parse trees to use as transfer rules, and so the user can debug transfer rules by stepping through their application to example parse trees.

The TreeTran program can also be run in batch mode to process an entire parse file by applying a set of transfer rules, using the following command-line arguments:

TreeTran.exe rules.xml parses.xml output.xml

where:

rules.xml = the file of tree-transfer rules to use.

parses.xml = the input file of XAMPLE parse trees produced by the Analysis Component.

output.xml = the name of the output file to which XAMPLE parse trees will be written after they have been modified by the tree-transfer rules. This file can then be read by the Synthesis Component.

4.1.2 TreeTran Window

The main TreeTran window, show in Figure 4.1, contains two child windows: one that displays transfer rules, and one that displays parses.

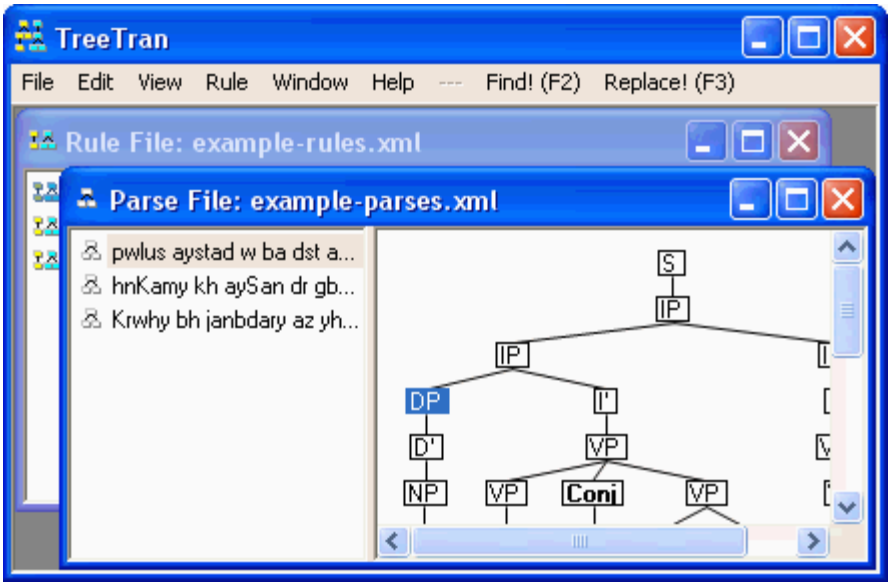

Figure 4.1 TreeTran Window

4.1.3 Rule Window

The Rule Window displays a list of rules in the panel on the left side of the window. The selected rule is displayed in two panels on the right side of the window. Each rule consists of two trees: a Find Pattern tree and a Replace Pattern tree.

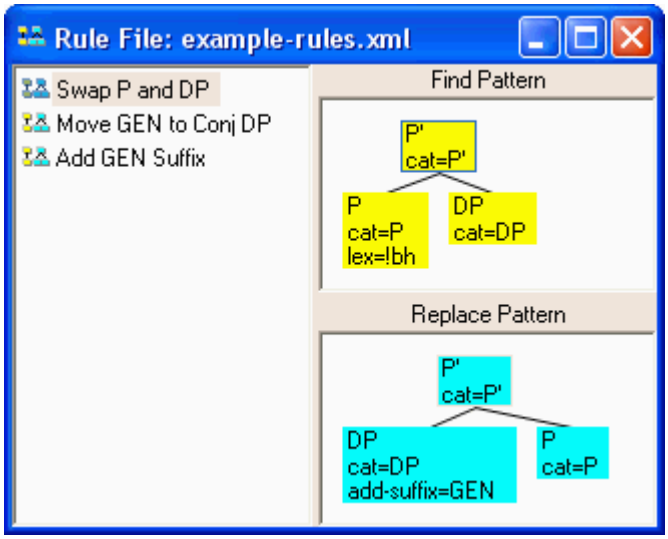

Figure 4.2 Rule Window

In Figure 4.2, the first rule, named "Swap P and DP", is selected. Its Find Pattern consists of a "P"' node dominating two other nodes: a "P" node followed by a "DP" node. The "lex" feature of the "P" node must not match the value "bh" (an exclamation mark indicates match negation). If the Find Pattern matches a sub-tree of a parse, the Replace Pattern indicates how the tree will be modified. In this case, the order of the "P" and "DP" children is switched, and the "DP" node is given a new feature named "add-suffix" with value "GEN".

4.1.4 Parse Window

The Parse Window displays a list of parses in the panel on the left side of the window. The selected parse tree is displayed in a panel on the right side of the window. The parse tree can optionally show morphology and node features, as seen in Figure 4.3.

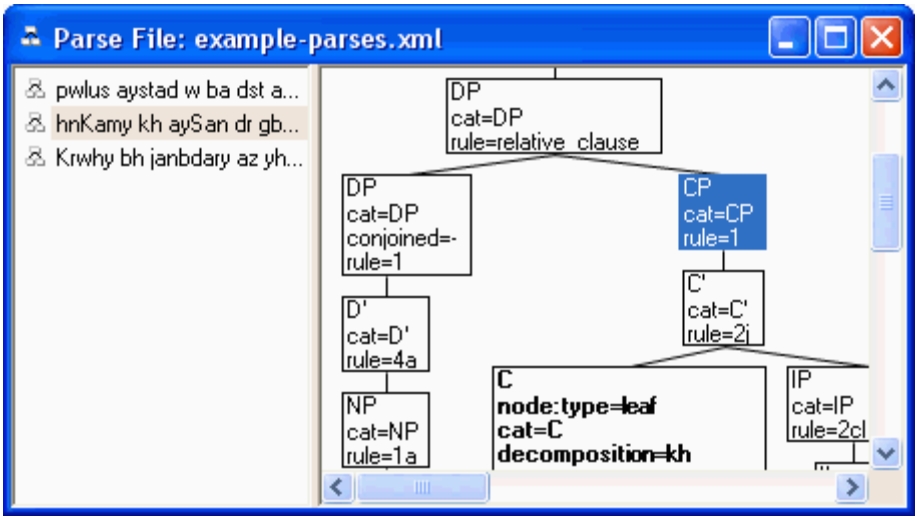

Figure 4.3 Parse Window

Sub-trees can be selected and copied from the tree displayed in the Parse Window, and then pasted and edited in the Rule Window to create the find and replace patterns of transfer rules.

The Parse Window is also useful for interactive debugging of transfer rules. When the user clicks the **Rule - Find Matching Branch** menu, the nodes that are matched by a rule's Find Pattern are highlighted in yellow in the displayed parse tree. For example, Figure 4.4 shows the three highlighted nodes that match the Find Pattern of the "Swap P and DP" rule (shown in Figure 4.2 above).

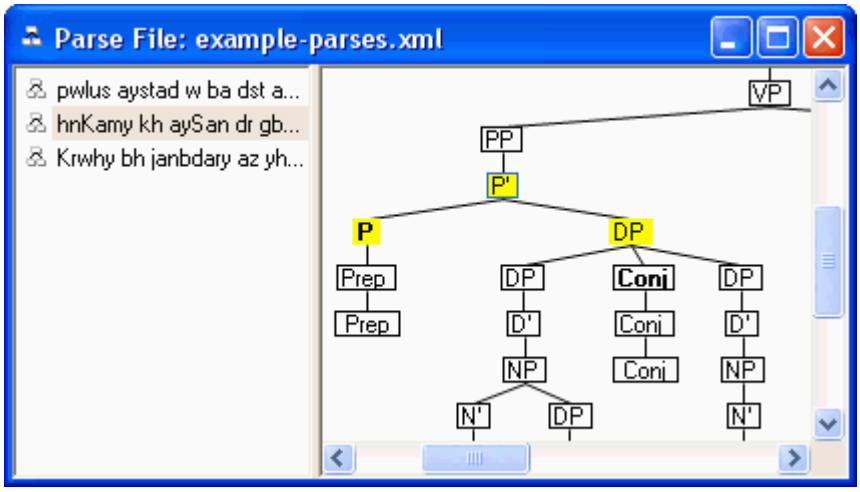

Figure 4.4 Find Pattern Highlighted in Parse Tree

When the user clicks the **Rule - Replace Matching Branch** menu, the nodes that correspond to the rule's Replace Pattern are highlighted in blue in the displayed parse tree. Figure 4.5 shows the highlighted nodes indicating the Replace Pattern changes made by the "Swap P and DP" rule.

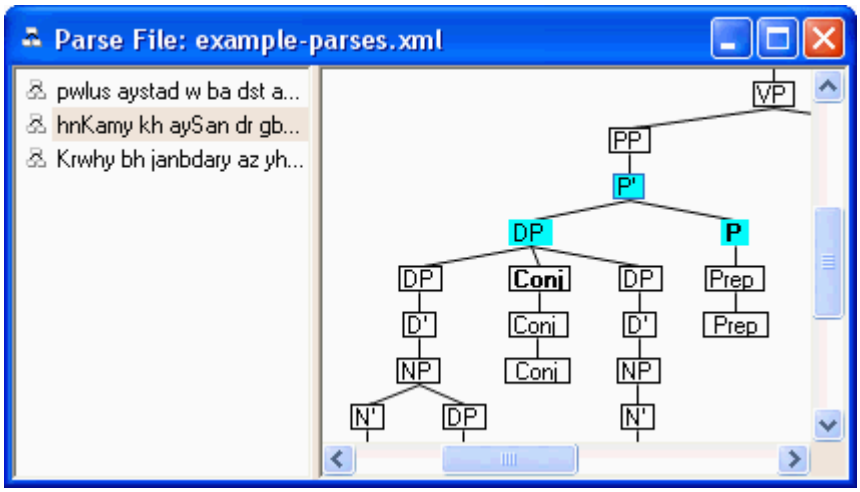

Figure 4.5 Replace Pattern Highlighted in Parse Tree

4.1.5 Menus

The lists and trees in the Rule Window and Parse Window can be edited and manipulated with the menus on the main TreeTran window:

File - Open Rule File.... Opens an xml file containing transfer rules.

File - Save Rule File. Saves the displayed transfer rules to the same file as before.

File - Save Rule File As.... Saves the displayed transfer rules to a new file.

File - Load Parse File.... Opens parse trees from an XAMPLE xml file.

File - Save Parse File As.... Saves the displayed parse trees to a new file.

File - Process Parse File.... Applies the displayed transfer rules to each of the parses in the file you select, and then saves the changes to a new file.

File - Exit. Closes the TreeTran program.

Edit - Undo. Undoes the last edit operation.

Edit - Cut. Cuts the selected node or list item to the clipboard.

Edit - Copy. Copies the selected node or list item to the clipboard.

Edit - Paste. Pastes the node or list item from the clipboard. When you paste a node, you append the branch on the clipboard as a child of the selected node. (If the selected node is blank, you replace the node with the branch on the clipboard).

Edit - Delete. Deletes the selected node or list item.

Edit - Features.... Opens the Features Window, which lets you edit the features of the selected node. Double-clicking a node also opens this dialog.

View - Show Features. Toggles the display of features in the parse tree.

View - Show Morphology. Toggles the display of morphology in the parse tree.

View - Font.... Changes the font used to display the lists and trees.

Rule - Find Matching Branch. Finds the next sub-tree that matches a Find Pattern. (You can also use the **Find!** menu or the F2 key to do the same thing.)

Rule - Replace Matching Branch. Replaces the matching sub-tree with the nodes specified by the Replace Pattern. (You can also use the **Replace!** menu or the F3 key to do the same thing.)

Rule - Find Only with Current Rule. Checks or unchecks this menu so the search for a matching sub-tree will use all rules or only the current rule.

Rule - Find Only in Current Parse. Checks or unchecks this menu so the search for a matching sub-tree will look in all parses or only in the current parse.

Rule - New Rule. Appends a new rule to the list of transfer rules.

Rule - Rename. Lets you rename the current rule in the list of transfer rules.

Help - Quick Help.... Displays a page of help topics.

Help - About TreeTran.... Displays the program's version number.

4.1.6 Features Window

The user can edit node features by double-clicking the node (or by clicking the **Edit - Features...** menu), which brings up the Features Window, shown in Figure 4.6.

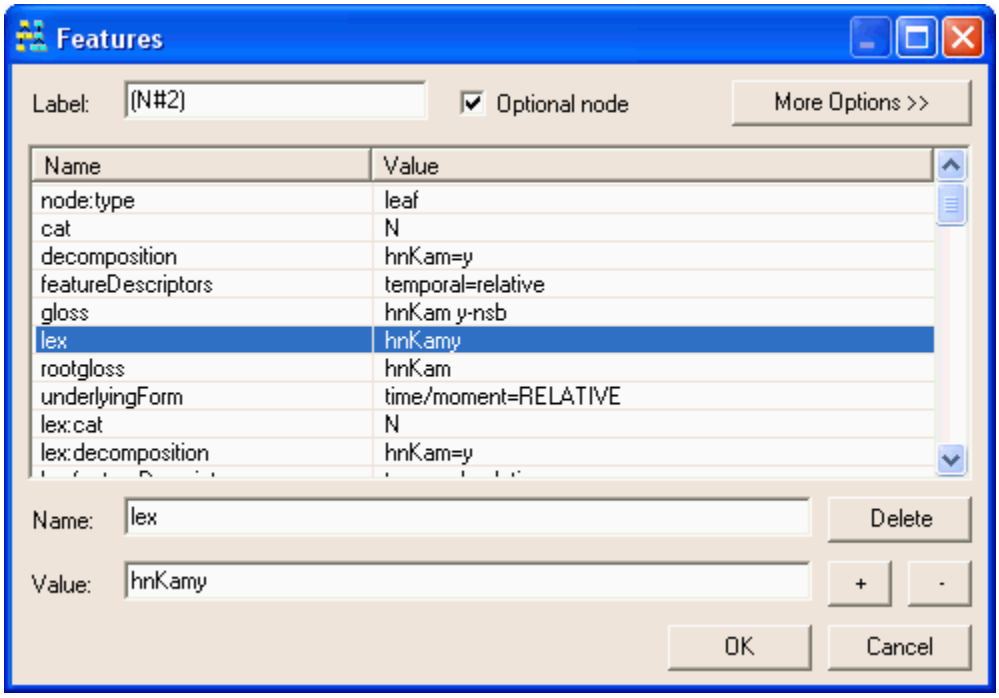

Figure 4.6 Features Window

The Features Window displays the following controls for viewing and editing node features:

The **Label** text box lets the user edit the node's label. By default the label is the same as the node's category, but a number will automatically be appended (for example, "PP#3") if a node is pasted into a pattern where that label already exists. Find Pattern and Replace Pattern nodes correspond if they have the same label.

The **Optional node** check box indicates whether the node is optional or not. The label of an optional node begins with an open parenthesis (for example, "(DP)").

The **Name / Value** list box displays the node's features.

The **Name** text box lets the user enter the name of a feature.

The **Value** text box lets the user modify the value of the feature indicated by the **Name** text box.

The **Delete** button deletes the features selected in the list box.

The **+** and **-** buttons change a feature's value to + or -.

The **OK** button closes the Features Window and saves the changes made to the node's features.

The **Cancel** button closes the Features Window without saving the changes.

By clicking the **More Options >>** button, several additional controls are displayed:

The **Filter** text box allows the user to enter a string, which restricts the list box so it only displays features with names containing that string.

The **Show favorites only** check box restricts the list box so it only displays features that have been marked as favorites.

The **Show all favorites** check box displays all features marked as favorites, including features that do not appear on this node. This can be useful for setting a new node feature or changing which features are marked as favorites.

The **Favorite** check box marks a feature as a favorite.

The **Show in tree** check box marks features to be displayed in the parse-tree view.

The **Copy favorites only** check box indicates whether all features or only favorites will be copied to the clipboard when a sub-tree is cut or copied.

4.1.7 Find Pattern

The TreeTran engine does a post-order (parent after its children) traversal of each parse tree, looking for a sub-tree that matches all the non-optional nodes of the Find Pattern (and possibly some of the optional nodes). Each optional node in the Find Pattern can match at most one parse-tree node.

For a parse-tree node to match, it must match all the features given in the corresponding pattern node. If the pattern node has children, all the children of the parse-tree node must match the children of the pattern node.

Features with the same name are compared, and they match if they have the same values. But if the feature value in the pattern begins with an exclamation mark (for example, "!NP"), the match is negated: the feature value in the parse-tree must not match the remaining pattern string. Note that the exclamation mark can be preceded by a backslash to escape this interpretation (for example "\!NP").

Syntactic leaf nodes are indicated by a feature named "node:type" with value "leaf". A leaf node has either one child or more than one: each child represents a different morphological analysis for the word. Because the leaf children represent morphology alternatives, the matching algorithm works slightly differently: at least one (but not necessarily all) of the leaf-node children in the parse tree must match a leaf-node child in the pattern.

Figure 4.7 shows an example Find Pattern. The "D"' node has three children, but two of them are optional, as indicated by the parentheses around their node labels: "(OPT#1)" and "(OPT#2)". The "cat" feature on these optional nodes must not be "NP", since the pattern value begins with an exclamation mark: "!NP".

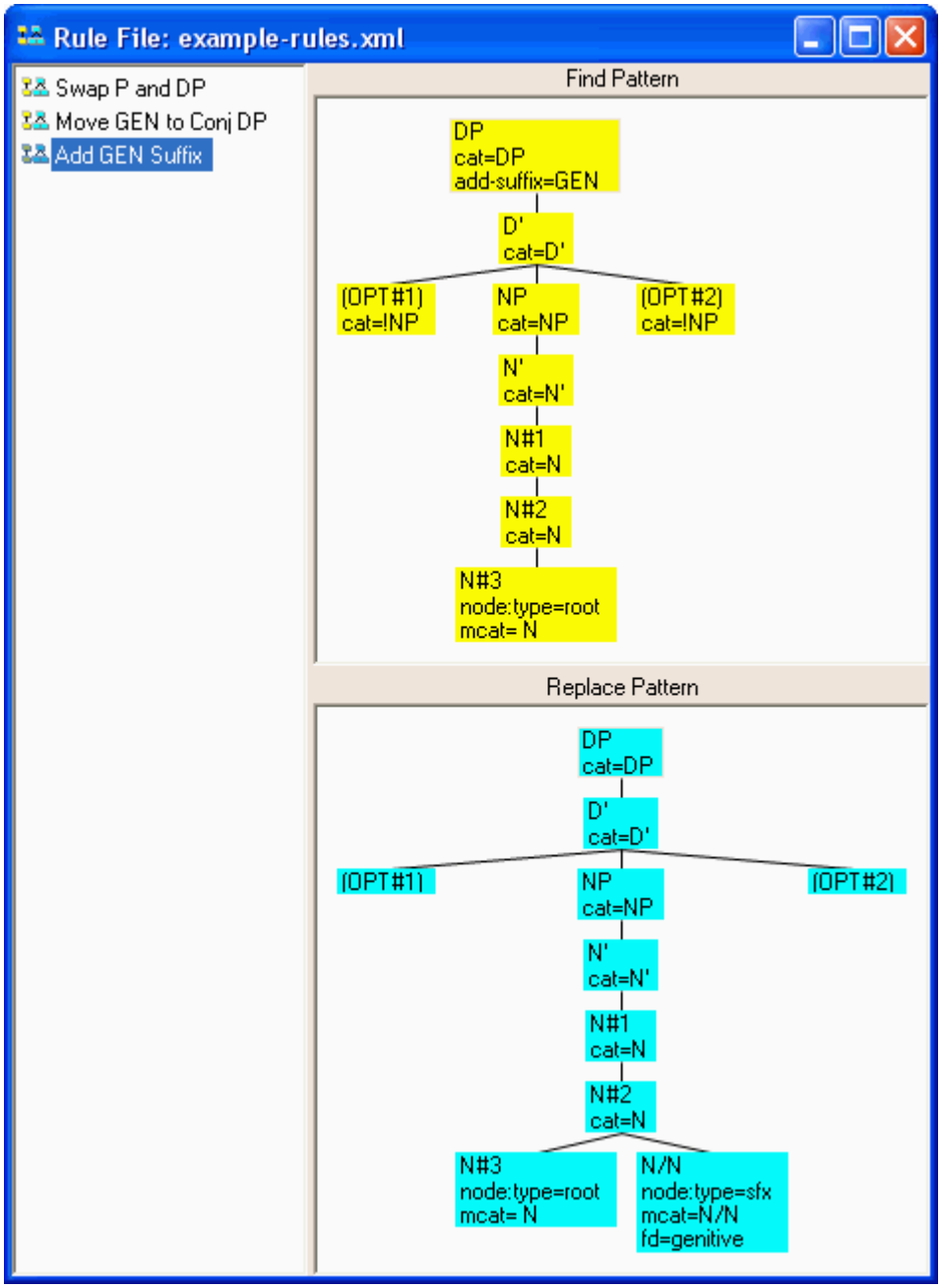

Figure 4.7 Example Find and Replace Patterns

4.1.8 Replace Pattern

After finding a matching sub-tree, the TreeTran engine replaces it with a new sub-tree specified by the Replace Pattern. The new sub-tree contains a copy of each node in the Replace Pattern. Optional nodes are only copied if they were matched by a corresponding node (one with the same label) in the Find Pattern.

When a Replace Pattern node is copied, it checks if a parse-tree node was matched by a corresponding node (same label) in the Find Pattern. If so, the features from the parse-tree node are copied to the new sub-tree node. Then the features from the Replace Pattern node are copied to the new sub-tree node, possibly replacing some of the feature values. Features with a blank value are deleted, and nodes with no features are deleted.

If a Replace Pattern node has children, they are copied to the new sub-tree node as its children. Otherwise, if a parse-tree node was matched by a corresponding node (same label) in the Find Pattern, the children of that parse-tree node are recursively copied as child branches of the new sub-tree node.

For syntactic leaf nodes, the child nodes (morphology alternatives) that were not matched by the Find Pattern are copied, in addition to any nodes specified by the Replace Pattern.

Figure 4.7 above also shows an example Replace Pattern. In the replaced sub-tree, a genitive suffix is added to the noun's morphology analysis.

4.2 Software Architecture

The TreeTran program is written in C# and is organized into three components: TreeTran (windows), TreeTranViewer (user interface) and TreeTranEngine (transfer algorithm).

4.2.1 TreeTran Component

The TreeTran namespace contains the following classes, which provide windows to contain the TreeTranViewer controls:

TreeTranMain. Implements the main entry point for the TreeTran.exe program.

TreeTranForm. Implements the main TreeTran form, an MDI-parent form that contains the RuleForm and ParseForm.

RuleForm. Implements an MDI-child form that displays a RuleListViewer, a FindPatternViewer and a ReplacePatternViewer.

ParseForm. Implements an MDI-child form that displays a ParseListViewer and a ParseTreeViewer.

HelpForm. Implements a form that displays user-help documentation.

4.2.2 TreeTranViewer Component

The TreeTranViewer component implements most of the user-interface functionality and makes the calls to the TreeTranEngine. These user-interface controls are separate from the TreeTran windows so the TreeTranViewer controls can be repackaged into other applications. The TreeTranViewer namespace contains the following classes:

RuleListViewer. Implements a control that allows the user to view and edit a list of transfer rules.

RuleListViewerItem. Represents a list item in the RuleListViewer control.

FindPatternViewer. Implements a control that allows the user to view and edit the find-pattern tree of a transfer rule.

ReplacePatternViewer. Implements a control that allows the user to view and edit the replace-pattern tree of a transfer rule.

ParseListViewer. Implements a control that allows the user to view and edit a list of parses.

ParseListViewerItem. Represents a list item in the ParseListViewer control.

ParseTreeViewer. Implements a control that allows the user to view and edit a parse tree.

FeaturesForm. Implements a form that allows the user to view and edit the list of features on a node in a parse tree or transfer rule.

TransferForm. Implements a form that reads parse trees from an input file, applies a set of transfer rules, and writes the modified parse trees to an output file.

TreeViewer. Implements a control that displays nodes in a tree. (The ParseTreeViewer, FindPatternViewer and ReplacePatternViewer are all derived from this control.)

TreeViewerNode. Represents a node in the TreeViewer control.

TreeViewerNodeCollection. Implements a collection of TreeViewerNode items.

TreeViewerNodeEventArgs. Represents event data for node-specific events raised by the TreeViewer control.

4.2.3 TreeTranEngine Component

The TreeTranEngine component implements all of the transfer algorithms to find matching sub-trees (using a backtracking stack for optional nodes) and to create replacement sub-trees. The TreeTranEngine also provides the classes to read and write rule and parse files. The TreeTranEngine namespace contains the following classes:

TreeTransfer. Represents the properties and methods used to apply a tree-transfer rule to a parse tree.

FindAlgorithm. Implements an algorithm to find a branch of the parse tree that matches the find-pattern tree.

MatchAlgorithm. Implements an algorithm to compare a branch of the parse tree against the find-pattern tree and produce a list of matching nodes.

ReplaceAlgorithm. Implements an algorithm to replace a branch of the parse tree, using the replace-pattern tree and the list of matching nodes.

ParseReader. Implements an XML reader for XAMPLE parse-tree data.

ParseWriter. Implements an XML writer for XAMPLE parse-tree data.

ParseXml. Defines XML string constants that are used by the ParseReader and ParseWriter classes.

RuleReader. Implements an XML reader for tree-transfer rule data.

RuleWriter. Implements an XML writer for tree-transfer rule data.

RuleXml. Defines XML string constants that are used by the RuleReader and RuleWriter classes.

SyntaxNode. Represents a node in a parse tree.

SyntaxNodeCollection. Implements a collection of SyntaxNode items.

SyntaxFeature. Represents a SyntaxNode feature as a name-value pair.

SyntaxFeatureCollection. Implements a collection of SyntaxFeature items.

SyntaxNodePair. Represents a pair of SyntaxNode objects: a node in the parse tree and the matching node from the find-pattern tree.

SyntaxNodePairStack. Implements a stack of SyntaxNodePair items.

SyntaxNodeTriple. Represents a triple of SyntaxNode objects: a new node in the replaced branch of the parse tree and the corresponding nodes from the find and replace patterns.

SyntaxNodeTripleStack. Implements a stack of SyntaxNodeTriple items.

TransferRule. Represents a transfer rule: the rule name, the root node of the find-pattern tree and the root node of the replace-pattern tree.

TransferRuleCollection. Implements a collection of TransferRule items.

TreeTranEngineString. Defines string constants that are used by the TreeTranEngine namespace.

4.2.4 Example Code

Figure 4.8 shows example code for reading a collection of transfer rules from a file named "rules.xml", using the RuleReader class.

```
// Read the list of transfer rules.

TransferRuleCollection transferRules
  = new TransferRuleCollection();
RuleReader ruleReader
  = new RuleReader(new StreamReader("rules.xml"));
while (ruleReader.Read())
{
  TransferRule rule = new TransferRule();
  rule.RuleName = ruleReader.RuleName;
  rule.FindPatternRoot = ruleReader.FindPatternRoot;
  rule.ReplacePatternRoot = ruleReader.ReplacePatternRoot;
  transferRules.Add(rule);
}
ruleReader.Close();
```

Figure 4.8 Example Code to Read Transfer Rules

Figure 4.9 shows example code for applying transfer rules. Parse trees are read from a file named "parses.xml", using the ParseReader class, each rule in the transfer-rules collection is applied, and the modified parse trees are written to a file named "output.xml", using the ParseWriter class.

A TreeTransfer object is used to apply transfer rules to a parse. First the TreeTransfer properties are set. Then the FindNextMatchingBranch() and ReplaceCurrentMatchingBranch() methods are called repeatedly, until FindNextMatchingBranch() return false.

```
// Open the input and output files.

ParseReader parseReader
  = new ParseReader(new StreamReader("parses.xml"));
ParseWriter parseWriter
  = new ParseWriter(new StreamWriter("output.xml"));

// Read and process each parse tree from the input file.

while (parseReader.Read())
{
  // Create a TreeTransfer object and set its ParseTreeRoot
  // to the parse tree that was read from the input file.

  TreeTransfer transfer = new TreeTransfer();
  transfer.ParseTreeRoot = parseReader.ParseTreeRoot;

  // Apply each rule in the transfer-rules collection.

  foreach (TransferRule rule in transferRules)
  {
    transfer.FindPatternRoot = rule.FindPatternRoot;
    transfer.ReplacePatternRoot = rule.ReplacePatternRoot;

    // Apply the rule to each matching branch.

    transfer.CurrentParseTreeNode = null;
    while (transfer.FindNextMatchingBranch())
    {
      transfer.ReplaceCurrentMatchingBranch();
    }
  }

  // Write the modified parse tree to the output file.

  parseWriter.ParseTreeRoot = transfer.ParseTreeRoot;
  parseWriter.Write();
}

// Close the input and output files.

parseReader.Close();
parseWriter.Close();
```

Figure 4.9 Example Code to Apply Transfer Rules

5 Experimental Methodology and Results

To evaluate the TreeTran system, a small-scale translation task was performed with three systems: (1) The first system used TreeTran to apply tree-transfer rules, (2) the second system used Perl to apply surface-transfer rules, and (3) the third system performed a simple word-for-word translation. The tree-transfer and surface-transfer systems each used the same number of rules. The results from these three systems were then compared by calculating the edit distance between each output sentence and a corrected target translation.

5.1 Translation with Tree-Transfer Rules

The source-language input was a set of XAMPLE parses for about two hundred Persian sentences from some short stories. The average sentence length was 6.8 words. Where the parser produced more than one parse for a sentence, the best parse was selected manually. The target language for this task was English.

Although TreeTran is intended for pairs of closely-related minority languages, its use is not limited to such pairs. Persian and English are neither closely-related nor minority languages, but the use of English as the target language made it more convenient for me to perform the evaluation.

After viewing the parses for the first fifty sentences, I selected four translation phenomena to address: (1) an SOV source language but an SVO target language, (2) subject pro-drop in the source language but not in the target language, (3) optional determiners in the source language that are required in the target language, and (4) determiners and adjectives after the noun in the source language that must appear before the noun in the target language.

TreeTran's interactive mode was used to create ten tree-transfer rules to address these phenomena. Figure 5.1 through Figure 5.4 show some examples of these rules.

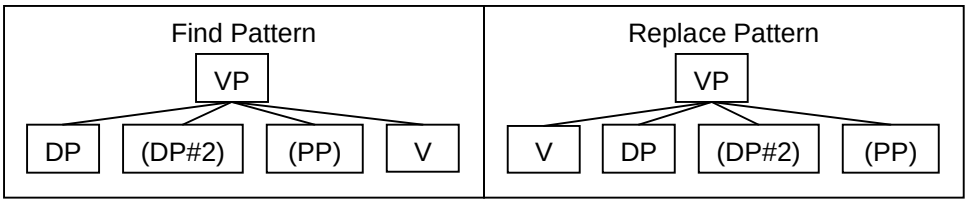

Figure 5.1 Example SOV-to-SVO Rule for VP Arguments

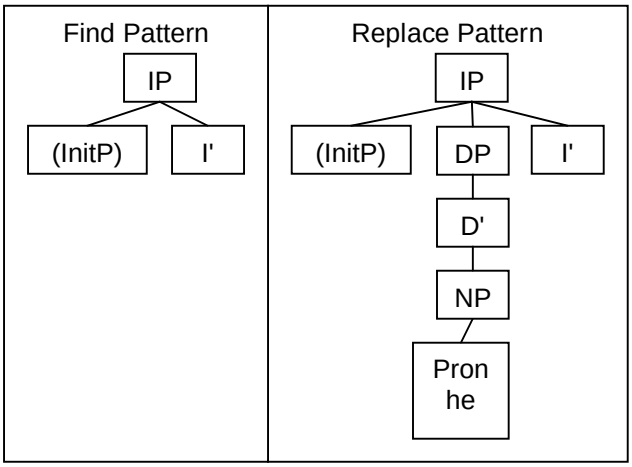

Figure 5.2 Example Pro-Drop Reversal Rule

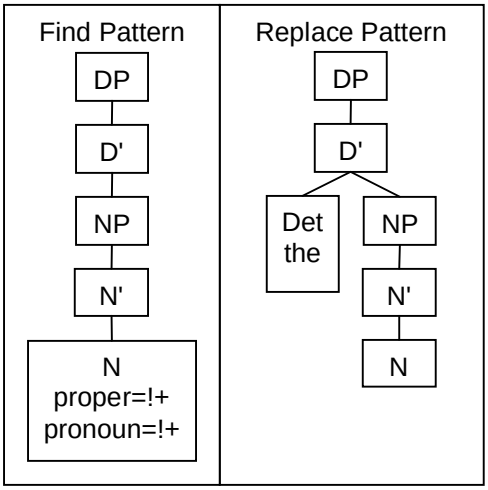

Figure 5.3 Example Determiner-Insertion Rule

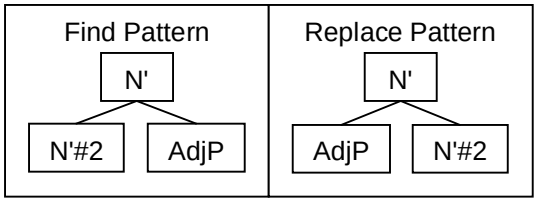

Figure 5.4 Example Noun-Modifier Ordering Rule

TreeTran's batch mode was then used to apply these ten tree-transfer rules to all of the input parses.

Figure 5.5 shows an example input parse tree (with lexical substitution), and Figure 5.6 shows the same parse tree after applying the ten tree-transfer rules.

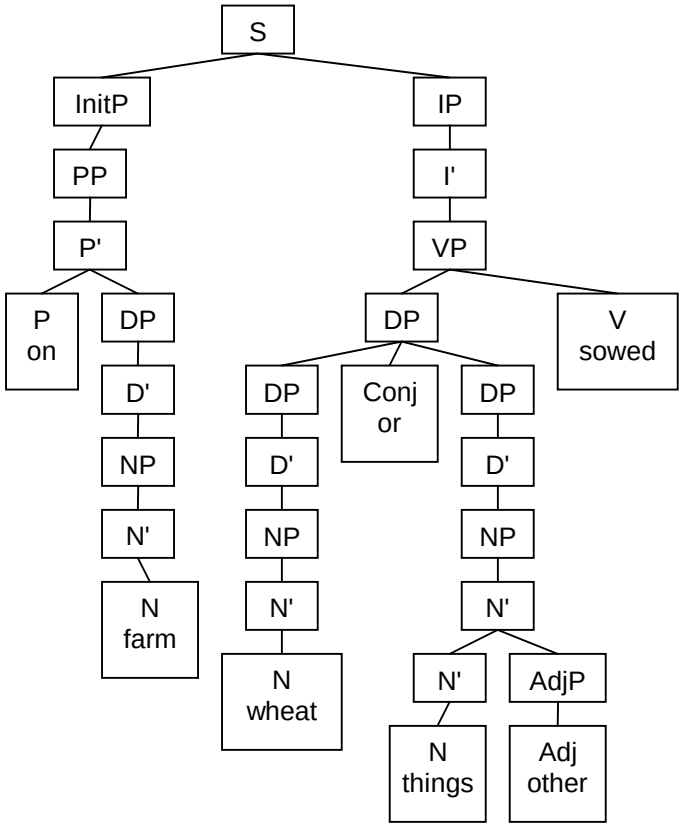

Figure 5.5 Example Input Parse Tree

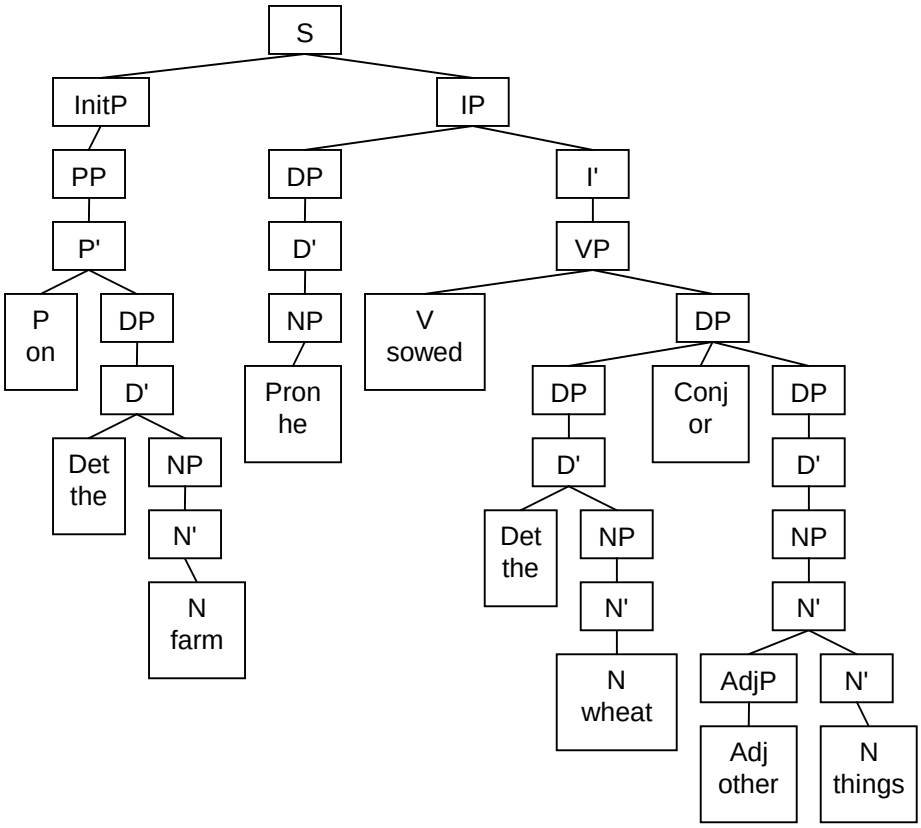

Figure 5.6 Example Output Parse Tree

After lexical substitution, the words associated with the ordered leaf nodes of output parses were used to produce the sentence translations. For the output parse tree in Figure 5.6, the translation output is:

On the farm he sowed the wheat or other things.

To evaluate the translation results, the set of output sentences was post-edited to produce a set of target corrected results. Each translated output sentence was compared to its corresponding target corrected sentence to calculate an edit-distance score, counting the minimum number of words that needed to be inserted or deleted in one sentence to produce the other sentence.

For the sentence translation given above, its corresponding target corrected sentence is:

On the farm he sowed wheat or other things.

So the edit-distance score is 1, since the two sentences differ by only a single word (the second "the" does not appear in the target corrected sentence).

5.2 Translation with Surface-Transfer Rules

The second translation system used surface-based transfer rules, rather than tree-based transfer rules. The words and part-of-speech tags from the input XAMPLE parse trees were used as the source-language input to the system. For example, given the parse tree in Figure 5.5, the following tagged words are used as the input:

on/P farm/N wheat/N or/Conj things/N other/Adj sowed/V

I used Perl regular expressions to write ten surface-transfer rules to address the same four translation phenomena as before. Figure 5.7 shows some examples of these rules.

```
### Pro-Drop Rule: ^ V ==> "he" V.
s/^ (\S*\/V) / he \1 /g;

### Noun-Mod Rule: N Adj ==> Adj N.
s/ (\S*\/N) (\S*\/Adj) / \2 \1 /g;

### Sov-Svo Rule: P Adj N V ==> V P Adj N.
s/ (\S*\/P) (\S*\/Adj) (\S*\/N) (\S*\/V) / \4 \1 \2 \3 /g;

### Add-Det Rule: V Adj N ==> V "the" Adj N.
s/ (\S*\/V) (\S*\/Adj) (\S*\/N) / \1 the \2 \3 /g;
```

Figure 5.7 Example Surface-Transfer Rules

After applying the surface-transfer rules and lexical substitution, the translation output produced from the tagged input given above is:

On farm wheat or other things sowed.

Since the target corrected sentence is:

On the farm he sowed wheat or other things.

the edit-distance score is 4 (the translation output needs to insert "the", insert "he", insert "sowed" in the middle, and delete "sowed" from the end).

Creating useful surface-transfer rules was much harder than creating tree-transfer rules. Many of the seemingly reasonable rules I tried actually made the edit distance worse, rather than better. In fact, using my first set of ten surface-transfer rules, the sentence translations had an average edit distance that was worse than the baseline word-for-word translation. So I modified the set of surface-transfer rules, making the rules include the part-of-speech context of more adjacent words, until every rule made more good changes than bad. (This is why some of the rules shown in Figure 5.7 have more specific contexts than one might expect intuitively.)

A second difficulty in creating surface-transfer rules had to do with the order in which rules were applied. For example, the rule that changed "N Adj" to "Adj N" needed to come before the rule that looked for a "P Adj N V" sequence, otherwise the second rule would fail to find a match.

Unlike tree-transfer rules, where entire sub-trees can be matched and moved without worrying about the order of constituents in their sub-structures, each surface-transfer rule must specify an entire sequence of words and tags, taking into consideration how words in that sequence may have been rearranged by any previous rule. As a result, more surface-transfer rules are needed to match different contexts, and the order and interaction of rules creates a more fragile system.

5.3 Word-for-Word Translation

The third translation system performed a simple baseline word-for-word lexical substitution on the words from the input XAMPLE parse trees. For example, given the parse tree in Figure 5.5, the translated output sentence is:

On farm wheat or things other sowed.

Since the target corrected sentence is:

On the farm he sowed wheat or other things.

the edit-distance score is 6 (the translation output needs to insert "the", insert "he", insert "sowed" in the middle, delete "sowed" from the end, insert "other" before "things", and delete "other" after "things").

5.4 Lexical Substitution

Before comparing the results of the three translation systems, a comment on how the target words were selected is in order. In the Analysis-Transfer-Synthesis model, the target-language parse-tree output of the Transfer component is used as the input to the Synthesis component, which generates the sentence translations. The Synthesis component selects the specific words and phrases to represent the information in the parse trees.

A complete Synthesis component is beyond the scope of this project, so a simple form of lexical substitution was used to compare the three translation systems in the evaluation task.

Each leaf node in the target parse trees contained the source-language word, its syntactic category, and a list of possible English translations for the word (from the lexicon used by the Analysis component). First I made the corrected target translations for all the sentences. Then I counted how often each set of leaf-node information (list of possible English translations and syntactic category)

translated to a specific word in the target corrected sentences, and the translation with the highest count was added to a table. This table was used to perform lexical substitution for all three translation systems.

In most cases, each leaf node translated unambiguously to a single word (within the context of the stories translated in this task). In a few cases, the translation with the highest count would be right for some sentences and wrong for others, but the same wrong word would appear in all three translations, so the results for the three systems could still be compared.

Table 5.1 shows part of the lexical-substitution table, illustrating the words in the target corrected sentence "and he said how much his heart hurt for him". One of the source-language words could be translated either "heart" (metaphorically) or "stomach", but in these stories it is always used in the metaphoric "heart" sense. The underspecified pronoun due to pro-drop ("he/she/it/they/Pron") is translated as "he", since this was most frequently the intended meaning.

Table 5.1 Excerpt from the Lexical-Substitution Table

Leaf-Node List of Possible English Translations And Syntactic Category	Most Frequent Target Word
and/Conj	and
he/she/it/they/Pron	he
to^say/tell-Past(3sg)/V	said
that/C	that
how^much/many/Adv	how_much
a/an/the/Det	the
heart(met.)/stomach-PPron(3sg)/N	heart
PROG-to^burn-Pres(3sg)/V	hurt
for/P	for
for^him/her/Pron	him

5.5 Comparison of Results

Figure 5.8 shows the average edit distance per sentence for each of the three systems evaluated in the translation task. These results are for the whole set of about 200 sentences.

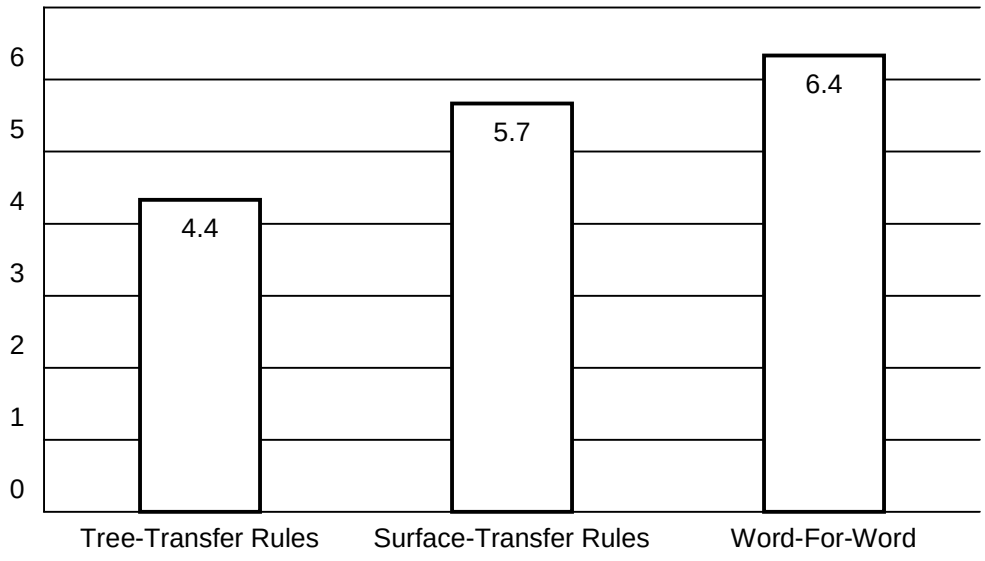

Figure 5.8 Average Edit Distance per Sentence

Table 5.2 gives the average edit distance per sentence, with separate results for the development-set sentences (used for rule development) and the test-set sentences (not used for rule development), as listed in the appendix.

Table 5.2 Average Edit Distance for Development and Test Sentences

	Development Set	Test Set
Tree-Transfer Rules	2.9	5.9
Surface-Transfer Rules	4.7	7.3
Word-For-Word	5.5	7.7

The translation results for the three systems were also compared by calculating the BLEU scores for each system. The BLEU evaluation uses a modified n-gram-precision score to measure the similarity between a machine translation and one or more good reference translations (Papineni, Roukos, Ward and Zhu, 2001). Table 5.3 gives the BLEU scores for each system (using the whole set of about 200 sentences) and also shows some of the individual n-gram scores.

Table 5.3 BLEU Scores

	BLEU	1-gram	2-gram	3-gram	4-gram
Tree-Transfer Rules	.3831	.7665	.4603	.2956	.2066
Surface-Transfer Rules	.1860	.8146	.3257	.1349	.0558
Word-for-Word Translation	.1216	.8211	.2491	.0856	.0241

5.6 Error Analysis

For the tree-transfer translations of each sentence listed in the appendix, I classified the errors into one of four categories: (1) Analysis Errors, where the input parse tree was incorrect, (2) Missing Transfer Rules, where the error could be corrected by adding a transfer rule, (3) Synthesis Errors, where the error could be corrected by better word choice or idiom handling, and (4) Other Errors, where there is not an obvious solution within this system. Table 5.4 shows the

proportion of errors in each category, for sentences from the development set (used for rule development) and from the test set (not used for rule development).

Table 5.4 Error Analysis

	Development Set	Test Set
Analysis Errors	12%	7%
Missing Transfer Rules	44%	35%
Synthesis Errors	28%	43%
Other Errors	16%	15%

Example of an Analysis Error: In the parse-tree input, the phrase that should be translated "phillip's father and mother" was represented by the incorrect structure shown below:

Incorrect:
```
(DP (DP father)
    and
    (DP (NP (N' mother) (DP phillip's))))
```

Should be: `(DP (NP (N' father and mother) (DP phillip's)))`

The resulting output was an incorrect translation ("the father and phillip's mother") that corresponded to the faulty input structure.

Example of a Missing Transfer Rule: The determiner-insertion rule, shown above in Figure 5.3, correctly adds a determiner for the first structure shown below, but fails for the other two structures, because they contain nested N' nodes:

```
(DP (D' (NP (N' (N horse)))))

(DP (D' (NP (N' (N' (N horse))
                (AdjP small)))))

(DP (D' (NP (N' (N' (N' (N horse))
                    (AdjP small))
                (AdjP old)))))
```

This can be fixed by adding rules to copy features (proper=+ and pronoun=+) from a child node to its N' parent, as shown in Figure 5.9. Since rules are applied

to each node in post-order (parent after children), the features will be copied up repeatedly through multiple levels of nested N' structures to the highest N' node. Then the determiner-insertion rule can be modified to check for those features on the N' node, as shown in Figure 5.10 (so the rule no longer depends on having a single N' node between the NP and N nodes).

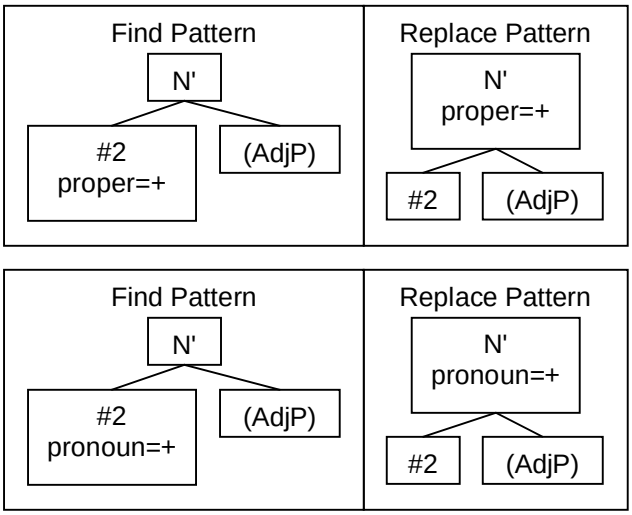

Figure 5.9 Rules to Copy Features up to the Parent N' Node

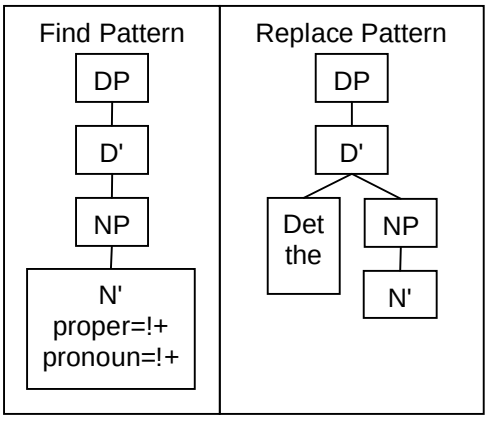

Figure 5.10 Modified Determiner-Insertion Rule

Example Synthesis Errors: The literal translation "I was angry from her hand" should be translated "I was angry at her." An improved Synthesis component

might handle idiomatic constructions like this, in addition to literal word translations. Other errors could be fixed if the Synthesis component looked at some amount of discourse context. For example, the pro-dropped subject could be translated "she" (rather than "he/it/they") if the subject of the previous sentence was singular and female.

Other Errors: In some sentences there are constructions and ambiguities that cannot be easily handled. For example, the literal translation "no animal did not come" should be interpreted as "no animal came," but not all double negatives should be handled this way.

6 Discussion

In the small-scale translation task described in the previous chapter, TreeTran demonstrated improvement over two baseline systems. (See Figure 5.8 above.) Compared to a word-for-word translation, a set of ten TreeTran rules improved the edit-distance score by over 30%. Compared to a set of ten surface-transfer rules, the ten TreeTran rules improved the edit-distance score by over 20%.

Useful TreeTran rules were much easier to identify than surface-transfer rules. The surface-transfer rules had trickier issues with picking enough context and ordering the rules, which made the surface-transfer system more fragile when changing or adding to the set of rules. There were also phenomena, like recursively embedded constituents, that the surface-transfer rules could not handle. With TreeTran rules, entire sub-tree structures could be matched and moved, regardless of the number or the order of constituents they contained. With surface-transfer rules, however, each of the potentially infinite number of word orders in the structure would require another rule.

TreeTran's user interface also offers advantages in terms of usability and efficiency when creating and debugging a set of tree-transfer rules. The TreeTran rules for the translation task described above were created in a matter of minutes, by copying parse-tree branches and visually editing them to create find and replace patterns. The set of rules was then tested by stepping through each rule to visually highlight the changes it made.

7 Conclusion and Future Work

This paper has described a computer program, called TreeTran, that aids a human translator in visually selecting, editing and debugging tree-transfer rules as part of an Analysis-Transfer-Synthesis system for machine translation.

In a small-scale experiment, the system demonstrates improvement over two baseline systems. Furthermore, the system offers a number of user-interface features for making the rule-creation task easier and less error-prone.

Although TreeTran is designed to be part of a minority-language translation system where human translators manually create transfer rules and perform post-editing on the output, there may be opportunities for future application of statistical methods. As a growing amount of post-edited output is produced in a language, semi-supervised learning methods might be applied to the tasks of testing and revising transfer rules, or improving the order of rules, for example.

Future user-interface enhancements might expand the support for undo and drag-and-drop. Future engine enhancements might include additional ways to match and manipulate features. For example, a find-pattern node might specify a list of feature values, like "DP|PP|AdjP" to indicate that the matching parse-node feature value must be "DP", "PP" or "AdjP". Instead of matching a given string, pattern nodes might also specify that a feature value should match or be copied from the feature value of some other indicated node. Other desirable enhancements may become apparent as the system is put into use on larger translation tasks.

REFERENCES

Bergman, William. 1996. "An Implementation of Transfer Based on Tree Editing." In H. Andrew Black, Alan Buseman, David Payne and Gary Simons, editors, *Proceedings of the 1996 general CARLA conference*, November 14-15, 1996. Waxhaw, NC/Dallas: JAARS and Summer Institute of Linguistics.

Brown, Peter, Vincent Della Pietra, Stephen Della Pietra and Robert Mercer. 1993. "The Mathematics of Statistical Machine Translation: Parameter Estimation." *Computational Linguistics, Volume 19, Number 2.*

Brown, Ralf. 1997. "Automated Dictionary Extraction for 'Knowledge-Free' Example-Based Translation." *Proceedings of the 7th International Conference on Theoretical and Methodological Issues in Machine Translation (TMI-97).*

Kaji, Hiroyuki, Yuuko Kida and Yasutsugu Morimoto. 1992. "Learning Translation Templates From Bilingual Text." *Proceedings. of the 14th International Conference on Computational Linguistics (COLING 1992).*

Knight, Kevin. 1997. "Automating Knowledge Acquisition for Machine Translation." *AI Magazine 18(4).*

Knight, Kevin. 1999. "A Statistical MT tutorial Workbook." JHU CLSP Summer Workshop. (Available on the internet at http://www.isi.edu/natural-language/mt/wkbk.rtf.)

Lavoie, Benoit, Michael White and Tanya Korelsky. 2002. "Learning Domain-Specific Transfer Rules: An Experiment with Korean to English Translation." In *Proceedings of the COLING 2002 Workshop on Machine Translation in Asia*. Taipei, Taiwan.

Lin, Dekang. 2004. "A path-based transfer model for MT." *Proceedings of the 20th International Conference on Computational Linguistics (COLING 2004).*

Manning, Christopher and Hinrich Schutze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press.

McConnel, Stephen, H. Andrew Black and Marius Doornenbal. 2006. "AMPLE Reference Manual: A Morphological Parser for Linguistic Exploration." SIL International. (The AMPLE software and documentation are available at http://www.sil.org/computing/catalog.)

McConnel, Stephen. 1995. "PC-PATR Reference Manual: A Unification Based Syntactic Parser." Academic Computing. Dallas, TX. (The PC-PATR software and documentation are available at http://www.sil.org/pcpatr. Also, see http://www.ai.mit.edu/courses/6.863/doc/pcpatr.html for an updated 2000 version.)

Menezes, Arul and Stephen Richardson. 2001. "A Best-first Alignment Algorithm for Automatic Extraction of Transfer Mappings from Bilingual Corpora." *Proceedings of the Workshop on Data-driven Machine Translation at the 39th Annual Meeting of the Association for Computational Linguistics, (ACL-01)*.

Meyers, Adam, Michiko Kosaka and Ralph Grishman. 2000. "Chart-based transfer rule application in machine translation." *Proceedings. of the 18th International Conference on Computational Linguistics (COLING 2000)*.

Nagao, Makao. 1984. "A Framework of a Mechanical Translation between Japanese and English by Analogy Principle." In Alick Elithorn and Ranan Banerji, editors, *Artificial and Human Intelligence*. Elsevier Science Publishers B.V. Amsterdam, The Netherlands.

Och, Franz Josef and Hermann Ney. 2002. "Discriminative Training and Maximum Entropy Models for Statistical Machine Translation." *Proceedings of the 40th Annual Conference of the Association for Computational Linguistics (ACL-2002)*. Philadelphia, PA.

Papineni, Kishore, Salim Roukos, Todd Ward and Wei-Jing Zhu. 2001. "BLEU: a Method for Automatic Evaluation of Machine Translation." Research Report, IBM Computer Science Research Division, T. J. Watson Research Center.

Probst, Katharina. 2005. "Learning Transfer Rules for Machine Translation with Limited Data." Unpublished PhD dissertation, Carnegie Mellon University.

Slocum, Jonathan. 1985. "A Survey of Machine Translation: Its History, Current Status, and Future Prospects." *Computational Linguistics 11(1)*.

Ward, Nigel and Daniel Jurafsky. 2000. "Machine Translation." Chapter 21 of *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition* by Daniel Jurafsky and James H. Martin, Prentice-Hall.

Appendix A: Example Sentence Translations

Listed below are translations of 40 input sentences (with four or more words) that were used in the evaluation task. For each sentence there are four translations: the Corrected Target Translation (CTT), the Word-For-Word translation (WFW), the translation using Surface-Transfer Rules (STR), and the translation using Tree-Transfer Rules (TTR). The edit-distance scores appear in square brackets after the sentences.

First 20 sentence translations from the set used for rule development:

CTT: **there was an old man that had three sons**
WFW: old man was that three sons had [6]
STR: the old man was that three sons had [7]
TTR: he was old man that had three sons [3]

CTT: **the small son's name was sasha**
WFW: name son small sasha was [7]
STR: name small son sasha was [7]
TTR: small son name was sasha [3]

CTT: **this old man had a farm**
WFW: this old man a farm had [2]
STR: this old man a farm had [2]
TTR: this old man had a farm [0]

CTT: **on the farm he sowed wheat rice or other things**
WFW: on farm wheat rice or things other sowed [6]
STR: on farm wheat rice or other things sowed [4]
TTR: on the farm he sowed the wheat the rice or other things [2]

CTT: **but every night an animal came**
WFW: but every night animal came [1]
STR: but every night came animal [3]
TTR: but every the animal night came [4]

CTT: **he ate the crops and went**
WFW: crops ate and went [4]
STR: crops ate and went [4]
TTR: he ate the crops and went [0]

CTT: **one day the old man said to the sons that one must go**
WFW: a day old man to sons said that must go [7]
STR: a old day man said to sons that must go [7]
TTR: a day old man said to the sons that must he go [5]

CTT: **because this animal came**
WFW: because this animal came [0]
STR: because this animal came [0]
TTR: because this animal came [0]

CTT: **he became tired and slept**
WFW: sleep caught and slept [5]
STR: sleep caught and slept [5]
TTR: he caught the sleep and slept [5]

CTT: **that animal came he destroyed the farm he ate the crops and went**
WFW: that animal came farm destroyed crops ate and went [8]
STR: that animal came the destroyed farm crops ate and went [7]
TTR: that animal came he destroyed the farm he ate the crops and went [0]

CTT: **the second night the middle son went also**
WFW: night second son middle also went [8]
STR: the second night son middle went also [3]
TTR: second night the middle son went also [1]

CTT: **he became tired and slept**
WFW: tired became and slept [3]
STR: tired became and slept [3]
TTR: he became tired and slept [0]

CTT: **the animal came he ate the wheat and went**
WFW: animal came wheat ate and went [5]
STR: animal came wheat ate and went [5]
TTR: the animal came he ate the wheat and went [0]

CTT: **the third night the small son that was named sasha made a decision to go**
WFW: night third son small that name sasha was decision caught to go [13]
STR: the third night son small that name was sasha decision caught to go [8]
TTR: third night small son that the name was sasha caught decision he to go [9]

CTT: **he poured salt on a wound to burn and prevent sleep**
WFW: top wound salt poured that to burn and sleep prevent [11]
STR: top wound poured salt that to burn and sleep prevent [9]
TTR: he poured the wound top the salt that to burn and prevent the sleep [9]

CTT: **he went and sat a long time**
WFW: went and very sat [5]
STR: he went and sat very [4]
TTR: he went and sat very [4]

CTT: **but the animal did not come**
WFW: but no animal did not come [2]
STR: but no animal did not come [2]
TTR: but no animal did not come [2]

CTT: **because his hand burned very much it prevented sleep**
WFW: leading to because hand very burned sleep prevented [9]
STR: leading to because hand burned very sleep prevented [7]
TTR: leading to because the hand burned very he prevented the sleep [8]

CTT: **he stood in a corner**
WFW: a corner stood became [5]
STR: a corner became stood [5]
TTR: he became stood a corner [2]

CTT: **in exchange he gave the promise**
WFW: in exchange promise gave [4]
STR: in exchange gave promise [2]
TTR: in the exchange he gave the promise [1]

Last 20 sentence translations from the test set (not used for rule development):

CTT: **all the people gathered in front of the cage and watched**
WFW: people all front cage gathered happened and watched [9]
STR: people all front cage happened gathered and watched [9]
TTR: people all happened the gathered cage front and the watched [9]

CTT: **they were surprised and did not know what to say**
WFW: they surprised were and did not know what must to say [3]
STR: they surprised were and did not know what must to say [3]
TTR: they were the surprised and did not know what must he to say [3]

CTT: **until the bird started to move and then he cheeped with a loud voice**
WFW: until that bird started full move gave and then with voice loud started to cheep [13]
STR: until that bird started full gave move and then with loud voice started to cheep [11]
TTR: until that the bird gave move the full started and then he the to the cheep with the started loud voice [15]

CTT: **she picked up the cage and ended this laughter**
WFW: cage picked up and to this laughter ended gave [8]
STR: cage picked up and to this laughter gave ended [8]
TTR: he picked up the cage and gave the ended to this laughter [5]

CTT: **and she went inside the house**
WFW: and to inside house went [5]
STR: and went to inside house [3]
TTR: and he went to inside house [4]

CTT: **all the time she said to herself**
WFW: in time that with herself said [9]
STR: in time that with herself said [9]
TTR: in the time that said with herself [6]

CTT: **that was really strange**
WFW: really that strange is [4]
STR: really that strange is [4]
TTR: really that strange is [4]

CTT: **one day early in the morning ruzali caught him by surprise**
WFW: a day morning early ruzali him caught by surprise [8]
STR: a day early morning ruzali him caught by surprise [6]
TTR: a day early morning ruzali caught by surprise him [6]

CTT: **phillip's father and mother questioned him**
WFW: father and mother phillip from he questioned [5]
STR: father and mother phillip from he questioned [5]
TTR: the father and phillip mother questioned from he [6]

CTT: **phillip described the little old horse's condition**
WFW: phillip condition horse little old described [7]
STR: phillip condition little horse old described [7]
TTR: phillip described old little horse condition [5]

CTT: **and he said how much his heart hurt for him**
WFW: and said that how much heart for him hurt [5]
STR: and said that how much heart for him hurt [5]
TTR: and he said that how much the heart hurt for him [3]

CTT: **and she did not give food to him**
WFW: and to him food did not give [7]
STR: and to him food did not give [7]
TTR: and he did not give the food to him [3]

CTT: **for this reason i was angry at her**
WFW: to this same reason i from hand his angry was [10]
STR: to this same reason i from hand his angry was [10]
TTR: to this same reason i angry was from his hand [10]

CTT: **sincerely ruzali confessed to her mistakes**
WFW: ruzali sincerely to mistakes confessed [5]
STR: the sincerely ruzali to mistakes confessed [4]
TTR: sincerely ruzali the confessed to the mistakes [3]

CTT: **and she said that phillip was justified and said the truth**
WFW: and said that phillip truth had and justified said [8]
STR: and said that phillip had truth and justified said [8]
TTR: and he said that phillip had the truth and said justified [10]

CTT: **she was not kind to the small horse at all**
WFW: he at all with horse small kind was not was [16]
STR: he at all with small horse kind was not was [16]
TTR: he kind was not with small horse at all was [8]

CTT: **they reconciled and phillip cleaned all of ruzali's windows for compensation**
WFW: they reconciled and phillip for compensation past whole windows house ruzali clean [11]
STR: they reconciled and phillip for past compensation whole windows house clean ruzali [11]
TTR: they reconciled and phillip clean whole ruzali house windows for past compensation [9]

CTT: **gradually the small horse's condition became better**
WFW: gradually condition horse small better became [7]
STR: gradually condition small horse better became [7]
TTR: gradually small horse condition became better [3]

CTT: **and with ruzali's nursing he was able to take the road to the market**
WFW: and with nursing ruzali's he was able to road market in track take [9]
STR: and with nursing ruzali's he was able to road take market in track [9]
TTR: and with ruzali's nursing he was able to he take the market road in the track [6]

CTT: **of course the load was two small baskets**
WFW: of course load two baskets small was [5]
STR: of course load two small baskets was [3]
TTR: of course the load was two small baskets [0]