

# Enhancing Machine Translation via Frame-Semantic Data

Matthew G. Sachs  
Department of Linguistics  
Brandeis University  
Waltham, MA  
matthewg@brandeis.edu

December 23, 2004

## Abstract

Frame semantics is an approach to examining meaning in natural language by considering clusters of related concepts. For instance, in the “commercial transaction” frame, there is a buyer, a seller, goods, and money; different predicates in this frame will place these agents in different syntactic roles, so, in English, the buyer will be the subject of *buy* while the seller will be the subject of *sell*.

Frame semantics presents a powerful aide to machine translation. Frame-semantic knowledge of an input phrase facilitates more precise word-sense disambiguation and allows greater flexibility in deciding which of multiple valid word orderings to emit in the target language. I have demonstrated this by creating a rudimentary system for translating from Spanish to English which can optionally take advantage of frame-annotated input, and then testing this system on a small corpus of phrases in the commercial transaction frame.

## 1 Methodology

There are a few projects attempting to do frame semantic work in other languages, including Spanish FrameNet<sup>1</sup> and German FrameNet<sup>2</sup>, the latter of which is geared towards machine translation, but both of these projects are in their early stages, so I had to construct my own data.

### 1.1 Assembling the Corpus

A small subcorpus of phrases containing predicates common in commercial transaction frames<sup>3</sup> was assembled. The commercial transaction frame was

selected because it has already been well-studied in English.<sup>4</sup> Target predicates were gathered by taking Fillmore and Atkins’s list, *buy*, *sell*, *charge*, *spend*, *pay*, *cost*, and finding the translations for those words in a Spanish-English dictionary.<sup>5</sup>

Next, phrases containing these predicates were extracted from a larger corpus, the “Corpus Lingüístico de Referencia de la Lengua Española en Chile.”<sup>6</sup> Because the corpus had not yet been parsed, this was done by listing all the conjugated forms of these predicates and grabbing any line of text on which one of those strings appeared as well as several lines before and after each appearance.

---

<sup>1</sup>Subirats and Petruck [2003]

<sup>2</sup>Boas [2002]

<sup>3</sup>Situations in the commercial transaction frame involve a *buyer* giving *money* to a *seller* in exchange for *goods*. Common English predicates in this frame include *buy* and *sell*.

<sup>4</sup>Fillmore and Atkins [1992]

<sup>5</sup>Carvajal and Horwood [1996]

<sup>6</sup>Sociedad Estatal del Quinto Centenario and La Agencia Española de Cooperación Internacional, Instituto de Cooperación con Iberoamérica; <http://www.llf.uam.es/~marcos/informes/corpus/>

The extractions were segregated by the lemmas of their keyword predicates, enabling the maintenance of distinct subcorpora for each predicate.

Finally, the subcorpora were tokenized, tagged, and parsed; the FreeLing suite<sup>7</sup> was used for this.

## 1.2 Frame Annotation

The parsed subcorpus for the Spanish predicate *cobrar*<sup>8</sup> was then hand-annotated with frame information. Phrases which were in the commercial transaction frame were delineated with the markers #FRAME-START and #FRAME-END, and any words which corresponded to particular agents in the frame were annotated with whichever of the markers #BUYER, #SELLER, #GOODS, and #MONEY corresponded to the frame role for the particular phrase. For this paper, I did not annotate the other subcorpora or attempt to run them through the translation program.

## 1.3 A Rudimentary Translator

Finally, a simple machine translation program was written.<sup>9</sup> The translator has a lexicon<sup>10</sup> and a set of phrase structure/syntax/grammar rules.<sup>11</sup> It reads a parsed, frame-annotated input in Spanish and, for each phrase marked as lying in the commercial transaction frame, outputs two translations, one which takes advantage of the frame data (the fact that we are in the commercial transaction frame as well as frame roles assigned to specific words or phrasal groups) and one which does not. The intent of the program is not to provide a robust general-purpose translation mechanism, but rather to compare the translation we are able to perform without taking advantage of frame data with the frame-enhanced translation.

For lexical entries with multiple alternative translations, all possible translations are output,

---

<sup>7</sup>Carreras, Chao, Padró, and Padró [2004]

<sup>8</sup>to charge, to earn, to recover, to give

<sup>9</sup>The source code for this program has been released into the public domain and is included in appendix 5.2, page 6.

<sup>10</sup>See appendix 5.1, page 6.

<sup>11</sup>See appendix 5.2.3, page 11.

bracketed by question marks; particular translations can optionally be marked as being either the “frame sense” (the translation to use when the input is in the commercial transaction frame) or the translation for a particular role within the frame. For instance, the Spanish verb *cobrar* has four possible English translations, and the *to charge* translation is marked as the “frame sense”, so *cobrar* only has one valid translation when we are making use of frame data. *Él*, which can be translated as *he* or *it*, has the “he” translation marked with the “BUYER” role and the “it” translation marked with the “MONEY” role, so while it defaults to having multiple valid translations, if we are making use of frame data and the particular *él* token being translated is marked as the buyer in the frame, the translation will be given as *he*.

Execution of individual phrase structure/syntax/grammar rules can also be conditional on whether the translation is in “frame-enhanced” mode or not.

## 2 Results

The output of the translation program on the “*cobrar* commercial transactions” subcorpus is below. In each group of three phrases, the first is the Spanish input, the second is the baseline translation, and the third is the frame-enhanced translation.

- (1)
  - a. los propietarios cobraran el arriendo del mes completo
  - b. the proprietors (?? charged, earned, recovered, gave ??) the rent of\_the complete month
  - c. the proprietors charged the rent of\_the complete month
- (2)
  - a. el Hotel\_Pacífico no ha cobrado nunca una cuenta semanal
  - b. the Hotel\_Pacifico never has (?? charged, earned, recovered, given ??) one weekly (?? bill, account, sum, count ??)
  - c. the Hotel\_Pacifico never has charged one weekly bill

- (3) a. el gobierno tiene que cobrar mas impuestos  
 b. the government has (?? to\_charge, to\_earn, to\_recover, to\_give ??) more taxes  
 c. the government has to\_charge more taxes
- (4) a. lo que se cobra por cajetilla  
 b. that\_which (?? is\_charged, is\_earned, is\_recovered, is\_given, charges, earns, recovers, gives ??) for\_a pack  
 c. that\_which is\_charged for\_a pack
- (5) a. el impuesto se cobra  
 b. the tax (?? is\_charged, is\_earned, is\_recovered, is\_given, charges, earns, recovers, gives ??)  
 c. the tax is\_charged
- (6) a. de cobrar impuestos sólo a quien honestamente desea pagarlos  
 b. (?? to\_charge, to\_earn, to\_recover, to\_give ??) taxes only to\_those\_who honestly want to pay them  
 c. to\_charge taxes only to\_those\_who honestly want to pay them
- (7) a. de cobrar el impuesto  
 b. (?? to\_charge, to\_earn, to\_recover, to\_give ??) the tax  
 c. to\_charge the tax
- (8) a. Cada tarea la cobro a diez\_pesos  
 b. each (?? task, homework ??) (?? I\_charge, I\_earn, I\_recover, I\_give ??) (?? her,it ??) 10\_pesos  
 c. each (?? task, homework ??) I\_charge her 10\_pesos
- (9) a. se la cobro  
 b. (?? I\_charge, I\_earn, I\_recover, I\_give ??) (?? her, it ??) (?? to\_him, to\_it ??)  
 c. I\_charge him it

## 2.1 Frame-Enabled Enhancements

One of the major enhancements available by taking into account the frame information is greater

precision in word-sense disambiguation. Simply by knowing that each instance of *cobrar* means *to charge*, we enable a significantly more precise translation. The other sense disambiguation enabled was the role-specific sense assignment of pronouns. If a pronoun has the “money” role, we can reasonably guess that it is probably a thing and not a person; likewise, a pronoun with the “buyer” or “seller” role is probably a person. The person/thing distinction allows selection between the gendered and neuter pronouns in English.

The other enhancement enabled by frame information is seen in example 9. Here, we have a phrase structure/syntax/grammar rule which we enable only in the commercial transaction frame. This rule states that instead of the normal action we would take when seeing two consecutive pronouns,<sup>12</sup> we maintain the order of the pronouns and do **not** lexicalize the indirect pronoun with a preceding *to*. So, instead of the standard translation seen in example P, we get:

- (10) Te lo digo.  
 you it I tell  
 I tell you it.

The reason that we only do this when taking advantage of frame information is that, while it is an acceptable English translation when *cobrar* is translated as *to charge*,<sup>13</sup> it is not acceptable for all

<sup>12</sup>In Spanish, when a verb has both an indirect and a direct object pronoun associated with it, both pronouns immediately precede the verb. The indirect pronoun must be before the direct pronoun, and in most cases the indirect pronoun is lexicalized differently than it would be if it appeared alone, in a form which is less specific in person and gender. Thus, our normal rule for transforming two consecutive pronouns is, after they’ve already been moved from before the verb to after the verb, we reverse the order of the two pronouns and the indirect pronoun is lexicalized in English with *to* in front of it. Thus:

- (P) Te lo digo.  
 you it I tell  
 I tell it to you.

<sup>13</sup>It does not appear that it is ever mandatory to use this translation form in English, although in many cases not using it results in a translation which is acceptable but less ideal.

possible translations of the verb *cobrar*:

- (11) Te lo cobro.  
you it I recover  
\*I recover you it.

## 2.2 Further Possible Enhancements

The syntactic distribution of frame roles for *cobrar* in the commercial transaction frame in Spanish appears to be identical to the syntactic distribution of frame roles for *to charge* in the commercial transaction frame in English. I have not examined the syntactic distribution of frame roles in the other Spanish commercial transaction predicates, but if they, or some other frame-predicate, had a different syntactic distribution than the corresponding English predicate, reconciling the two syntaxes would become possible through the use of frame information.

## 2.3 Viability in Real-World Machine Translation Systems

### 2.3.1 The Frame Annotation Problem

Word-sense disambiguation is a well-studied problem, and there are many other approaches to it, although it is still an area of active research. Hand-annotation of phrases with frame information may not be viable in many real-world situations, although in some, such as human-aided translation, manual frame annotation could be viable. However, it may be possible to attempt automatic assignment of phrases to frames and frame roles; detailed investigation of the implementation of such a scheme is an area for further research.

One possible approach would make use of syntactic information. Fillmore and Atkins<sup>14</sup> have shown that the syntactic distribution of entities in a phrase to frame roles is syntactically motivated in a manner instantiated by the particular frame-predicate being used. In conjunction with probabilistic analysis of which sets of words tend to fill which roles in which frames, and collocations in-

dicative of certain frames, it may be possible to do this with a reasonable degree of accuracy.

Probabilistic analysis may also be a usable means by which to infer the existence of a frame from a predicate. Corpus analysis could be used to determine how likely it is that *cobrar* indicates the commercial transaction frame.

### 2.3.2 Word Sense Disambiguation

Blithely assuming that all instances of pronominal buyers in Spanish should be lexicalized as gendered in the English translation is overly simplistic. A corporation, for instance, is a neuter entity which can be a buyer or seller. That particular disambiguation should normally be performed through a combination of anaphora resolution and lexical data, although in certain circumstances frame-based disambiguation may be less expensive (due to the added complexity and computational cost of inserting an anaphora resolution stage into a processing pipeline) and sufficiently accurate.

## 3 Conclusions

Frame semantics is a useful tool for applying semantic information to machine translation. Because it is organized directly around principles of the relation between multiple semantic entities in an interaction, it is a semantic framework which is particularly well-suited to this task. Some of the challenges in machine translation to which it can be applied are word-sense disambiguation and target language word ordering.

It would be interesting to see what sorts of enhancements frame information might enable in an existing machine translation system. Such a system would already have mechanisms for handling word-sense disambiguation and word reordering, and attempting to augment these mechanisms would be an important test. Furthermore, by taking advantage of an existing system, a much broader range of input could be tried without the added labor of having to make significant additions to the system's lexical and grammatical knowledge with each new phrase given to it.

<sup>14</sup>Fillmore and Atkins [1992]

Another worthwhile area for further research would be applying this translation system to phrases containing a frame-predicate whose syntactic distribution differs significantly between the source and target languages. Precise translation of these phrases is probably a task which is extremely well-suited to take advantage of frame information. As the non-English FrameNet projects advance, data on the identity and nature of these frames will be come more numerous.

**Acknowledgments** Discussions with the following proved extremely helpful in developing the ideas discussed in this paper: Patrick Hanks, Ray Jackendoff, James Pustejovsky, Jonathan Sagotsky, and Jonathan North Washington.

## 4 References

Hans C. Boas. Bilingual FrameNet dictionaries for machine translation. In M. González Rodríguez and C. Paz Suárez Araujo, editors, *Proceedings of the Third International Conference on Language Resources and Evaluation*, volume IV, pages 1364–1371, Las Palmas, 2002.

Xavier Carreras, Isaac Chao, Lluís Padró, and Muntsa Padró. Freeling: An open-source suite of language analyzers. *Processings of the 4th International Conference on Language Resources and Evaluation*, 2004. URL <http://www.lsi.upc.edu/~nlp/freeling/>.

Carol Styles Carvajal and Jane Horwood, editors. *The Oxford Spanish Dictionary: New International Edition*. Oxford University Press, 1996.

Charles J. Fillmore and Beryl T. Atkins. *Toward a Frame-Based Lexicon: The Semantics of RISK and its Neighbors*, pages 75–102. Lehrer and Kit-tay, 1992.

Carlos Subirats and Miriam R.L. Petruck. Suprise: Spanish framenet. *Workshop on Frame Semantics, International Congress of Linguists*, July 2003.

## 5.1 Translation Lexicon

Cada:DI(each)  
 Hotel\_Pacífico:NP(Hotel\_Pacifico)  
 arriendo:VMG(rent)  
 cajetilla:N(pack)  
 cobra:VMIP3S0\_PASSIVE>(\*is\_charged,is\_earned,is\_recovered,is\_given+subj=3S)  
 cobra:VMIP3S(\*charges,earns,recovers,gives+subj=3S)  
 cobrado:VMP(\*charged,earned,recovered,given)  
 cobrar:VMN(\*to\_charge,to\_earn,to\_recover,to\_give)  
 cobrarán:VMSI3P(\*charged,earned,recovered,gave+subj=3P)  
 cobro:VMIP1S(\*I\_charge,I\_earn,I\_recover,I\_give+subj=1S)  
 completo:A(complete)  
 cuenta:N(?MONEY:bill,?BUYER:account,sum,count)  
 desea:VMIP3S(want+subj=3S)  
 diez\_pesos:Z(10\_pesos)  
 gobierno:N(government)  
 ha:VAIP3S(has+subj=3S)  
 honestamente:R(honestly)  
 impuesto:N(tax)  
 impuestos:N(taxes)  
 más:(more)  
 mes:N(month)  
 no:R(no)  
 nunca:R(never)  
 pagarlos:VMN(to pay them)  
 propietarios:N(proprietors)  
 semanal:A(weekly)  
 sólo:A(only)  
 tarea:N(task, homework)  
 tiene:VMIP3S(has+subj=3S)  
 una:D(one)  
 los:D(the)  
 el:D(the)  
 él:P(?BUYER:he,?MONEY:it)  
 ella:P(?BUYER:her,?MONEY:it)  
 lo:P(?BUYER:him,?MONEY:it)  
 la:P(?BUYER:her,?MONEY:it)  
 se:P0(?BUYER:to\_him,?MONEY:to\_it)  
 lo\_que:(that\_which)  
 del:(of\_the)

## 5.2 Translator Source Code

I relinquish all copyright interest in this code to the public domain.

```

#!/usr/bin/perl

use strict;
use warnings;
use Storable qw(dclone);
use FindBin;
use lib "$FindBin::Bin/lib";
use frametrans::node;
use frametrans::lexicon;
use frametrans::psg;
10

sub print_tree($);
my $in_frame = 0;
my $group = frametrans::node->new();
my $root = $group;

binmode(STDOUT, ":utf8");
die "Usage: $0 lexicon input\n" unless @ARGV == 2;
my $lexicon = frametrans::lexicon->new(shift);
20

my $infile = shift;
open(INPUT, "<:utf8", $infile) or die "Couldn't open input file $infile: $!\n";
while(<INPUT>) {
    chomp;

    if($in_frame) {
        if(/#FRAME-END/) {
            $in_frame = 0;
            30

            my $spanish = $root;
            my $english_noframe = dclone($root);
            my $english_frame = dclone($root);

            print_tree($spanish);
            print "\n";

            frametrans::psg::translate($english_noframe, 0);
            $lexicon->translate($english_noframe, 0);
            print_tree($english_noframe);
            print "\n";
            40

            frametrans::psg::translate($english_frame, 1);
            $lexicon->translate($english_frame, 1);
            print_tree($english_frame);
            print "\n\n";

            $root = $group = frametrans::node->new();
            next;
            50
        } else {
            my $framerole = "";
            my $inspoint;

```

```

my $prevpoint;
my $otherpoint;

if(/#(GOODS|MONEY|SELLER|BUYER)/) {
    $framerole = $1;
}
60

if(
/^ # Start of a line
(\s*) #Some whitespace ($1)
(?: #Either...
(?(.*)-\[) #Some gunk ($2) followed by "[" (phrase group)
| #Or...
(?:\((.*)\)) #Stuff inside parentheses ($3) (tagged, lemmatized token)
)
/x) {
70
    my $prefix = $1;
    my $class = $2;
    my $tagtok = $3;

    while(length($prefix) < length($group->{prefix})) {
        $group = $group->{parent};
        while($group->{next}) {
            $group = $group->{next};
        }
    }
80

    if($prefix eq $group->{prefix}) {
        $inspoint = "next";
        $prevpoint = "prev";
        $otherpoint = "parent";
    } else {
        $inspoint = "child";
        $prevpoint = "parent";
    }
}
90

my $newgroup = frametrans::node->new();
$group->{$inspoint} = $newgroup;
$newgroup->{$prevpoint} = $group;
$newgroup->{$otherpoint} = $group->{$otherpoint} if $otherpoint;
$newgroup->{framerole} = $framerole;
$newgroup->{prefix} = $1;

if($class) {
    $newgroup->{class} = $class;
} else {
100
    $newgroup->{class} = "TOKEN";

    $tagtok =~ /^(.*) (.*) (.*)$/ or
        die "Couldn't parse tagged Spanish token: $tagtok\n";
    $newgroup->{token} = $1;
    $newgroup->{lemma} = $2;
    $newgroup->{tag} = $3;
}

$group = $newgroup;
110

```

```

    } elsif(/(\s+)/) {
        # Do nothing
    } else {
        die "Couldn't parse: $_\bn";
    }
}
} else {
    if(/#FRAME-START/) {
        $in frame = 1;
        next;
    }
}
}

sub print_tree($) {
    my($node) = @_;
    my $show_ps = 0;

    my $emitted = 0;

    NODE: while($node) {
        if($node->{class} ne "TOKEN" and $show_ps) {
            print $node->{prefix}, "<", $node->{class}, ">\n";
            $emitted = 1;
        } elsif($node->{class} eq "TOKEN") {
            if($show_ps) {
                print $node->{prefix};
                $emitted = 1;
            }
            print $node->{token}, " ";
        }

        if($node->{child}) {
            print_tree($node->{child});
        }

        if($node->{class} ne "TOKEN" and $show_ps) {
            print $node->{prefix}, "</", $node->{class}, ">";
            $emitted = 1;
        }

        $node = $node->{next};
        if($show_ps and $emitted) {
            print "\n";
            $emitted = 0;
        }
    }
}
}

```

120

130

140

150

---

## 5.2.2 frametrans::lexicon, the lexicon parser and lexical translation module

---

```
package frametrans::lexicon;
```

```
use strict;
use warnings;
```

```
sub new($$) {
    my $class = ref($_[0]) || $_[0] || "frametrans::lex";
    my $lexfile = $_[1];

    my %lexicon;
    open(LEXICON, "<:utf8", $lexfile) or die "Couldn't open lexicon $lexfile: $!\n";
    while(<LEXICON>) {
        /^(.+?):(.*)\((.*)\)$/ or die "Couldn't parse lexical entry: $_";
        my($lexeme_es, $tag, $lexemes_en) = ($1, $2, $3);

        $lexicon{$lexeme_es} ||= {};
        $lexicon{$lexeme_es}->{$tag} ||= {metadata => "", lexemes_en => []};
        my $entry = $lexicon{$lexeme_es}->{$tag};

        if($lexemes_en =~ s/\+(.*)$//) {
            $entry->{metadata} = $1;
        }
        my @lexemes = split(/,/ , $lexemes_en);
        foreach my $lexeme (@lexemes) {
            my $lex_entry = {};

            if($lexeme =~ s/^\*///) {
                $lex_entry->{framesense} = 1;
            } else {
                $lex_entry->{framesense} = 0;
            }

            if($lexeme =~ s/^\?(.*)://) {
                $lex_entry->{framerole} = $1;
            } else {
                $lex_entry->{framerole} = "";
            }

            $lex_entry->{word} = $lexeme;

            push @{$entry->{lexemes_en}}, $lex_entry;
        }
    }
    close LEXICON;

    my $self = \%lexicon;
    bless $self, $class;
    return $self;
}
```

```
sub translate($$$;$) {
    my($lexicon, $node, $use_framedata, $parent_framerole) = @_;
    my($framerole);

    NODE: while($node) {
        my $node_framerole = $node->{framerole} || "";
        $framerole = $node_framerole || $parent_framerole || "";
    }
}
```

```

if($node->{class} eq "TOKEN") {
    next if $node->{translated};
    my($token, $lemma, $tag) = ($node->{token}, $node->{lemma}, $node->{tag});

    # Get lexemes which have the right POS
    my @lexemes_en = map {
        @{$lexicon->{$token}->{$_-}>{lexemes_en}}
    } grep {
        substr($tag, 0, length($_-)) eq $_-
    }
    sort {length($b) <=> length($a)} # More specific tags are tried first
    keys %{$lexicon->{$token}}};

if(@lexemes_en == 0) {
    $node->{token} = "'$token'";
    next;
} elsif(@lexemes_en == 1) {
    $node->{token} = $lexemes_en[0]->{word};
} else {
    if($use_framedata) {
        foreach my $lexeme (@lexemes_en) {
            if($lexeme->{framesense}) {
                $node->{token} = $lexeme->{word};
                next NODE;
            } elsif($lexeme->{framerole} and
                $lexeme->{framerole} eq $framerole
            ) {
                $node->{token} = $lexeme->{word};
                next NODE;
            }
        }
    }

    $node->{token} =
        "(?? " . join(" ", map {$_->{word}} @lexemes_en) . " ??)";
}
} continue {
    if($node->{child}) {
        $lexicon->translate($node->{child}, $use_framedata, $framerole);
    }

    $node = $node->{next};
}
}

```

1;

---

### 5.2.3 frametrans::psg, the phrase structure/syntax/grammar rule applicator

---

*# Translations for phrase structure/syntax/grammar*

```
package frametrans::psg;
```

```
use strict;  
use warnings;  
use frametrans::node;
```

```
sub translate {  
    my($node, $use_framedata) = @_;  
  
    while($node) {  
        # The order of these is important!  
        # Certain rules assume that certain motions have  
        # already been done.  
  
        psrule_adjmove($node);  
        psrule_adjmove2($node);  
        psrule_advnegmove($node);  
        psrule_prepinf($node);  
        psrule_loque($node);  
        psrule_passive($node);  
        psrule_fornoun($node);  
        psrule_aquien($node);  
        psrule_anum($node);  
        psrule_promove($node);  
        if($use_framedata) {  
            psrule_indpro($node);  
        } else {  
            psrule_proswap($node);  
        }  
  
        if($node->{child}) {  
            translate($node->{child}, $use_framedata);  
        }  
    } continue {  
        $node = $node->{next};  
    }  
}
```

```
#RULE: Move adjectives from after nouns to before nouns
```

```
#N  
#s-a*: A  
sub psrule_adjmove {  
    my($adj) = @_;  
    return unless $adj->{class} =~ /^s-a/;  
  
    my $noun = $adj->find_predecessor("TOKEN", "N");  
    return unless $noun;  
  
    $adj->unlink();  
    $adj->insert_before($noun);  
}
```

```
#RULE: Another adjective move rule. This one finds adjective groups inside noun groups.
```

```
#grup-nom:  
# foo  
# s-a:
```

```

sub psrule_adjmove2 {
    my($adj) = @_;
    return unless $adj->{class} =~ /^s-a/;

    my $noungrp = $adj->find_container("grup-nom");
    return unless $noungrp;

    $adj->unlink();
    $adj->insert_before($noungrp->{child});
}

#RULE: Move adverbs from after negated verbs to before them and delete the neg
#neg:
#grup-verb:
#sadv:
sub psrule_advnegmove {
    my($adv) = @_;
    return unless $adv->{class} =~ /^sadv/;

    my $verb = $adv->find_predecessor("grup-verb");
    return unless $verb and $verb->{prev} and $verb->{prev}->{class} =~ /^neg/;

    $adv->unlink();
    $verb->{prev}->unlink();
    $adv->insert_before($verb);
}

#RULE: Remove a prep before an infinitive
#grup-verb:
# que
# infinitiu:
#OR:
# prep:
# infinitiu:
sub psrule_prepinf {
    my($prep) = @_;
    return unless
        ($prep->{class} eq "TOKEN" and $prep->{lemma} eq "que") or
        $prep->{class} =~ /^prep/;
    return unless $prep->{next} and $prep->{next}->{class} =~ /^infinitiu/;

    $prep->unlink();
}

#RULE: Replace "lo que" with "that which"
#espec-ms: lo
#que
sub psrule_loque {
    my($lo) = @_;
    return unless $lo->{class} eq "TOKEN" and $lo->{lemma} eq "e1";
    return unless
        !$lo->{next} and
        $lo->{parent}->{next} and
        $lo->{parent}->{next}->{class} eq "TOKEN" and
        $lo->{parent}->{next}->{lemma} eq "que";
}

```

```

$lo->{parent}->{next}->unlink();
$lo->{lemma} .= "_que";
$lo->{token} .= "_que";
}

#RULE: Add _PASSIVE_ to tag of verb preceeded by se, delete the “se”
#grup-verb: se
# grup-verb:
sub psrule_passive {
  my($se) = @_;
  return unless $se->{class} eq "TOKEN" and $se->{token} eq "se";
  return unless $se->{next} and $se->{next}->{class} =~ /^grup-verb/;

  # The verb could be buried in multiple layers
  my $verb = $se->find_successor("TOKEN", "V");
  return unless $verb;

  $verb->{tag} .= "_PASSIVE_";
  $se->unlink();
}

#RULE: Replace “por” before a noun with “for a”
#prep: por
#sn:
sub psrule_fornoun {
  my($prep) = @_;
  return unless $prep->{class} =~ /^prep/ and $prep->{child};
  return unless $prep->{child}->{class} eq "TOKEN" and $prep->{child}->{token} eq "por";
  return unless $prep->{next}->{class} =~ /^sn/;

  $prep->{child}->{token} = "for_a";
  $prep->{child}->{translated} = 1;
}

#RULE: Replace “a quien” with “to those who”
#prep: a
#quien
sub psrule_aquien {
  my($prep) = @_;
  return unless $prep->{class} =~ /^prep/;
  return unless $prep->{child} and
    $prep->{child}->{class} eq "TOKEN" and
    $prep->{child}->{token} eq "a";
  return unless $prep->{next} and
    $prep->{next}->{class} eq "TOKEN" and
    $prep->{next}->{lemma} eq "quien";

  $prep->{child}->{token} = "to_those_who";
  $prep->{child}->{translated} = 1;
  $prep->{next}->unlink();
}

#RULE: Delete “a” before a number
#prep: a
#Z

```

```

sub psrule_anum {
  my($a) = @_;
  return unless $a->{class} eq "TOKEN" and $a->{token} eq "a";

  my $next = $a->{parent}->{next};
  while($next) {
    if($next->{class} eq "TOKEN" and $next->{tag} =~ /^Z/) {
      $a->unlink();
      return;
    } else {
      $next = $next->{child};
    }
  }
}

#RULE: Move pronouns after verbs
#P*
#grup-verb:
sub psrule_promove {
  my($pro) = @_;

  my @pro;
  while($pro and $pro->{class} eq "TOKEN" and $pro->{tag} =~ /^P/) {
    unshift @pro, $pro;
    $pro = $pro->{next};
  }
  return unless @pro;
  $pro = $pro[0];

  my $verb = $pro->find_successor("grup-verb") || $pro->find_successor("TOKEN", "V");
  return unless $verb;

  foreach $pro (@pro) {
    $pro->unlink();
    $pro->insert_after($verb);
  }
}

#RULE [FRAME ONLY]: Remove to_ from the translation of an indirect object pronoun.
#We do this by faking it into a direct object pronoun.
#P...[~A]
sub psrule_indpro {
  my($pro) = @_;
  return unless $pro->{class} eq "TOKEN" and $pro->{tag} =~ s/^(P...)[~A]/$1A/;

  if($pro->{token} eq "se" or $pro->{token} eq "le") {
    $pro->{token} = "lo";
  }
}

#RULE: Swap the order of two adjacent pronouns
#P P
sub psrule_proswap {
  my($pro) = @_;
  return unless $pro->{class} eq "TOKEN" and $pro->{tag} =~ /^P/;
  return unless $pro->{next} and

```

180

190

200

210

220

```
$pro->{next}->{class} eq "TOKEN" and  
$pro->{next}->{tag} =~ /^P/;
```

```
my $next = $pro->{next};  
$next->unlink();  
$next->insert_before($pro);  
}
```

230

1;

---

## 5.2.4 frametrans::node, utility functions for managing the parse tree

---

```
# Nodes in frametrans phrase structure
```

```
package frametrans::node;
```

```
use strict;  
use warnings;
```

```
sub new {  
    my $class = ref($_[0]) || $_[0] || "frametrans::node";  
    my $self = {  
        prev => undef,  
        parent => undef,  
        child => undef,  
        next => undef,  
        prefix => "",  
        framerole => "",  
        text => "",  
        class => ""  
    };  
    bless $self, $class;  
    return $self;  
}
```

10

20

```
# Unlink the subtree rooted at self
```

```
sub unlink($) {  
    my($node) = @_;  
  
    if($node->{prev}) {  
        $node->{prev}->{next} = $node->{next};  
    } elsif($node->{parent}) {  
        $node->{parent}->{child} = $node->{next};  
    }  
  
    if($node->{next}) {  
        $node->{next}->{prev} = $node->{prev};  
    }  
  
    $node->{parent} = $node->{prev} = $node->{next} = undef;  
}
```

30

40

*# Insert a subtree just prior to another node, at the same level*

```
sub insert_before($$) {  
    my($node, $inspoint) = @_;  
  
    $node->{parent} = $inspoint->{parent};  
    $node->{prev} = $inspoint->{prev};  
    if($node->{prev}) {  
        $node->{prev}->{next} = $node; 50  
    } else {  
        $node->{parent}->{child} = $node;  
    }  
  
    $inspoint->{prev} = $node;  
    $node->{next} = $inspoint;  
}
```

*# Insert a subtree just after to another node, at the same level*

```
sub insert_after($$) { 60  
    my($node, $inspoint) = @_;  
  
    $node->{parent} = $inspoint->{parent};  
    $node->{next} = $inspoint->{next};  
    if($node->{next}) {  
        $node->{next}->{prev} = $node;  
    }  
  
    $inspoint->{next} = $node;  
    $node->{prev} = $inspoint; 70  
}
```

```
sub matches($$;$) {  
    my($node, $class, $tag) = @_;  
    $tag ||= "";  
  
    return if $node->{class} !~ /^$class/;  
    return if $node->{class} eq "TOKEN" and $node->{tag} !~ /^$tag/;  
    return 1; 80  
}
```

```
sub find_container($$;$) {  
    my($node, $class, $tag) = @_;  
  
    while($node) {  
        next unless $node->matches($class, $tag);  
        return $node;  
    } continue {  
        $node = $node->{parent}; 90  
    }  
  
    return 0;  
}
```

```
sub find_predecessor($$;$) {  
    my($node, $class, $tag) = @_;
```

```

while($node) {
  while($node) {
    next unless $node->matches($class, $tag);
    return $node;
  } continue {
    last unless $node->{next};
    $node = $node->{prev};
  }

  $node = $node->{parent};
}

return undef;
}

sub find_successor($$;$) {
  my($node, $class, $tag) = @_;

  while($node) {
    while($node) {
      next unless $node->matches($class, $tag);
      return $node;
    } continue {
      last unless $node->{next};
      $node = $node->{next};
    }

    $node = $node->{child};
  }

  return undef;
}

1;

```

100

110

120

130