Sentence Transfer Program Reference Manual
Alan Buseman
21 Nov 91 (1.0a)
01 Jul 2002 (2.1.0) (by H. Andrew Black)
24 Nov 2004 (2.2.0) (by Marius Doornenbal & H. Andrew Black)

Table of Contents

1. Introduction

The Sentence Transfer program is a part of the CADA (Computer Assisted Dialect Adaptation) system. It is designed to be used between the AMPLE and STAMP programs. It is useful both for removing ambiguities from a source analysis, and for adjusting a source analysis to reflect a target order.

Sentence Transfer is normally run two different times between AMPLE and STAMP. The first time it uses a set of disambiguation rules to remove ambiguities from the source analysis. This results in an improved analysis. But the analysis is still purely an analysis of the source language, not of the target.

The second time Sentence Transfer is run, is uses a set of transfer rules to modify the source analysis into a target analysis. This can include adjusting morphemes based on context, splitting affixes off into separate words, joining words together, and rearranging words and phrases.

2. Rule File

The rule file is made with an editing program. It contains the rules for disambiguating and rearranging the analysis. The format and usage of the rule file is described below.

3. File Identification Line

The rule file can have an `\id` line at the top. It is ignored.

4. Category field

The rule file should contain near the top a field listing all of the categories used in the analysis. This field looks just like the category field in the AMPLE Analysis Data Control File. For example:

```
\ca N V Pron Adj Adv
```

5. Disambiguation Rules

Rules that start with the marker `\am` (for ambiguity) are disambiguating rules. The program removes ambiguities by matching the pattern given in the rule, and then removing all ambiguities that are not part of that pattern.
For example, the following rules could be used to remove some of the ambiguities in the Frisian examples in the CADA tutorial:

```
\am V -INF / V _ | INF after verb
\am V -PRES_1S/2SPO/123PL | Tense elsewhere
```

The infinitive marker and one of the tense suffixes are ambiguous in Frisian. The first rule above says that whenever a verb with that suffix comes after another verb, then the suffix means infinitive. The second rule says that all other instances of that suffix mean present tense.

The program knows that "V" is a category because it is listed in the `\ca` field earlier in the rule file. It knows that -INF is a suffix because it has a hyphen in front of it. The vertical bars start comments, just as they do in AMPLE control files.

Here are two more rules that remove ambiguities from Frisian:

```
\am they / _ V -PRES_1S/2SPO/123PL | they before plural
\am she / _ V -PRES_3S | she before singular
```

The first rule says that "she/they" is "they" before a verb with a plural suffix. The second says the "she/they" is "she" before a verb with a singular suffix.

One useful technique for disambiguation is to end a series of rules with a rule that forces all remaining ambiguities to the most common case. For example, we might decide that "they" is much more common than "she" in the documents we are adapting.
Then we could put after the above rules the following rule:

```
\am they | all others become they
```

This forces all "she/they" ambiguities not removed by other rules to become "they". This technique is especially useful for cases where a common word is ambiguous with an unusual one.

6. Rearrangement Rules

Rules starting with the marker `\ru` (for rule), are rearrangement rules. A rearrangement rule has a pattern just like an ambiguity rule, followed by a right wedge, and a replacement pattern. For example, the following rule moves a word glossed "usually" from after a verb to before it.

```
\ru V usually > usually V
```

Notice that the morphname "usually" actually stands for the root of the word, and matches any word with that root, including any affixes it may have. In the same way, the category "V" stands for an entire verb including all its affixes. So all the affixes of a word automatically move with the word.

If a particular affix is named, then the category or root morphname of the word stands for the root and all the affixes not named. This can be used to move an affix from one word to another, or to split a morpheme off a word as discussed below.

As far as possible, affixes not mentioned in a rule are kept. If one word is substituted for another, all affixes not mentioned in the match are carried across to the replacement word. For example, the following rule:

```
\ru kitten > cat
```

Applied to the word "kittens" produces "cats".

If there are multiple words in the match, the following principle is used to decide what to do with affixes. If the first word of the match does not appear in the replacement, then any prefixes on it (that are not mentioned in the match) are put on the first word of the replacement. In the same way, if the last word of the match does not appear in the replacement, then any suffixes on it are put on the last word of the replacement.

For example, consider the following rule:

```
\ru little kitten > small cat
```

This rule copies any prefixes on the word "little" to "small", and any suffixes on the word "kitten" to "cat".

As another example, consider the following rule:

```
\ru chicken little > small chicken
```

Any affixes on "chicken" are carried with "chicken". Any prefixes on "little" are lost. Any suffixes on "little" are placed on "chicken", because "chicken" is the last word of the replacement. This may not be what is wanted. To copy the affixes from "little" onto "small", use a two-step process, as follows:

```
\ru little > small / chicken _
\ru chicken small > small chicken
```

To remove affixes which should not be kept, mention them as optional in the match, but not in the replacement.

As another example, the following rule matches a sequence of number, classifier, and noun, and changes it to noun, number, classifier.

```
\ru Num Class N > N Num Class
```

Rearrangement rules can also disambiguate. For example, if a noun matched by the rule above is ambiguous with a verb, the verb meaning is eliminated before the rearrangement is done. However, I recommend that Sentence Transfer be run twice during the adaptation process, using two different rule files. The first run is to remove as many ambiguities as possible from the analysis produced by AMPLE. This process has nothing to do with the target language. It simply produces a cleaner version of the analysis. The second run is to rearrange and adjust words and affixes as necessary for transfer to the target analysis.

7. Sentence Boundary

The number sign (#) can be used to specify the sentence boundary. For example, the following rule removes the Frisian ambiguity between "lake" and "but" at the beginning of a sentence.

```
\am but / # _ | Not lake at sentence start
```

This does not eliminate all of the "lake/but" ambiguities. It never removes the meaning "but", and it removes the meaning "lake" only at the beginning of a sentence. To remove them all would take many more rules, and probably cannot be done completely accurately with this program.

Notice that even this rule is likely to choose wrong on certain rare occasions, because it is perfectly possible for a sentence to begin with the word "lake". But I think that at least 99 percent of the time this rule will be right. And when this rule is wrong, the result will probably look strange enough that it will be obvious and easy to fix.

A good question to ask yourself about any disambiguation is whether the wrong choice will look odd enough that it is unlikely to mislead the person who does the initial revising. Many times the wrong choice will look so odd that it will be easy to spot and fix the small percentage of times it occurs.

8. Optional Elements

Parentheses around an element say it is optional. For example, the following rule says that a noun can optionally occur between the two verbs.

```
\am V -INF / V (N) _ | INF after verb
```

A single set of parentheses cannot be placed around multiple elements, only around a single element.

9. Affixes

Affixes are signalled by hyphens, as shown in the first example above. If the hyphen is on the left, it means suffix, if on the right, it means prefix. Infixes are pulled out by AMPLE so that they look like prefixes during sentence transfer.

When referring to an affix, one should normally mention the root.
A suffix is assumed to be part of the word to its left, and a prefix is assumed to be part of the word to its right. For example, in the following rule, "-PRES_3S" is assumed to be attached to the verb.

```
\am she / _ V -PRES_3S | She before singular
```

The following form of the rule will not work because the suffix will be assumed to attach to "she", not to the following verb.

```
\am she / _ -PRES_3S | WRONG!
```

10.Negated Elements

A tilde before an element says that it must not be present. For example, the following says that "she/they" is "she" before any verb that does not have a plural suffix.

```
\am she / _ V ~-PRES_1S/2SPO/123PL | she before non-plural
```

The tilde should not be used on optional elements. Tilde is also no good on an element in an environment with ellipses. To negate an environment with ellipses, put the tilde with the underline.

11.Ellipses

Ellipses can be used to stand for any string of elements. For example, the following rule says that "she/they" is "she" if a verb with a present singular suffix is found anywhere in the sentence after it.

```
\am she / _ ... V -PRES_3S | she anywhere before singular
```

With no limit on ellipes, this is a rather far-reaching rule, of course, and may find a singular verb in some other clause later in the sentence. To prevent such problems there is a limit to the distance ellipses can reach. The default limit is 5 words. The limit can be adjusted with a line like the following:

```
\... 6
```

This tells the program to let ellipses go as far as 6 words. The limit applies to all rules until another limit is given. For example, the following limits the ellipses in the first two rules to 3, and those in the third rule to 6:

```
\... 3
\am x / _ ... y
\am x / _ ... z

\... 6
\am x / _ ... w
```

To allow unlimited ellipses, give a large limit, such as 100.

12.Classes

Classes are somewhat like morpheme classes in AMPLE, except that they can contain both categories and morphemes, as well as the names of other classes. A class is defined by the marker `\cl` followed by the name of the class, followed by a list of the members of the class.

Classes are very useful for capturing generalizations. For example, the set of classes and rule below rearrange the Frisian analysis of "He has never there back been." into "He has never been back there." But the rule does far more than just that sentence. It uses classes and categories and optionality to rearrange a large number of related sentence patterns into English order.

```
 \cl Aux has have will must
 \cl Loc here there
 \cl Neg never not

\ru (Loc) (Neg) (back) V >
 (Neg) V (back) (Loc) / Aux _
```

I give classes names with mixed upper and lower case to distinguish them from roots, which are normally all lower case, and affixes, which are normally all upper case. This makes the rules easier to understand. But class names can be any case, and can contain non-alphabetic symbols, except for those with special meanings in rules, such as parends, wedges, hyphens, and equal sign.

The example above also illustrates that rules do not have to fit on one line. A rule continues until the next backslash marker.
Class definitions also can continue for as many lines as needed.

13.Classes of Affixes

To use a class of affixes, list the affixes without hyphens in the class, and use the class name with a hyphen in the rule. For example the following adds a 3RD person affix to any verb that has no person:

```
\cl Pers 1ST 2ND 3RD
\ru V > V -3RD / ~_ -Pers
```

Note that since the "-Pers" is a suffix immediately after the underline in the environment, it is assumed to be on the "V" in the match. To refer to a suffix on the following word, the category of the following word should be included. For example:

```
\am she / _ V -PAST_13S
```

The replacement of a rule can mention a class of affixes if the rule refers to the same class in the match. For example, the following class and rule are acceptable:

```
\cl Nsuff PLUR POSS
\ru N Num (-Nsuff) > Num N (-Nsuff)
```

Note that such a construction is only necessary for the unusual case of moving affixes from one word to another during rearrangement. If the affixes are not mentioned, they move with the word they are on. If the rule is a substitution instead of a rearrangement, affixes not mentioned are moved to the substitute word as described above.

It is not possible to refer to two different affixes by using the same class name twice. If the same class name is used twice, they will both match the same affix. To refer to two different affixes by class, make two different classes containing different sets of affixes.

14.Multiple Environments

Multiple environments are or'ed, so that if any one of them is true, the whole is true. For example, the following says that the ambiguity "she" should be chosen in any of three environments:

```
\am she / _ V -PAST_13S | she before singular
 / _ V ~-PRES_1S/2SPO/123PL | or before non-plural
 / _ ... V -PRES_3S | or anywhere before sing
```

15.Negated Environments

An environment is negated by placing a tilde just before the underline in the environment. For example:

```
\ru (Loc) (Neg) (back) be >
 be (Neg) (back) (Loc) / Aux ~_
```

This rule covers cases like "he there never back was", which becomes "he was never back there". If we had left "Aux" off the previous rule, it would have turned "he there never back was" into "he never was back there", which doesn't seem as good.
(Note that I am assuming an analysis in which "was" is analyzed as "be-PAST", so that "be" refers to the root of any "be" verb.)

Negation works correctly at the beginning and end of a sentence.
For example, the above rule works if the "Loc" occurs as the first word of a sentence.

16.Negated Elements

A negated element can be used in an environment. For example, the following does the same as the previous rule:

```
\ru (Loc) (Neg) (back) be >
 be (Neg) (back) (Loc) / ~Aux _
```

However, unless there is some good reason to negate an element, I recommend negating the entire environment instead.

17.Rules Can Feed Each Other

Rules are executed from the top of the list down, so the output of one rule can produce the elements needed for a later rule to apply. This allows approaches where one rule handles a general case and another rule corrects exceptions to the general case.
For example, the above rules which use "Aux" and "~Aux" as environments could also be done using the feeding rules below.

```
\ru (Loc) (Neg) (back) V >
 (Neg) V (back) (Loc)

\ru Neg be > be Neg / ~Aux _ (back) Loc
```

18.Deletion

If nothing (other than a possible environment) is given after the wedge, deletion is performed. For example, the following rule changes two occurences of the word "receive" into one. (This is from a Southeast Asian transfer pair.)

```
\ru receive > / receive _
```

19.Insertion

Any root or affix word on the right side of a rule that is not mentioned on the left side of the rule is inserted.

If an affix is inserted, it is joined to the nearest root.

When a root is inserted, it must be given a category. This is done by giving the category before the morphname of the root, with an equal sign between. For example, the following inserts
between any noun and pronoun the word "belong", with a category of "Prep". (This is from a Southeast Asian transfer pair.)

```
\ru N > N Prep=belong / _ Pron
```

This capability can also be used to change the category of a word, if that is desired.

20.Splitting Affixes off Words

An affix can be split off a word by giving the affix on the left of a rule and the word it should become with a category on the right of the rule. For example, the following rule splits the possessive suffix "POSS" off a word of category N and changes it into the into the word "possessive", with a category of "Part" (for particle). (Remember that these are morph names, so that "possessive" will be changed into a real word in the target language.)

```
\ru N -POSS > N Part=possessive
```

If an affix to be split could be found on more than one category, use a class of categories. For example, the following splits "-POSS" off a variety of categories:

```
\cl W N V Adj Adv Pron Det Prep
\ru W -POSS > W Part=possessive
```

21.Joining Words on as Affixes

Words can be changed into affixes using the reverse of the split rule above. For example, the following changes the word "possessive" into the suffix "POSS" on the preceding word.

```
\ru N possessive > N -POSS
```

The above rule can also be written as follows:

```
\ru possessive > -POSS
```

22.Compound Roots

To match a compound root, put the parts together with a double hyphen. For example:

```
\am black--bird
```

This is different from affixes in that prefixes have space after the hyphen, and suffixes have space before the hyphen. Double hyphen is used for compound roots so that a single embedded hyphen can be part of a morphname.

Compounds can consist of more than two roots. Any number of compound roots can be strung together with double hyphens.

To remove a root from a compound, do as follows:

```
\ru black--bird > black N=bird
```

Affixes stay with the first root of the compound. So if there were affixes on "black--bird", they will be placed on "black".
If affixes should be placed on another root, they should be mentioned as options. For example, the following rule puts plural on "bird":

```
\ru black--bird (-PL) > black N=bird (-PL)
```

To build a compound root, do as follows:

```
\ru black bird > black--N=bird
```

The "N=" gives the root category of "bird" in the compound.

Affixes come from the first root of the compound. So if there were affixes on "black", they will be placed on "black—bird".
If there were prefixes on "bird", they will be lost. If there were suffixes on "bird", they will be placed on "black—bird" after any suffixes from "black".

To keep affixes which would otherwise be lost, mention them as optional both in the match and replacement. To remove affixes which should not be kept, mention them as optional in the match, but not in the replacement.

Classes of affixes can be used to more easily mention all possible affixes. But the same class cannot be used twice on the same word. So multiple classes should be used for multiple affixes.

The second and following roots from compound roots are kept across root substitutions. For example, consider the following rule:

```
\ru black > Adj=white
```

When this rule is applied to "black--bird", it produces "white--bird".

Compound roots after the first are handled in a similar way to suffixes. Only the first root is matched by a stand-alone root.
Other roots are matched only if the first root is included in the match.

For example, consider the following rule:

```
\ru bird > N=cow
```

This rule will not change "black--bird" to "black--cow". (But it will change "bird--brain" to "cow--brain".)

To change "black--bird" to "black--cow", and any other kind of bird to the same kind of cow, use the following rule:

```
\ru N--bird > N--N=cow
```

In this rule "N" matches the first root, and "--bird" matches the second. Observe that "N" is used even though "black" is an adjective. This is because the "N" category is the category of the entire word, not of the first root.

23.Sentence Punctuation

The program uses a list of sentence punctuation marks to determine sentence boundaries. It defaults to using period, question mark, exclamation mark, colon, and semicolon as sentence punctuation.

You can change the list of sentence punctuation by including a "\sentpunc" field near the top of the rule file. The sentence punctuation marker is followed by a list of characters to be treated as sentence punctuation, as in the following example:

```
\sentpunc . ? !
```

The above line says that only period, question mark and exclamation mark are to be considered sentence punctuation. You can use the sentence punctuation field if you have a non-Roman script, or if for other reasons you want sentences to be broken differently than the default.

One useful possibility is to include comma in the list. This causes the sentence boundary marker to also apply to phrase boundaries marked by commas. Many rules apply equally well at both places.

Beginning with version 2.1.0, one *must* separate punctuation characters with spaces.

24.Punctuation Marks in Rules

It is also possible to refer directly to punctuation marks in rules. To do that the program needs a list of punctuation marks. This is given using a "`\punc`" field marker, followed by a list of punctuation marks. The program knows the sentence ending marks are punctuation, so you do not need to include those.

Beginning with version 2.1.0, one *must* separate punctuation characters with spaces.

The following example says that comma is punctuation:

```
\punc ,
```

The program defaults to considering comma, parends, slash, and hyphen as punctuation. This does not mean these have to be punctuation, but that they can be referred to as punctuation in rules if you want to do so.

To refer to a punctuation mark in a rule, just include it with spaces on both sides. For example, the following rule (for Frisian) says that the "lake/but" ambiguity is "but" after a comma:

```
\am but / , _ | Not lake after comma
```

Punctuation marks are allowed only on the match side of a rule.
They cannot be removed or inserted by a rule, and they do not need to be mentioned on the replacement side.

Begin punctuation is identified in the rule file in a field named `\bpunc`, which works just like the `\punc` field.

If I get requests for punctuation changing capability I will add it. But for now that is best done by running a consistent change table on the output text. CC is also the best way to do capitalization adjustments, which cannot be done with SENTRANS.

25.Notes on Categories in Rules

SenTrans normally uses the content of the `\cat` field (in the input ANA file) as the category for a word. If you invoke SenTrans with the -x option, then it will ignore this field and use the category of the root.

Beginning with version 2.2.0, one can tell SenTrans to use the category of the root in a given rule even though it would normally use the category of the word.
You do this by prepending a percent sign (%) before the category. For example, the rejection rule shown below will use the category of the root (vt) even though the category of the word might be something else (say, v).

```
\rej %vt=speak
```

This rule means to reject words with the root category 'vt' - irrespective of the computed, final category.

26.More Examples

Here are a few more examples from actual languages.

The following rule is from a Scotch to English adaptation. It is to change the input phrase "gie me't back" to "give it back to me". The AMPLE analysis calls the "'t" a suffix glossed "-IT".

```
\ru me -IT (back) > Pron=it (back) Prep=to me
```

This rule removes the suffix "-IT" and substitutes the word "it".
The word "it" must be given a category because it is not the same as "-IT", and so is considered an insertion. The rule also moves "me" from the front of the phrase to the end and adds the preposition "to" in front of it.

Here is another interesting rule. This one moves the word "yet" to the end of the sentence. It does that by inverting the position of the word "yet" and the words matched by the ellipses.

```
\ru yet ... # > ... yet
```

The following rule corrects the English of the first Frisian sentence in the CADA tutorial. In the tutorial it came out "Born and raised in Indonesia, will there his grave be." The rule changes it to "his grave will be there."

```
\cl Poss his her their your our my

\ru Aux Loc (Poss) N be >
 (Poss) N Aux be Loc
```

This rule actually needs to refer to a noun phrase between "there" and "be". The capability to do that is in process, but is not yet available in this version of the Sentence Transfer program.

27.Other Uses of Sentence Transfer

Sentence Transfer can also be used for other things besides disambiguating analyses and rearranging them for transfer to a target language.

One possibility is that Sentence Transfer can be used to avoid the need for null morphemes in AMPLE. To do this, run Sentence Transfer on the output of AMPLE with a rule file that adds the default morpheme wherever none of the others in the set occurs.
For example, the following rule puts a "3S" suffix on any verb that does not already have a person and number suffix.

```
\cl Pers 1S 1P 2S 2P 3P

\ru V > V -3S / _ ~-Pers
```

Another possible use of Sentence Transfer is to help make sentence parsing more effective. One example is that certain Greek conjunctions that introduce clauses are always shifted one word into the clause. The result is sentences that would analyze into something like "The but man went to town."

The word "but" in that position cannot be cleanly analyzed by most sentence parse programs, because it cannot be attached at the right level of the tree. It should be attached to the sentence above the clause, but it actually occurs between two constituents of the subject noun phrase of the clause. If it is attached to the sentence, then the line attaching it will have to cross the line attaching "the" to the noun phrase.

A reasonable solution to this dilemma is to use Sentence Transfer to move the "but" one word to the left before the sentence parse is performed. This allows it to attach exactly where it belongs.
Moving the "but" may also help in transfer to a target language, because it will no longer be in the way of transfer rules that change the order of the constituents in the noun phrase. If the target language does not want "but" in second position, then it is already out. If the target does want it in second position, then it can be moved back into second position after the rest of the sentence level transfer changes have been made.

Another way in which Sentence Transfer can help with sentence parsing is in handling clitics. A clitic is something attached to a word, but functioning at a higher syntactic level. In other words, it is something that syntactically acts like a separate word, but it is phonologically attached to another word. Such clitics raise similar problems to the Greek "but". Unless the syntactic parse builds different word structures than AMPLE has produced, the clitic must be attached under the same phrase as the word to which it is affixed.

Having a sentence parser build different word structures than AMPLE is possible, but it leads to many difficulties and complexities for the person writing the grammar. It is much easier to use Sentence Transfer to split the clitic off as a separate word before parsing. This is linguistically correct because the clitic really is acting like a word.

Splitting off clitics can make transfer easier. Clitics are sometimes in the way of transfers that affect the words to which they are attached. The transfer process is easier if clitics are split off first, then the words are rearranged, and then each clitic is attached to whatever word it adjoins. This correctly reflects the linguistics of the transfer process.

28.Running the Program

The Sentence Transfer program is named SENTRANS.EXE. It takes all its file names and options from the command line. I recommend that the command to run it be entered at the beginning of the synthesis batch file xxxSY.BAT.

Rule file -r, Input file -i, Output file -o

The program takes the names of the rule file, input file, and output file as command line arguments, as in the following example:

```
sentrans –r rules.tra –i file.ana –o file.ant
```

The -r gives the rules file, the -i gives the input file, and the -o gives the output file. I recommend that the output file be given a file type of .ANT (for Analysis Transferred).

Ambiguity Monitor Option -a

The -a option shows a dot or digit for each word output, just like the -m option in AMPLE and STAMP. It shows the number of ambiguities remaining in the output analysis file. This can be useful to tell you how effective your disambiguation is. It shows how many words are still ambiguous. To see how many places your rules remove ambiguities, use the -m option described below.

Rule Application Monitor Option -m

The -m option shows a dot or digit for each sentence processed. If no rules are applied to a sentence, then them monitor output shows a dot. If rules are applied, then a digit is output showing the number of rules applied to the sentence. If more than 9 rules are applied, the digit 9 is output. The dots and digits are grouped into 10's and 50's like those from AMPLE and STAMP. The -m option is very useful for transfer rules, as it shows how many rules are being applied. During disambiguation it tells how many places have been disambiguated, but -a may be more helpful there, because it shows how many are still ambiguous.

Trace Option -t

Another option is -t, which gives a trace of rule applications. If this option is used, then for every rule that matches, the program displays the analysis before the rule is applied, the part that is matched, and the resulting analysis after the rule is applied.

For example, consider the following rule:

```
\am N / _ . | Sentence must end with noun
```

When this rule is applied to a sentence that ends in "little bears", where "bears" is ambiguous between "V" and "N", the trace looks as follows:

```
Before: Adj=little V=bear-PRES%N=bear-PLUR
Rule: N .
```

```
Matched: N=bear-PLUR
After: Adj=little N=bear-PLUR
```

The "Before:" line shows one word of context before the matched word, the matched word, and some following context if there is any. The Category of each word is shown with an equal sign between it and the word. Ambiguities are shown separated by percent sign.

The "Rule:" line shows the rule that matched. The "Matched:" line shows the words that were matched. And the "After:" line shows the result after disambiguation and any requested rearrangement has been performed. In the example above the "N/V" ambiguity has been removed, and only the noun form is left.

The trace is very helpful in showing how effective rules are. It is also useful in that it shows all of the places a particular rule applies. By looking at the trace you can see if a rule is applying too broadly and making changes in places where it should not apply.

If you have looked through your source file and are aware of a number of places where a rule should apply, the trace can tell you if the rule is applied in all of those places. This is especially useful if you test your transfer rules on a set of sample sentences that illustrate all of the things your rules should do.

A useful technique for studying the effects of a particular rule is to extract the rule to a temporary rule file. (This can be done easily with the "file write" command of ED, for example.)
Then run Sentence Transfer using that temporary rule file and the trace option. The result will be a trace of only the places that particular rule applies.

29.All of the Examples Together

Below are two sample files containing most of the examples given in this document. They are given here for reference, and so that you can see what a complete rule file looks like. I recommend file types of .AMB for the disambiguating file and .TRA for the transfer rule file.

Example 1

```
\id sample.amb Sample SENTRANS Disambiguation File

\ca N V Pron Adj Adv

\am V -INF / V _ | INF after verb
\am V -PRES_1S/2SPO/123PL | Tense elsewhere

\am they / _ V -PRES_1S/2SPO/123PL | they before plural
\am she / _ V -PRES_3S | she before singular
```

```
\am but / # _ | Not lake at sentence start
\am but / , _ | Not lake after comma

\am V -INF / V (N) _ | INF after verb

\am she / _ V -PRES_3S | She before singular
\am she / _ V ~-PRES_1S/2SPO/123PL | she before non-plural
\am she / _ ... V -PRES_3S | she anywhere before singular
```

Example 2

```
\id sample.tra Sample Sentence Transfer Rule File

\ca N V Pron Adj Adv

\ru V usually > usually V

\ru Num Class N > N Num Class

\cl Aux has have will must
\cl Loc here there
\cl Neg never not

\ru (Loc) (Neg) (back) V >
 (Neg) V (back) (Loc) / Aux _

\ru (Loc) (Neg) (back) be >
 be (Neg) (back) (Loc) / Aux ~_

\ru (Loc) (Neg) (back) V > | This rule feeds the next
 (Neg) V (back) (Loc)

\ru Neg be > be Neg / ~Aux _ (back) Loc

\ru receive > / receive _ | Delete a word

\ru N > N Prep=belong / _ Pron | Insert a word

\ru -POSS > Part=possessive | Split

\ru possessive > -POSS | Join

\ru yet ... # > ... yet
```

Send bug reports and feature requests to: Alan Buseman,
International Computer Services, JAARS, Box 248, Waxhaw, NC
28173.