Gefördert vom
bmb+f
Bundesministerium für
Bildung, Wissenschaft,
Forschung und Technologie

Verbmobil
Verbundvorhaben

Using Templates to Encode Transfer Rules

Bianka Buschbeck-Wolf

Universität Stuttgart

Vm
Report 229
Oktober 1998

Oktober 1998

Bianka Buschbeck-Wolf

Institut für Maschinelle Sprachverarbeitung
Universität Stuttgart
Azenbergstraße 12
D – 70174 Stuttgart

Tel.: (0711) 121 - 1367
Fax: (0711) 121 - 1366
e-mail: `bianka@ims.uni-stuttgart.de`

Gehört zum Antragsabschnitt: 3.1 Konstrastive Phänomene und Kodierung

Die vorliegende Arbeit wurde im Rahmen des Verbundvorhabens Verbmobil vom Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie (BMBF) unter dem Förderkennzeichen 01 IV 701 N3 gefördert. Die Verantwortung für den Inhalt dieser Arbeit liegt bei der Autorin.

ISSN 0949-6084

Abstract

Transfer in Verb*mobil* is chiefly semantic-based. It transforms source into target language semantic representations by applying context sensitive transfer rules. Since the semantic representations abstract away from language-specific morpho-syntactic peculiarities, they allow for transfer rules to be specified in a rather compact way. However, the complexity of the semantic representation language used in Verb*mobil* poses certain difficulties to the rule writer if the source structure cannot be preserved in the target language. To simplify the writing of transfer rules and to cope with classes of translation patterns in a systematic way, *templates for transfer rules* have been developed. Templates turned out to be also advantageous in the way they allow systematic adaptation of changes in the semantic representations.

This report outlines the architecture of the transfer module, provides the definitions of templates and illustrates their usage with examples. It also presents the types of contextual restrictions used for disambiguation and gives an overview of the translation patterns that occur in the Verb*mobil* domain.[1]

[1]The work presented here is strongly influenced by discussions with my colleagues of the Verb*mobil* project. I would like to express my gratitude to Anke Lüdeling, Rosemary Vaughan, Michael Dorna, Martin Emele and Michael Schiehlen for most valuable comments on the topics of this report.

Contents

1 Introduction

Verb*mobil* is a real-time speech-to-speech translation system which is applied to the task of translating spoken language in the domain of appointment scheduling and travel planning (for an overview of the Verb*mobil* scenario see [Bub et al., 1997]). Currently, the system includes German, English and Japanese.

The Verb*mobil* system combines different MT approaches in a multi-engine MT technology [Nirenburg, 1995]. The combination of various translation engines ensures that, in most cases, a translated speech output can be produced. Another reason for this architecture is the interest in comparing the quality and performance of different approaches to speech-to-speech translation. The three main engines of the Verb*mobil* system are semantic-based transfer [Dorna and Emele, 1996b], statistical translation [Tillman et al., 1997] and example-based translation [Kroner and Schwinn, 1997] (see Figure 1). The common base for the three translation engines is a word lattice produced by a speech recognizer which is enriched with prosodic information.

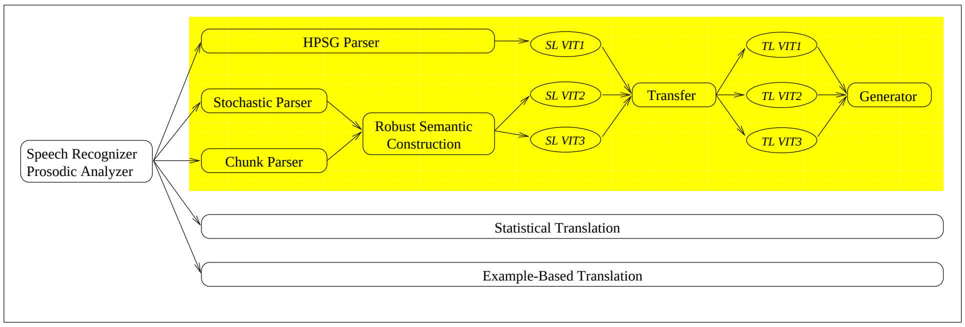

Figure 1: Combination of Different MT approaches in Verb*mobil*

Consider the transfer module in Figure 1. Its input is provided by various modules. Firstly, there is a deep semantic representation provided by an HPSG grammar that integrates semantic processing [Copestake et al., 1995]. Since spoken language is highly elliptical and often ungrammatical, its processing requires less exact methods. Hence, Verb*mobil* integrates two shallow processing modules: a cascaded chunk parser [Abney, 1996], [Light, 1996], and a statistical parser [Block and Ruland, 1998]. They produce (partial) syntactic structures from which a robust semantic module [Schiehlen, 1998] constructs semantic representations.

It is the task of the transfer module to provide a translation for all input representations regardless of their producer and their quality. This requires, on the one hand, that all analysis results are encoded in a common interface format, see Section 2. On the other hand, transfer should be very robust to serve the various inputs. Robustness is achieved by dynamically relaxing transfer rules that expect a correct input (in terms of a given linguistic specification) [Buschbeck-Wolf and Dorna, 1998].

This report is structured as follows: Section 2 outlines the semantic representations that are processed by the transfer. The internal architecture of the transfer module is described in Section 3. Section 4 introduces templates that are used for transfer mappings and illustrates them by examples. Finally, in Section 5, contextual restrictions and templates for most frequently used conditions are presented.

2 Semantic Representation

The results of the semantic analysis are encoded in a common format called Verb*mobil* Interface Term (henceforth VIT). For a more detailed description of VITs and the encoding of particular semantic analysis, I refer to [Bos et al., 1998], [Kasper et al., 1997], or [Dorna, 1998b]. A VIT is also used as input to the generator [Becker et al., 1997]. Figure 2 shows the Prolog encoding of a VIT for the German utterance in (1).

(1) Ginge dieser Termin vielleicht?
Would this time be possible?

```
vit(vitID(sid(102,a,de,0,4,1,de,y,synsem),
          [word(ginge,1,[l1]),
           word(dieser,2,[l2]),
           word(Termin,3,[l3]),
           word(vielleicht,4,[l4])]),
    index(l5,l6,i1),             % INDEX
   [quest(l5,h1),                % SEMANTICS
    vielleicht(l4,h2),
    gehen(l1,i1),
    arg3(l1,i1,i2),
    dies(l2,i2,l7,h3),
    termin(l3,i2)],
   [in_group(l1,l6),             % SCOPAL
    in_group(l3,l7),             % CONSTRAINTS
    in_group(l4,l8),
    leq(l8,h1),
    leq(l6,h2),
    leq(l6,h3)],
   [s_sort(i1,mental_sit),       % SORTS
    s_sort(i2,temporal),
    s_class(l4,sa)
    disc_func(l4,smoothen)],
   [dialog_act(l5,request),      % DISCOURSE
    dialog_phase(l5,negotiation)],
   [num(i2,sg)],                 % MORPHO-SYNTAX
   [ta_tense(i1,pres),           % TENSE & ASPECT
    ta_mood(i1,conj),
    ta_perf(i1,nonperf)],
   [pros_accent(l2),             % PROSODY
    pros_mood(l5,quest)])
```

Figure 2: VIT for Example (1)

A VIT is a record-like data structure whose fields are filled with semantic, scopal, sortal, morpho-syntactic, prosodic, discourse and other types of information. These slots can be seen as analysis layers collecting information of different linguistic nature that is produced by several modules. The information within and between the layers is linked by using constant symbols, called "labels", "instances" and "holes", see below.

In addition to pure linguistic information, a VIT contains a unique identifier that encodes the time span of the analyzed speech input, the analyzed path of the original word lattice, the producer of the VIT, the considered language, etc. This ID is used, for example, to synchronize the processing of different analyses.

The most important information for the transfer are the semantic predicates in the semantics slot. Each predicate has a unique label, `l`i, as its first argument. Referential predicates introduce an instance variable, `i`i, that binds arguments and modifiers. Additional semantic information is provided in the constraint slot. There are "groupings", `in_group(l`i`,l`j`)`, and subordination statements, `leq(l`i`,h`j`)`, for representing scope, coordination and propositional embedding in an underspecified way [Bos, 1996]. The method of grouping provides labels to address possible scope domains as a single unit.[2] Scope bearing predicates provide, besides a label, a hole variable, `h`i, for their underspecified scope which is constrained by `leq` statements. They describe indirect (less than) or direct (equal) subordination relations between hole variables and group labels.

3 Transfer Architecture

The transfer employs a two-level approach, see Figure 3. A refinement step precedes the actual translation mapping. The primary knowledge bases for transfer are a set of monolingual refinement and restructuring rules and the database of bilingual transfer rules.

There are several motivations for a refinement step within the transfer module. It is well known that the more abstract the representation to be transferred, the easier the mapping. Hence, the refinement module produces a partially language-neutral representation in order to keep the distance between the two languages small. To obtain a refined representation bilingual abstractions for contextually synonymous predicates are used, see Section 3.3. Ambiguities of preposition are resolved by mapping them to meaning abstractions. Complex predicates are and decomposed into language-independent semantic primitives [Buschbeck-Wolf and Tschernitschek, 1996]. The information encoded in the abstract predicate can be used for the resolution of translational ambiguities in the actual transfer mapping. It allows for a compact representation of contextual restrictions.

Since all restructuring and refinement operations are motivated by contrastive data, I assume this functionality to be part of the transfer process. This way, the modularity of the grammar components can be maintained [Kay et al., 1994].

Consider a simple example in (2).

[2]Groupings can be seen as some sort of UDRS boxes [Reyle, 1993].

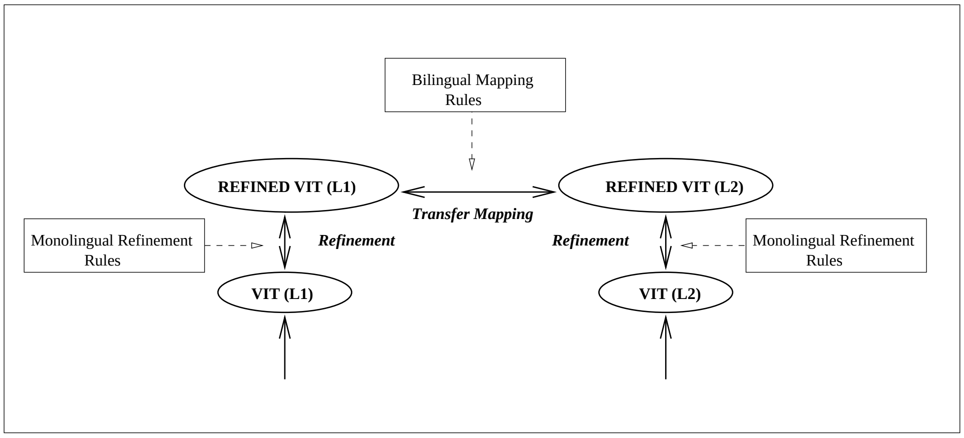

Figure 3: Two-Level Transfer Module

(2) Ich komme *am* Montag/*im* Mai/*um* neun/*zu* Ostern.
I'll come *on* Monday/*in* May/*at* nine o'clock/*at* Easter

German and English use different prepositions to express temporal localization. Occurring with time expressions, such as *Monday, May, nine o'clock* or *Easter*, the use of German *in, um, an, zu*, or English *in, on, at* is rather idiosyncratic. The encoding of translation mappings between these prepositions would lead to a large number of rules that hold only in very specific contexts. This can be avoided by bundling the German and English prepositions wrt. their meaning in the refinement step and considering the produced abstraction as an interlingual predicate, such as `temp_loc` for temporal localization in Figure 4.

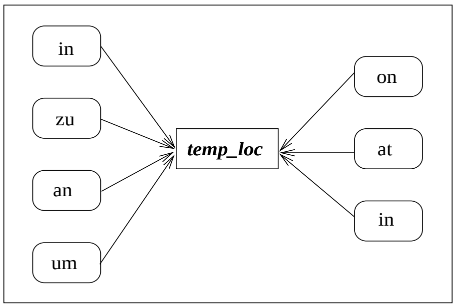

Figure 4: Abstraction over Contextually Synonymous Expressions

The use of lexical abstraction and decomposition in the refinement step reduces the number of transfer rules to the necessary minimum. The output of the transfer is a partially language-neutral representation that allows the generator to produce paraphrases.

3.1 Transfer Rules

The transfer relates underspecified source language (SL) semantic descriptions and target language (TL) semantic descriptions by the application of bilingual transfer rules.

The general form of a transfer rule is shown in (3). It establishes the equivalence between sets of SL semantic predicates (`SL_Sem`) and sets of TL semantic predicates (`TL_Sem`) if certain conditions (`SL_Cond` and `TL_Cond`) are fulfilled. The operator `TauOp` indicates in which direction a rule is applied, i.e., bi-directional ($\leftrightarrow$) or uni-directional ($\rightarrow$ or $\leftarrow$). The particular members `SL_Sem`, `SL_Cond`, `TL_Sem` and `TL_Cond` are conjoined by commas. A period indicates the end of the rule.[3]

(3) `SL_Sem # SL_Cond TauOp TL_Sem # TL_Cond.`

In general, transfer rules are specified bidirectionally, except for cases in which backward translation is not desired, such as for rules that remove redundant information from the input utterance. In the transfer mapping module, `SL_Sem` and `SL_Cond` are *always* considered to be German, and `TL_Sem` and `TL_Cond` to be English.

(4) shows a simple transfer rule for the unambiguous semantic predicates *wollen* and *want*. The capitalized symbols `L` and `I` stand for logical variables which are bound to concrete values when applying a rule to a given input. If a variable is not bound within the rule, an anonymous variable is used. It is indicated by an underscore.

(4) `wollen(L,I) <-> want(L,I).`

To guarantee lexical consistency, all semantic predicates that form part of the rule must be compatible with the semantic lexicons, i.e., elements of `SL_Sem` should be contained in the German semantic lexicon, and elements of `TL_Sem` in the English semantic lexicon.

To resolve translational ambiguities, the rules are optionally provided with a condition part (`SL_Cond` and `TL_Cond`) that restricts the transfer mappings to the relevant context. The `#` sign separates it from the mapping part. The condition to the left of `TauOp` restricts the application direction $\rightarrow$, and vice versa. The condition part contains only tests. On the one hand, these are tests on certain semantic predicates, on their sort, their semantic class, etc., i.e., tests on information that is contained in the actual VIT representation, see Section 5.1. On the other hand, these are interface predicates that encode calls to other components for providing information that is not part of the actual VIT, see Section 5.2. Conditions can also be combined disjunctively by the use of a semicolon. For syntactic issuses I refer to [Dorna, 1998a].

Consider the ambiguous verb *verlegen*, which is translated, among others, into *put off*, *publish*, *move* and *misplace*. The different readings can be identified by sortal restrictions on the theme argument (`arg3`). The rule that is relevant for the translation of *verlegen* into *put off* is shown in (5). It is constrained to contexts in which the theme argument is realized by a noun of the sort `situation`.

[3] A rule application reduces the SL input by the set of semantic predicates in `SL_Sem` if they form a matching subset of the input. On the other hand, the TL semantic predicates `TL_Sem` are added to the TL output (see [Dorna and Emele, 1996a] for details).

(5)
```
verlegen(L,I) # arg3(L,I,I1),sort_check(I1,situation)
<-> put_off(L,I).
```

The application of rules is guided by a *specificity principle* which takes into account the number of predicates in `SL_Sem`, the number of restrictions in `SL_Cond` and their specificity. The most complex rule is given priority over competing rules. Additionally, a transfer rule compiler partially orders the sets `SL_Sem` and `SL_Cond`. This makes matching on a given sorted SL input set more efficient.[4]

3.2 Refinement Rules

Monolingual refinement rules (6) follow the same pattern as transfer rules in (3). Here, the translation direction is fixed and the conditions are specified only on the SL side.

(6)
```
SL_Sem # SL_Cond → SL_RefinedSem.
```

(7) shows the mapping of the prepositions *in, um, an* and *zu* to the abstraction `temp_loc`, see example (2). `stat_temp_prep` is an *abstract predicate type* that allows these prepositions to be mapped in one step. Its declaration is shown in (12) in Section 3.3. The refinement rule in (7) is constrained by the sortal restriction `time` on the internal argument of these prepositions with the instance variable `I1`.

(7)
```
stat_temp_prep(L,I,I1) # sort_check(I1,time) -> temp_loc(L,I,I1).
```

Note that in the refinement module, `SL_Sem` is considered to be German for German refinement rules, and English for English refinement rules.

3.3 Abstract Predicate Types

Abstract predicate types are, on the one hand, meaning abstractions or concepts that bundle synonymous lexicalizations in the SL and the TL. They allow semantic predicates to be transferred as a whole class, and thus, move up the mapping level. On the other hand, they are used to group predicates wrt. a certain property they have in common, e.g., in order to formulate contextual restrictions compactly, see Section 5.

(8)
```
type(AbstrPred,[Preds]).
```

Predicate types are declared by the definition in (8), where `AbstrPred` is the name of the abstract predicate and `Preds` is a list of predicates that are either synonymous or that have some feature in common. The definitions of predicate types for each language are stored in a separate file. By using types in the definition of other abstract types, it is also possible to construct a hierarchy (see [Buschbeck-Wolf and Tschernitschek, 1996]).

(9) shows some German intensifiers with an approximating meaning. They are grouped together to the type `approx_grad` by the type definition in (10).

[4]For further compilation details, see [Dorna and Emele, 1996a].

(9) Es kommen *etwa/ungefähr/so/zirka* 10 Leute.
Around 10 people are coming.

(10) `type(approx_grad,[etwa,ungefaehr,so,zirka]).`

Used in a transfer rule, such as in (11), the abstract type is replaced by the predicates belonging to it; i.e., the transfer rule is multiplied for each predicate.

(11) `approx_grad(L,I) <-> approx_grad(L,I).`

Abstractions are introduced for all kinds of contextually synonymous expressions, and for prepositions that are disambiguated by the mapping to meaning relations, see Section 3.2. (12) shows the definition of the type for temporal stative prepositions that is used by the rule in (7).

(12) `type(stat_temp_prep,[in,um,an,zu]).`

4 Templates for Transfer Rules

For building the transfer rule bases, I introduced a template mechanism that ensures the adaptation and reusability of transfer rules independent of the concrete front and back end of the transfer component. Templates simplify the writing of transfer rules considerably, and allow us to capture translation patterns in a systematic way, see [Buschbeck-Wolf and Dorna, 1997].

There are two types of templates:

1. Templates for transfer equivalences (Section 4.1 - Section 4.3)
(the template call is prefixed by the `@` operator)
2. Templates for contextual constraints (Section 5.3)
(the template call is prefixed by the `+` operator)

To restrict the application of templates to the relevant predicate classes and to prevent errors in the template usage, the template definitions contain type checks. They are shown in Table 1.

In the following sections, the template definitions are presented. They are subdivided into three main groups:

1. Templates for regular mappings
2. Templates for deletions and insertions
3. Templates for categorial changes

In most cases, I provide two variants of the same template, one with and one without variables for instances, labels, and holes. The former is used whenever contextual restrictions or other templates should be attached. The latter is used for default rules.

To ensure the rule writer a maximal degree of flexibility, rules can be written in various ways: with or without templates, or in a combination of both variants,

Type Name	Predicate Class
verb	verbal predicate
mod	adverbial and adjectival predicate
noun	nominal predicate
prep	prepositional predicate
quant	quantifier
scop	scopal adverb
grad	intensifier
subord	subordination
coord	coordination
arg	arg1,arg2,arg3
deg	comp, superl

Table 1: Types in Template Definitions

depending on the regularity of the translation pattern. Templates can be combined with each other by using at least one variable in common. They are chained together without any commas. A period after the last template indicates the end of the rule. Note that if the period is missing the template is attached to the next one occurring in the file.

In general, templates expand to bidirectional rules. If we want to suppress an undesirable back translation, the rule application has to be restricted by using a unidirectional tau operator, i.e., $\rightarrow$ or $\leftarrow$ between the SL and TL condition parts. If there are no conditions on a mapping but the direction of its application should be determined we use the corresponding tau operator between two `true` predicates. An example is given in (13). The adjective *geschickt* is translated into *good*. It is not translated back, since there is another regular translation for *good* into German. Similarly, *crucial* is mapped to *wichtig* but not the other way round. The `mod` template is introduced in Section 4.1.3.

(13) `true -> true @mod(geschickt,good).` German to English only
`true <- true @mod(wichtig,crucial).` English to German only

Below, I introduce the different template types and provide examples for each type. Some templates are used only as auxiliary building blocks in other templates.

4.1 Templates for Regular Mappings

Templates do not really simplify the rule writing for regular one-to-one mappings, nethertheless, I provide them in order to make the combination of different templates more convenient, and to have a basis for systematic adaptation of changes in the semantic representations.

The template names indicate the semantic class of the predicates to be translated. The variables SL and TL stand for the source and target language words that are translation correspondences. The default template is derived from the template with the full number of variables of instances, labels and holes.

4.1.1 Templates for the Regular Mapping of Nouns

In (14), you find the definition of templates for the translation between nouns. The template name is `noun`. The template head in (14a) has four arguments, i.e., a SL predicate of the type noun, a TL predicate of the type noun, and their label and instance variables `L` and `I`. (14b) shows the template definition for the default mapping of nouns.

(14a) `noun(noun(SL),noun(TL),L,I) := SL(L,I) <-> TL(L,I).`

(14b) `noun(SL,TL) := @noun(SL,TL,_,_).`

The application of (14a) is shown in (16) with the translation of *Umstände* into *trouble*, see (15). It is valid only if the German plural noun is used as theme argument (`arg3`) of the verb *machen*. This condition is expressed in the first line of the rule in (16) after the `#` sign. The number value has to be changed as well.

(17) shows its application of (14a) for the default mapping of *Frage* to *question*, and vice versa. For this translation, no conditions are stated.

(15) Das macht keine *Umstände* für mich.
This doesn't cause any *trouble* to me.

(16)
```
num(I,pl) # machen(L1,I1),arg3(L1,I1,I) -> num(I,sg)
@noun(umstand,trouble,_,I).
```

(17) `@noun(frage,question).`

Proper names are not translated with the exception of certain place and holiday names. For these cases, I provide the templates in (18) and (19), respectively. (20) and (21) give examples of their usage.

(18) `named(SL,TL) := named(L,I,SL) <-> named(L,I,TL).`

(19) `holiday(SL,TL) := holiday(L,I,SL) <-> holiday(L,I,TL).`

(20) `@named(koeln,cologne).`

(21) `@holiday(karfreitag,good_friday).`

4.1.2 Templates for the Regular Mapping of Verbs

In (22), you find the definition of templates for the translation between verbs. The template name is `verb`.

(22a) `verb(verb(SL),verb(TL),L,I) := SL(L,I) <-> TL(L,I).`

(22b) `verb(SL,TL) := @verb(SL,TL,_,_).`

(24) exemplifies the application of (22a) for the translation of *etwas bearbeiten* into *to work on something*, see (23). Here, the theme changes its category from a verb argument in German to an argument of a preposition in English. This change is

managed by the template `argTOprep` that is attached by the coindexation of the variables `L` and `I`. It will be introduced in Section 4.3.2. (25) shows the application of (22b) for the default translation of *ändern* into *change*.

(23) Könnten Sie *das bearbeiten*?
Could you *work on that*?

(24)
```
@verb(bearbeiten,work_v,L,I)
@argTOprep(arg3,[on_prep,no],L,I,_,_).
```

(25)
```
@verb(aendern,change).
```

Modal verbs have their own template called `modal` since they have one more argument than regular verbs, see (26).

(26a)
```
modal(verb(SL),verb(TL),L,I,H) := SL(L,I,H) <-> TL(L,I,H).
```

(26b)
```
modal(SL,TL) := @modal(SL,TL,_,_,_).
```

(28) shows the use of the modal verb template in (26a). The mapping of *dürfen* into *may*, see (27), is constrained to the context of questions. This is what `sent_mood(L,quest)` tests for, see (277) in Section 5.1.5. The restriction is attached to the template via the coindexation of the label `L`. (29) shows the application of (26b) for the default translation of *können* into *can*.

(27) *Dürfte* ich Ihnen eine Frage stellen?
May I ask you a question?

(28)
```
# sent_mood(L,quest) <-> true
@modal(duerfen,may,L,_,_).
```

(29)
```
@modal(koennen,can).
```

4.1.3 Templates for the Regular Mapping of Modifiers

(30) shows the definition of templates for the translation of intersective adverbial and adjectival modifiers. The template name is `mod`.

(30a)
```
mod(mod(SL),mod(TL),L,I) := SL(L,I) <-> TL(L,I).
```

(30b)
```
mod(SL,TL) := @mod(SL,TL,_,_).
```

(32) shows the use of the template in (30a) for the translation of *da* into *there*, see (31). The mapping in (32) is constrained to the predicative usage of the deictic adjective *da*. Whether the adjective occurs as predicative is tested by the predicate `stat(L,_)` in the condition part. It has the same label `L` as the adverb. (33) illustrates the application of (30b) for the default translation of *da* into *then*.

(31) Am Montag bin ich *da*.
I'll be *there* on Monday.

(32)
```
# stat(L,_) <-> true
@mod(da_isect,there,L,_).
```

(33)
```
true -> true
@mod(da_isect,then).
```

4.1.4 Templates for the Regular Mapping of Prepositions

(34) shows the definition of templates for the transfer of prepositions. The template name is `prep`.

(34a) `prep(prep(SL),prep(TL),L,I1,I2) := SL(L,I1,I2) <-> TL(L,I1,I2).`

(34b) `prep(SL,TL) := @prep(SL,TL,_,_,_).`

(36) shows the application of (34a) for the translation of *auf* into *in*, see (35). It is restricted to contexts in which the internal argument of these prepositions (indicated by the instance variable `I`) refers to a language. `abstr_language` is a predicate type that covers language names. (37) uses the template in (34b) for the unambiguous mapping of *zwischen* to *between*.

(35) Wie heißt das *auf* Englisch?
How is this called *in* English?

(36)
```
# abstr_language(_,I) <-> # abstr_language(_,I)
@prep(auf,in,_,_,I).
```

(37) `@prep(zwischen,between).`

4.1.5 Templates for the Regular Mapping of Intensifiers

In (38), you find the definition of templates for the translation between intensifiers. The template name is `grad` for *gradual*. The default variant in (38b) does not include the dimensionality predicate `dim` because it is not always assigned in the analysis.

(38a)
```
grad(mod(SL),mod(TL),L,I,L1) :=
        SL(L,I) # dim(L1,I) <-> TL(L,I) # dim(L1,I).
```

(38b) `grad(SL,TL,L,I) := SL(L,I) <-> TL(L,I).`

(40) and (41) use the templates in (38a) and (38b), respectively. If the intensifier *besonders* is preceded by the negation particle *nicht*, it is translated into *very*, see (39). This restriction is expressed in the condition part of (40) by the adjacency constraint `adjac(L,L1)`. It says that the words with the labels `L` and `L1` should be adjacent on the surface, see Section 5.1.6. In general, *besonders* is mapped to the abstract type `special_grad` that includes the nearly synonymous intensifiers *particularly* and *especially*.

(39) Das paßt mir nicht *besonders* gut.
This doesn't suit me *very* well.

(40)
```
# neg(L,_),adjac(L,L1) -> true
@grad(besonders_grad,very,L1,_,_).
```

(41) `@grad(besonders_grad,special_grad).`

4.1.6 Templates for the Regular Mapping of Quantifiers

(42) shows the definition of templates for the translation of quantifiers. The template name is `quant`.

(42a) `quant(quant(SL),quant(TL),L,I,L1,H) := SL(L,I,L1,H) <-> TL(L,I,L1,H).`

(42b) `quant(SL,TL) := @quant(SL,TL,_,_,_,_).`

(44) and (45) illustrate the use of the templates in (42a) and (42b), respectively. If the noun *Zeit* is in the restrictor of the quantifier *jeder* it is translated into *any*, see (43). The corresponding rule is shown in (44). Otherwise, it is assigned the default translation *every* in (45).

(43) Mir ist *jede* Zeit recht.
Any time is fine with me.

(44)
```
# zeit(_,I) <-> # time(_,I)
@quant(jed,any,_,I,_,_).
```

(45) `@quant(jed,every).`

4.1.7 Templates for the Regular Mapping of Scopal Adverbs

In (46), the definition of templates for the translation between scopal adverbs is given. The template name is `scop`.

(46a) `scop(scop(SL),scop(TL),L,H) := SL(L,H) <-> TL(L,H).`

(46b) `scop(SL,TL) := @scop(SL,TL,_,_).`

(48) and (49) show the transfer rules for *immer* that use the templates in (46a) and (46b), respectively. If *immer* occurs in combination with the particle *noch*, see (47), it is translated into *still*. Here, *noch* has to be deleted. This is done by the template `lhs_scop`, see (89) in Section 4.2.5. The condition part of (48) contains the restriction that *immer* and *noch* occur directly adjacent. The default translation of *immer* - *always* is assigned in (49).

(47) Wir haben *immer noch* keinen Termin für unser Treffen.
We *still* don't have a time for our meeting.

(48)
```
# adjac(L,L1) -> true
@scop(immer_scop,still,L,_)
@lhs_scop(noch_scop,L1,_,_).
```

(49) `@scop(immer_scop,always).`

4.1.8 Templates for the Regular Mapping of Conjunctions

Because of their different representations, coordinating and subordination conjunctions have their own templates, see (50) and (51). They are called `coord` and `subord`, respectively.

(50a) `coord(coord(SL),coord(TL),L,I,H1,H2) := SL(L,I,H1,H2) <-> TL(L,I,H1,H2).`

(50b) `coord(SL,TL) := @coord(SL,TL,_,_,_,_).`

(51a) `subord(subord(SL),subord(TL),L,H1,H2):= SL(L,H1,H2) <-> TL(L,H1,H2).`

(51b) `subord(SL,TL) := @subord(SL,TL,_,_,_).`

For the transfer of conjunctions, only the default templates in (50b) and (51b) have been used so far. Examples for the translation between *sondern* and *but*, and *wenn* and *if* are shown in (52) and (53).

(52) `@coord(sondern,but_coord).`

(53) `@subord(wenn,if).`

4.1.9 Templates for the Regular Mapping of Exclamatives

Finally, (54) shows the definition of templates for the translation of exclamatives. The template name is `excl`.

(54a) `excl(SL,TL) := excl(L,SL) <-> excl(L,TL).`

(54b) `excl(SL,TL) := @excl(SL,TL,_).`

(55) shows the usage of the template in (54b) to map the English exclamative *yep* to German *ja*. The direction of the template application is restricted by the tau operator to block the translation from German to English.

(55) `true <- true @excl(ja,yep).`

4.2 Templates for Insertions and Deletions

Deletion and insertion of language material is a common operation in translation. For various reasons, languages differ in the way complex meanings are built from elementary units. For example, negation, mood, direction, causativity, might be contained in the meaning of a word in one language but not in the other, where it has to be lexicalized by separate words.

In addition to this general phenomenon, in Verb*mobil* there is a particular need for deletion operations because we are dealing with spoken language which is rather redundant because of the speaker's hesitations and repetitions. To produce an acceptable translation, it is desirable to decrease the degree of redundancy of the input by removing redundant material during transfer. This concerns especially the treatment of particles. Compared with German, English uses less discourse particles and filler words. Therefore, particles are often dropped in order to achieve a stylistically well-formed English translation.

Templates for insertion and deletion operations are built up in a way that they can be used recursively, i.e., first, I define elementary templates, which I then use to define more complex ones. This is the reason why some of the elementary insertion and deletion templates are not used for the actual translation mappings. They exist only for the purpose of defining other templates.

Insertion and deletion templates are prefixed with `lhs` and `rhs`; `lhs` means that the predicates under consideration are on the rule's left hand side, and `rhs` – that they are on the rule's right hand side. Given the convention that German is on the left hand side of the tau operator, and English on the right hand side, `lhs`- and `rhs`-prefixed templates are interpreted as shown in Table 2.

Prefix	Direction	Action
`rhs_*`	German → English	add information to English output
`rhs_*`	English → German	remove information from English input
`lhs_*`	German → English	remove information from German input
`lhs_*`	English → German	add information to German output

Table 2: Interpretation of Insertion and Deletion Prefixes

4.2.1 Insertions and Deletions of Pronouns

In most cases, pronouns are inserted or deleted either as an argument of a verb, see (56), or as an argument of a preposition, see (57). In order to define templates that insert or remove pronouns in these functions, the auxiliary templates in (58) and (59) are provided.

(56) Wir sehen *uns* am Montag.
We'll meet on Monday.

(57) Ich würde das jetzt gern *hinter mich* bringen.
I would like to get that over with.

Given the translation direction German-English, the template in (58) inserts a pronoun in the English output, and (59) removes it from the German input. With the reverse transfer direction, it works the other way round. The template head contains only information that cannot be predicted, i.e., the instance variable of the pronoun for attaching other templates or conditions, and the values of the pronoun type (`prontype`) designators `D1` and `D2` that specify the type of the considered pronoun and its number. We also have to take care of the group label `G` of the pronoun that has to be instantiated when applying these templates. It depends on the kind of embedding of the pronoun. If the pronoun is the internal argument of a preposition, its group label is the group label of the preposition, if it is the subject of the copula, it corresponds to the group label of `stat`, and if it is bound by an argument role, its group label is that of the verb.

(58)
```
rhs_pron(I,D1,D2,Num,G) :=
        true <-> pron(L,I),prontype(I,D1,D2),num(I,Num),in_g(L,G).
```

(59)
```
lhs_pron(I,D1,D2,Num,G):=
        pron(L,I),prontype(I,D1,D2),num(I,Num),in_g(L,G) <-> true.
```

Template head:		
	`I`	Instance variable of `pron`
	`D1`	Reference designator of `pron`
	`D2`	Sort designator of `pron`
	`Num`	Number of `pron`
	`G`	Group label of `pron`

(58) and (59) are applied for the definition of more complex deletion and insertion templates, see Sections 4.2.6 and 4.2.7.

4.2.2 Insertions and Deletions of NPs

The insertion and deletion of NPs is a rather common operation in translation. In most cases, NPs are added or removed in a particular function, i.e., as arguments of verbs, nouns or adjectives, see (60), or as internal arguments of prepositions, see (61). For these kinds of operations (they are presented in Sections 4.2.6 and 4.2.7), we need the auxiliary templates in (62) and (63).

(60) Wo werden Sie *übernachten*?
Where do you *stay the night*?

(61) Sollen wir dieses Wochenende *ins Auge fassen*?
Should we *consider* this weekend?

(62) shows the template for inserting an NP in the TL, and (63) for its deletion in the SL. In the template head, the quantifier, the noun and its number are provided. When inserting an NP, the correct semantic embedding of the quantifier is problematic. On the one hand, the scope domain `G2` of the quantifier has to be entered when applying the template. It corresponds to the group label of the predicate of which the NP is an argument. On the other hand, the group label of the introduced quantifier `G1` has to be bound by a scopal constraint (`leq(G1,H1)`). Since all possible variants of the context above the quantifier cannot be considered within a transfer rule, it is put under the top hole `H1`. It is accessed by the `top_hole` constraint that delivers the top hole variable of a given predicate.

(62)
```
rhs_np(quant(Qua),noun(Noun),Num,L1,I,G2) :=
      # top_hole(Qua,G2,H1) <-> Qua(L,I,G,H),Noun(L1,I),num(I,Num),
      in_g(L1,G),in_g(L,G1),leq(G1,H1),leq(G2,H).
```

(63)
```
lhs_np(quant(Qua),noun(Noun),Num,L1,I,G2) :=
      Qua(L,I,G,H),Noun(L1,I),num(I,Num),in_g(L1,G),in_g(L,G1),
      leq(G1,H1),leq(G2,H) <-> # top_hole(Qua,G2,H1).
```

Template head:	`Qua`	Quantifier of `Noun`
	`Noun`	Nominal predicate
	`Num`	Number of `Noun`
	`L1`	Label of `Noun`
	`I`	Instance variable of `Noun`
	`G2`	Scope of `Qua`

It is also useful to provide a template for *adding and deleting NPs that are modified by an adjective*, see (64) and (65).

(64) Könnten Sie das bitte *konkretisieren*?
Could you please *put* it *in concrete terms*?

(65) Wir müssen ganz *kurzfristig* einen Termin vereinbaren.
We have to make an appointment *at short notice*.

The templates that add and remove modified NPs are shown in (66) and (67). They differ from those in (62) and (63) only in so far as they have an additional variable for entering the adjective in the template head. These templates are again used for the definition of more complex ones, see Sections 4.2.6 and 4.3.5.

(66)
```
rhs_mod_np(quant(Qua),mod(Adj),noun(Noun),Num,L1,I,G2) :=
     # top_hole(Qua,G2,H1) <->
     Qua(L,I,G,H),Noun(L1,I),Adj(L2,I),num(I,Num),in_g(L1,G),
     in_g(L2,G),in_g(L,G1),leq(G1,H1),leq(G2,H).
```

(67)
```
lhs_mod_np(quant(Qua),mod(Adj),noun(Noun),Num,L1,I,G2) :=
     Qua(L,I,G,H),Noun(L1,I),Adj(L2,I),num(I,Num),in_g(L1,G),in_g(L2,G),
     in_g(L,G1),leq(G1,H1),leq(G2,H) <-> # top_hole(Qua,G2,H1).
```

Template head:	`Qua`	Quantifier of `Noun`
	`Adj`	Modifier of `Noun`
	`Noun`	Nominal predicate
	`Num`	Number of `Noun`
	`L1`	Label of `Noun`
	`I`	Instance variable of `Noun`
	`G2`	Scope of `Qua`

For some of the nouns, the semantic analysis provides three-place interlingual predicates. They have a label, an instance variable and a designator `D` that stores the relevant information. For days of the week, it is `dofw(L,I,D)`, for months of the year `mofy(L,I,D)`, for proper names `named(L,I,D)` and for holiday names in particular `hd(L,I,D)`. According to their nature, these predicates are not objects of insertion or deletion. However, they might occur as parts of German compounds, see (68), and for more examples Section 4.3.8. For decomposing these kinds of compounds, we need an insertion/deletion template for three-place nouns. They are defined in (69) and (70). Concerning the correct semantic embedding, they follow those in (62) and (63). They differ only in the representation of the nominal predicate.

(68) Wie sieht es in der zweiten *Maihälfte* aus?
How about the second *half of May*?

(69)
```
rhs_np3(quant(Qua),Noun3,D,Num,L1,I,G2) :=
     # top_hole(Qua,G2,H1) <-> Qua(L,I,G,H),Noun3(L1,I,D),
     num(I,Num),in_g(L1,G),in_g(L,G1),leq(G1,H1),leq(G2,H).
```

(70)
```
lhs_np3(quant(Qua),Noun3,D,Num,L1,I,G2) :=
     Qua(L,I,G,H),Noun3(L1,I,D),num(I,Num),in_g(L1,G),
     in_g(L,G1),leq(G1,H1),leq(G2,H) <-> # top_hole(Qua,G2,H1).
```

Template head:	`Qua`	Quantifier of `Noun3`
	`Noun3`	Interlingual nominal predicate
	`D`	Designator of `Noun3`
	`Num`	Number of `Noun3`
	`L1`	Label of the `Noun3`
	`I`	Instance variable of `Noun3`
	`G2`	Scope of `Qua`

4.2.3 Insertions and Deletions of Adjectival and Adverbial Modifiers

Consider the example in (71). When translating *falsch verstehen* into English *misunderstand* we have to remove the German adverb, the meaning of which is conveyed by a prefix in English. With the reverse translation direction, *falsch* has to be inserted as a verb modifier.

(71) Da haben Sie mich *falsch verstanden*.
You *misunderstood* me.

For adding and deleting modifiers, the templates in (72) and (73) are used. Here, we have to take care of the instance variable of the modifier, which has to be coindexed with the instance variable of the modified predicate, and of its group label `G`. It corresponds to the group label of the modified predicate that has to be instantiated when applying the template. This is shown with the rule in (74) that captures the example in (71).

(72)
```
rhs_adx(mod(Adx),L,I,G) := true <-> Adx(L,I),in_g(L,G).
```

(73)
```
lhs_adx(mod(Adx),L,I,G) := Adx(L,I),in_g(L,G) <-> true.
```

Template head:	`Adx`	Adjectival or adverbial modifier
	`L`	Label of `Adx`
	`I`	Instance variable of `Adx`
	`G`	Group label of `Adx`

(74)
```
true <-> # in_g(L,G)
@verb(verstehen,misunderstand,L,I)
@lhs_adx(falsch,_,I,G).
```

After applying the rule in (74) we obtain the expanded rules in (75) and (76). Note that with the deletion of the modifier (German to English) its grouping is simply removed, while with its insertion (English to German), the appropriate group label is specified in the condition part.

(75)
```
falsch(A,B),verstehen(C,B),in_g(A,D) ->
        misunderstand(C,B).
```

(76)
```
misunderstand(A,B) # in_g(A,C) ->
        verstehen(A,B),falsch(D,B),in_g(D,C).
```

Sometimes we want to insert or delete modifiers in their comparative or superlative forms, as shown in (77).

(77) Könnten wir das einen Tag *vorverlegen*?
Could we *schedule* this one day *earlier*?

For this, the templates in (78) and (79) are provided. Here, the adjective together with the degree specification is entered in the template head.

(78)
```
rhs_deg_adx(Adx,CompSup,L,I,G) :=
        true <-> CompSup(L,I,_)
        @rhs_adx(Adx,L,I,G).
```

(79)
```
lhs_deg_adx(Adx,CompSup,L,I,G) :=
        CompSup(L,I,_) <-> true
        @lhs_adx(Adx,L,I,G).
```

Template head:	`Adx`	Adjectival or adverbial modifier
	`CompSup`	`comp` or `superl`
	`L`	Label of `Adx` and `CompSup`
	`I`	Instance variable of `Adx` and `CompSup`
	`G`	Group label of `Adx`

As an example, consider in (80) the rule for the verbs *vorziehen* and *vorverlegen* that are subsumed by the abstract type `backward_temp_move`. They are translated into *schedule earlier*, where *early* is a modifier of the verb (indicated by the coindexation of `I`). Its group label `G` is instantiated by the group label of the verb in the condition part.

(80)
```
# in_g(L,G) <-> true
@verb(backward_temp_move,schedule,L,I)
@rhs_deg_adx(early,comp,_,I,G).
```

(81) shows the expansion of the template in (80) for German-English, and (82) for English-German. In the TL, the abstract type `backward_temp_move` is not expanded. The lexical choice is left to the generator.

(81a)
```
vorverlegen(A,B) # in_g(A,C) ->
     schedule(A,B),comp(D,B,E),early(D,B),in_g(D,C).
```

(81b)
```
vorziehen_move(A,B) # in_g(A,C) ->
     schedule(A,B),comp(D,B,E),early(D,B),in_g(D,C).
```

(82)
```
early(A,B),schedule(C,B),comp(A,B,D),in_g(A,E) ->
     backward_temp_move(C,B).
```

4.2.4 Insertions and Deletions of Prepositions

In most cases, prepositions are inserted or deleted as whole PPs, see (83).

(83) Können wir das *beschließen*?
Could we *come to an agreement*?

To handle PP insertion and deletion, we need an auxiliary definition for adding and removing prepositions. They are presented in (84) and (85). These templates work essentially the same way as those for adjectival and adverbial modifiers, see Section 4.2.3. Since prepositions differ wrt. directionality (`YesNo`) it is necessary to enter this information each time; `yes` means that the preposition is directional; `no` means that the preposition is not directional. The group label `G` has to be instantiated when applying the template.

(84)
```
rhs_prep(prep(Prep),YesNo,L,I,I1,G) :=
     true <-> Prep(L,I,I1),dir(L,YesNo),in_g(L,G).
```

(85)
```
lhs_prep(prep(Prep),YesNo,L,I,I1,G) :=
     Prep(L,I,I1),dir(L,YesNo),in_g(L,G) <-> true.
```

Template head:	`Prep`	Preposition
	`YesNo`	Directionality of `Prep`
	`L`	Label of `Prep`
	`I`	External argument of `Prep`
	`I1`	Internal argument of `Prep`
	`G`	Group label of `Prep`

The templates in (84) - (85) are used, first of all, for templates that delete and insert PPs with nominal and pronominal arguments, see Section 4.2.7, as well as for changes of adverbial into prepositional modifiers, see Section 4.3.5.

4.2.5 Insertions and Deletions of Scopal Adverbs

In Verb*mobil* discourse particles and filler words are classified as scopal adverbs. For a semantic classification of scopal adverbs I refer to [Bos and Schiehlen, 1997]. In spoken German, discourse particles occur with high frequency without contributing much to the propositional content of the utterance. Since English does not allow as many discourse particles within the same utterance, we need a template that manages their deletion, see (86).

(86) Wie sieht es bei Ihnen *denn* Anfang Dezember aus?
How about the beginning of December?

But not all scopal adverbs can simply be deleted. There is need for bidirectional insertions/deletion operations as well. In (87), for example, the construction *erst wieder an X* is translated into *not before X*. Here, the combination of the two scopal adverbs *erst* and *wieder* is expressed in English by the negation and the transfer of the preposition changes.

(87) Ich habe *erst wieder am* Montag Zeit.
I'll have time *not before* Monday.

To comply with these requirements, the template definitions for adding and deleting scopal adverbs in (88) and (89) look somewhat different. The definition is splitted wrt. directionality. This allows the frequently used deletion templates in (88a) and (89a) to be formulated in a robust way by using the `nop` predicate. The deletion of a scope bearing predicate requires the label chain above and below it to be closed. Otherwise the VIT structure is interrupted and cannot be traced. To avoid this the `nop` predicate is introduced. It is an auxiliary predicate that equates the label with the hole variable, such that the gap gets closed. Since `nop` does not occur on the input the template definitions in (88a) and (89a) cover only the case of deletion.

The templates in (88b) and (89b) perform the restructuring of the scopal and grouping constraints when inserting a scopal adverb. In contrast to the (a) cases, their template head contains in addition to the label `L` and the hole variable `H` a variable `G` that corresponds to the scope domain of the adverb in question. `G` is usually the group label of the main verb of the utterance. It has to be instantiated when applying the template. The inserted scopal adverb has to be bound to the top, which means that we have to find the hole variable `H1` for its group label `G1`. `H1` is consider to be the top hole. In order to find it we can use again the `top_hole` request that was introduced in Section 4.2.2.

```
(88a)  rhs_scop(scop(Adv),L,H,_) := nop(L,H) <- Adv(L,H).
(88b)  rhs_scop(scop(Adv),L,H,G) := # top_hole(Adv,G,H1) ->
            Adv(L,H),leq(G,H),in_g(L,G1),leq(G1,H1).

(89a)  lhs_scop(scop(Adv),L,H,_) := Adv(L,H) -> nop(L,H).
(89b)  lhs_scop(scop(Adv),L,H,G) := Adv(L,H),leq(G,H),in_g(L,G1),leq(G1,H1)
            <- # top_hole(Adv,G,H1).
```

Template head:	`Adv`	Scopal adverb
	`L`	Label of `Adv`
	`G`	Group label that corresponds to the scope domain of `Adv`
	`H`	Scope of `Adv`

An example of the application of (89a) is given in (90). The deletion of *denn* from the German input is restricted to cases in which it occurs as a modal particle (`s_class(L,mp)`), see Section 5.1.4. The expanded rule is shown in (91).

(90)
```
# s_class(L,mp) -> true
@lhs_scop(denn,L,_,_).
```

(91)
```
denn(A,B) # s_class(A,mp) -> nop(A,B).
```

The rule in (92) shows the mapping of the construction *erst wieder an X*.[5] The rule is restricted by adjacency constraints, see Section 5.1.6. For inserting the scopal adverb *wieder* in the translation from English to German we anchor the preposition's group label `G` as the scope domain of the adverb on the right hand side condition part. `G` corresponds to the group label of the verb.

(92)
```
# adjac(L1,L2),adjac(L2,L3) <-> # in_g(L3,G)
@scop(erst,neg,L1,_)
@lhs_scop(wieder,L2,_,G)
@prep(temp_loc,before_prepx,L3,I,_).
```

(93) shows the expansion of (92) for German to English, and (94) for English to German.

(93)
```
erst(A,B),wieder(C,D),temp_loc(E,F,G) # adjac(A,C),adjac(C,E) ->
     neg(A,B),nop(C,D),before_prepx(E,F,G).
```

(94)
```
neg(A,B),before_prepx(C,D,E)
     # in_g(C,F),top_hole(wieder,F,G) ->
     erst(A,B),wieder(H,I),temp_loc(C,D,E),in_g(H,J),leq(F,I),leq(J,G).
```

4.2.6 Insertions and Deletions of Arguments

Now let us take a look at some combined templates. In this section, we consider the insertions and deletions of nominal, pronominal and propositional arguments that make use of templates defined in the previous sections.

Deletion and Insertion of Nominal Arguments. An example of this case is shown in (95). The template is defined in (96) and (97).

(95) Entschuldigung, jetzt habe ich mich *vertan*.
Sorry, I've just *made a mistake*.

The definitions in (96) and (97) use the NP templates predefined in Section 4.2.2. The template head takes the usual variables of an NP argument and the group label `G` which binds the scope of the quantifier. It has to be instantiated when applying the template. Note that for attaching these templates to the verb the variables `L` *and* `I` have to be instantiated.

[5] The predicate `temp_loc` used in the `prep` template of (92) is assigned in the refinement step, see Section 3.2. It abstracts away from the concrete static temporal preposition used in the utterance. Hence, this rule works for other temporal PPs as well, e.g. *erst wieder um 10* or *erst wieder im Mai*.

(96)
```
rhs_arg_np(arg(Arg),Qua,Noun,Num,L,I,I1,G) :=
     # in_g(L,G) <-> Arg(L,I,I1)
     @rhs_np(Qua,Noun,Num,_,I1,G).
```

(97)
```
lhs_arg_np(arg(Arg),Qua,Noun,Num,L,I,I1,G) :=
     Arg(L,I,I1) <->  # in_g(L,G)
     @lhs_np(Qua,Noun,Num,_,I1,G).
```

Template head:	`Arg`	Argument type
	`Qua`	Quantifier of `Noun`
	`Noun`	Nominal predicate
	`Num`	Number of `Noun`
	`L`	Label of `Arg`
	`I`	Instance variable of `Arg`
	`I1`	Instance variable of `Noun`
	`G`	Group label of `L`

(98) illustrates the use of (96) for the bidirectional translation of the verb *sich vertun* into the support verb construction *make a mistake*, see (95). It requires the insertion of an `arg3` NP. The corresponding rule expansions are shown in (99) and (100).

(98)
```
@verb(vertun_refl,make,L,I)
@rhs_arg_np(arg3,indef,mistake,sg,L,I,_,_).
```

(99)
```
vertun_refl(A,B) # in_g(A,C),top_hole(indef,C,D) ->
     make(A,B),mistake_n(E,F),arg3(A,B,F),indef(G,F,H,I),
     in_g(E,H),in_g(G,J),leq(C,I),leq(J,D),num(F,sg).
```

(100)
```
make(A,B),mistake_n(C,D),arg3(A,B,D),indef(E,D,F,G),
     num(D,sg),in_g(C,F),in_g(E,H),leq(H,I),leq(J,G) ->
     vertun_refl(A,B).
```

It is useful to have also templates for the insertion and deletion of nominal arguments *with an adjectival modifier*, see (101).

(101) Dann könnten wir *schnell essen.*
Then we could *have a quick meal.*

The templates used for this translation pattern are presented in (102) and (103). They are based on the predefined templates in (66) and (67), see Section 4.2.2.

(102)
```
rhs_arg_mod_np(Arg,Qua,Adj,Noun,Num,L,I,I1,G) :=
     # in_g(L,G) <-> Arg(L,I,I1)
     @rhs_mod_np(Qua,Adj,Noun,Num,_,I1,G).
```

(103)
```
lhs_arg_mod_np(Arg,Qua,Adj,Noun,Num,L,I,I1,G) :=
     Arg(L,I,I1) <-> # in_g(L,G)
     @lhs_mod_np(Qua,Adj,Noun,Num,_,I1,G).
```

Template head:	`Arg`	Argument type
	`Qua`	Quantifier of `Noun`
	`Adj`	Modifier of `Noun`
	`Noun`	Nominal predicate
	`Num`	Number of `Noun`
	`L`	Label of `Arg`
	`I`	Instance variable of `Arg`
	`I1`	Instance variable of `Noun`
	`G`	Group label of `L`

As an example, consider the transfer rule of *schnell essen* into *have a quick meal* in (104). If a modified verb is translated into a support verb construction the modifier has to be attached to the nominal head of the support verb in the TL. Sometimes it is assigned a translation that differs from its regular translation because of changed co-occurrence restrictions in the TL. But even if the translation of the modifier remains the same, we cannot use its regular translation rule because its instance variable has to be changed. So, we insert a modified argument NP and remove the SL verb modifier. The two expanded versions of the rule in (104) are shown in (105) and (106).

(104)
```
# not(arg3(L,I,_)) <-> true
@verb(essen,have,L,I)
@rhs_arg_mod_np(arg3,indef,quick,meal,sg,L,I,_,G)
@lhs_adx(schnell,_,I,G).
```

(105)
```
schnell(A,B),essen(C,B),in_g(A,D)
     # not(arg3(C,B,E)),in_g(C,D),top_hole(udef,D,F) ->
     have(C,B),meal(G,H),quick(I,H),arg3(C,B,H),udef(J,H,K,L),
     in_g(G,K),in_g(I,K),in_g(J,M),leq(D,L),leq(M,F),num(H,sg).
```

(106)
```
have(A,B),meal(C,D),quick(E,D),arg3(A,B,D),udef(F,D,G,H),
     num(D,sg),in_g(C,G),in_g(E,G),in_g(F,I),leq(I,J),leq(K,H) ->
     schnell(L,B),essen(A,B),in_g(L,K).
```

Insertions and Deletions of Pronominal Arguments. In the translation from German to English, we often have to remove or insert pronominal arguments, in particular reflexive pronouns, see (56).

(56) Wir sehen *uns* am Montag.
We'll meet on Monday.

For this kind of operation, we use the templates in (107) and (108). The template head is much the same as for the insertion of pronouns, see Section 4.2.1, except for adding the type of argument in question.

(107)
```
rhs_arg_pron(arg(Arg),D1,D2,Num,L,I,I1,G) :=
     # in_g(L,G) <-> Arg(L,I,I1)
     @rhs_pron(I1,D1,D2,Num,G).
```

(108)
```
lhs_arg_pron(arg(Arg),D1,D2,Num,L,I,I1,G) :=
     Arg(L,I,I1) <-> # in_g(L,G)
     @lhs_pron(I1,D1,D2,Num,G).
```

Template head:	`Arg`	Argument type
	`D1`	Reference designator of the pronoun
	`D2`	Sort designator of the pronoun
	`Num`	Number of the pronoun
	`L`	Label of `Arg`
	`I`	Instance variable of `Arg`
	`I1`	Instance variable of the pronoun
	`G`	Group label of `L`

The rule in (109) shows the translation of *sich sehen* into *meet*, see (56). Before we can delete the pronoun we have to make sure that it is coreferent with the subject.[6] If it is not used reflexively and cannot be removed. The coreference of the pronouns in the `arg3` role (direct object) and the `arg1` role (subject) is checked with the values in the condition part: they have to have the same reference designator `D` and the same number, which is plural for this particular verb. The same procedure is necessary for inserting a reflexive pronoun to generate the appropriate form. The expanded rules of (109) for both translation directions are shown in (110) and (111).

(109)
```
# arg1(L,I,I2),num(I2,pl),prontype(I2,D,std) <->
# arg1(L,I,I2),num(I2,pl),prontype(I2,D,std)
@verb(sehen,meet_v,L,I)
@lhs_arg_pron(arg3,D,refl,pl,L,I,_,_).
```

(110)
```
sehen(A,B),pron(C,D),arg3(A,B,D),prontype(D,E,refl),num(D,pl),
     in_g(C,F) # num(G,pl),arg1(A,B,G),prontype(G,E,std) ->
     meet_v(A,B).
```

(111)
```
meet_v(A,B) # in_g(A,C),num(D,pl),arg1(A,B,D),prontype(D,E,std)
     pron(F,G),sehen(A,B),arg3(A,B,G),in_g(F,C),num(G,pl),
     prontype(G,E,refl).
```

Insertions and Deletions of Propositional Arguments. The insertion or deletion of propositional arguments of nouns, as required for the translation of *Terminschwierigkeit* into English, see (112), is captured by the templates in (113) and (114). The proposition is attached to the noun by the `equal` relation which takes a hole variable as third argument. This hole is plugged by the group label of the embedded verb.

(112) Im Mai habe ich *Terminschwierigkeiten.*
I have *difficulties to find a time* in May.

(113)
```
rhs_prop_arg(verb(V),L,I,L2,I1) :=
     # in_g(L,G) <-> equal(L1,I,H),V(L2,I1),leq(G1,H),in_g(L1,G),
     in_g(L2,G1).
```

[6]This is actually a task of anaphoric resolution. The identification of coreference in the transfer rules is rather expensive. Moreover, a rule as in (109) accounts only for the case that both arguments, i.e., the argument of the subject (`arg1`) and the argument of the object (`arg3`), are realized by pronouns. There are some more variants to be considered. But this is not possible since there are no values to be compared. In general, coreference checking makes the rules less robust, since they become very context sensitive with a high number of constraints.

(114)
```
lhs_prop_arg(verb(V),L,I,L2,I1) :=
     equal(L1,I,H),V(L2,I1),leq(G1,H),in_g(L1,G),in_g(L2,G1) <->
     # in_g(L,G).
```

Template head:	`V`	Verb
	`L`	Label of the noun it attaches to
	`I`	Instance variable of the noun it attaches to
	`L2`	Label of `V`
	`I1`	Instance variable of `V`

(115) shows the application of (113) to the translation of the noun *Terminschwierigkeit* into *difficulty to find a time.* In (116) and (117) show the two template expansions.

(115)
```
@noun(terminschwierigkeit,difficulty,L,I)
@rhs_prop_arg(find,L,I,L1,I1)
@rhs_arg_np(arg3,indef,time,sg,L1,I1,_,_).
```

(116)
```
terminschwierigkeit(A,B) # in_g(A,C),in_g(D,E),top_hole(indef,E,F) ->
     difficulty(A,B),equal(G,B,H),find(I,J),arg3(D,J,K),
     indef(L,K,M,N),time(O,K),leq(P,H),in_g(G,C),in_g(I,P),
     num(K,sg),in_g(O,M),in_g(L,Q),leq(Q,F),leq(E,N).
```

(117))
```
difficulty(A,B),find(C,D),time(E,F),arg3(G,D,F),equal(H,B,I),
     indef(J,F,K,L),leq(M,I),in_g(H,N),in_g(C,M),num(F,sg),in_g(E,K),
     in_g(J,O),leq(O,P),leq(Q,L) -> terminschwierigkeit(A,B).
```

4.2.7 Insertions and Deletions of PPs

Now let us turn to the insertions and deletions of prepositional modifiers. There are templates for PPs with nominal and pronominal internal arguments.

Insertion and Deletion of PPs with Nominal Arguments. These operations are required for cases as shown in (118). They are defined by the templates in (119) and (120) that make use of the templates given in Sections 4.2.2 and 4.2.4. As for the insertion of adjectival and adverbial modifiers, the label of the prepositional modifier has to be put into the group of the modified predicate. It has to be instantiated when applying the template, see Section 4.2.3.

(118) Ich muß morgen wieder *verreisen.*
Tomorrow I'll have to *go on a trip* again.

(119)
```
rhs_pp(Prep,YesNo,Qua,Noun,Num,L,I,I1,G) :=
    @rhs_prep(Prep,YesNo,L,I,I1,G)
    @rhs_np(Qua,Noun,Num,_,I1,G).
```

(120)
```
lhs_pp(Prep,YesNo,Qua,Noun,Num,L,I,I1,G) :=
     @lhs_prep(Prep,YesNo,L,I,I1,G)
     @lhs_np(Qua,Noun,Num,_,I1,G).
```

Template head:	`Prep`	Preposition
	`YesNo`	Directionality of `Prep`
	`Qua`	Quantifier of `Noun`
	`Noun`	Internal argument of `Prep`
	`Num`	Number of `Noun`
	`L`	Label of `Prep`
	`I`	External argument of `Prep`
	`I1`	Instance variable of `Noun`
	`G`	Group label of `Prep`

(121) shows the application of the template in (119) for the translation of *verreisen/wegfahren* into *go on a trip*. The verb and PP templates are connected via the instance variable `I`. The label of the preposition is put into the group `G` of the verb. The expansions of this rule are shown in (122) and (123).

(121)
```
# in_g(L,G),not(stat(L,_)) <-> true
@verb(abstr_verreisen,go,L,I)
@rhs_pp(on,yes,indef,trip,sg,_,I,_,G).
```

(122a)
```
verreisen(A,B) # not(stat(A,C)),in_g(A,D),top_hole(indef,D,E) ->
     go(A,B),trip(F,G),on(H,B,G),indef(I,G,J,K),dir(H,yes),
     in_g(H,D),in_g(F,J),in_g(I,L),leq(D,K),leq(L,E),num(G,sg).
```

(122b)
```
wegfahren(A,B) # not(stat(A,C)),in_g(A,D),top_hole(indef,D,E) ->
     go(A,B),trip(F,G),on(H,B,G),indef(I,G,J,K),dir(H,yes),
     in_g(H,D),in_g(F,J),in_g(I,L),leq(D,K),leq(L,E),num(G,sg).
```

(123)
```
go(A,B),trip(C,D),prep(E,B,D),indef(F,D,G,H),dir(E,yes),
     in_g(E,I),num(D,sg),in_g(C,G),in_g(F,J),leq(J,K),leq(I,H) ->
     abstr_verreisen(A,B).
```

We also provide PP insertion templates *with a modified internal argument*, see (124) and (125). An example of their usage will be given in (174) in Section 4.3.4. For the most part, they are used to define the templates for the change of an adverbial or adjectival modifier into a PP with a modified internal argument, and vice versa, see Section 4.3.5.

(124)
```
rhs_mod_pp(Prep,YesNo,Qua,Adj,Noun,Num,L,I,I1,G) :=
     @rhs_prep(Prep,YesNo,L,I,I1,G)
     @rhs_mod_np(Qua,Adj,Noun,Num,_,I1,G).
```

(125)
```
lhs_mod_pp(Prep,YesNo,Qua,Adj,Noun,Num,L,I,I1,G) :=
     @lhs_prep(Prep,YesNo,L,I,I1,G)
     @lhs_mod_np(Qua,Adj,Noun,Num,_,I1,G).
```

Template head:	`Prep`	Preposition
	`YesNo`	Directionality of `Prep`
	`Qua`	Quantifier of `Noun`
	`Adj`	Modifier of `Noun`
	`Noun`	Internal argument of `Prep`
	`Num`	Number of `Noun`
	`L`	Label of `Prep`
	`I`	External argument of `Prep`
	`I1`	Instance variable of `Noun`
	`G`	Group label of `Prep` and scope of `Qua`

Insertion and Deletion of PPs with Pronominal Arguments. An example for this operation is shown in (57). The corresponding templates are defined in (126) and (127). Here, again, we need the group label of the predicate the PP attaches to.

(57) Ich würde das jetzt gern *hinter mich* bringen.
I would like to get it over with.

(126)
```
rhs_pp_pron(Prep,YesNo,D1,D2,Num,L,I,I1,G) :=
     @rhs_prep(Prep,YesNo,L,I,I1,G)
     @rhs_pron(I1,D1,D2,Num,G).
```

(127)
```
lhs_pp_pron(Prep,YesNo,D1,D2,Num,L,I,I1,G) :=
     @lhs_prep(Prep,YesNo,L,I,I1,G)
     @lhs_pron(I1,D1,D2,Num,G).
```

Template head:	`Prep`	Preposition
	`YesNo`	Directionality of `Prep`
	`D1`	Reference designator of the pronoun
	`D2`	Sort designator of the pronoun
	`Num`	Number of the pronoun
	`L`	Label of the pronoun
	`I`	External argument of `Prep`
	`I1`	Instance variable of the pronoun
	`G`	Group label of `Prep` and the pronoun

(128) shows the mapping of *etwas hinter sich bringen* into *to get something over with*. The German PP which contains a reflexive pronoun that is coreferent with the subject has to be deleted if we translate from German into English, and inserted when translated the reverse direction. The condition parts instantiate the values in which the subject (`arg1`) and the reflexive pronoun have to agree, namely those of the reference designator `D` and number `Num`.

(128)
```
# arg1(L,I,I1),prontype(I1,D,std),num(I1,Num) <->
# arg1(L,I,I1),prontype(I1,D,std),num(I1,Num),in_g(L,G)
@verb(bringen,get_over_with,L,I)
@lhs_pp_pron(hinter,yes,D,refl,Num,_,I,_,G).
```

(129)
```
bringen(A,B),pron(C,D),hinter(E,B,D),dir(E,no),in_g(E,F),prontype(D,G,refl),
     num(D,H),in_g(C,F) # num(I,H),arg1(A,B,I),prontype(I,G,std) ->
     get_over_with(A,B).
```

(130)
```
get_over_with(A,B) # in_g(A,C),num(D,E),arg1(A,B,D),prontype(D,F,std) ->
        bringen(A,B),pron(G,H),hinter(I,B,H),dir(I,yes),
        in_g(I,C),in_g(G,C),num(H,E),prontype(H,F,refl).
```

A variant of this kind of template is the insertion and deletion of possessive pronouns that are analyzed with a `poss(_,I1,I)` predicate which is a prepositional relation between the possessed (`I1`) and the possessor (`I`). The definitions in (131) and (132) use the same templates as in (126) and (127) with fixed values for `Prep` and `YesNo`. An example of its application is given in (191) in Section 4.3.5.

(131)
```
rhs_poss_pron(I1,I,D1,D2,Num,G) :=
        @rhs_prep(poss,no,_,I1,I,G)
        @rhs_pron(I,D1,D2,Num,G).
```

(132)
```
lhs_poss_pron(I1,I,D1,D2,Num,G) :=
        @lhs_prep(poss,no,_,I,I1,G)
        @lhs_pron(I1,D1,D2,Num,G).
```

Template head:		
	`I1`	Instance variable of the possessed
	`I`	Instance variable of the possessor (pronoun)
	`D1`	Reference designator of the pronoun
	`D2`	Sort designator of the pronoun
	`Num`	Number of the pronoun
	`G`	Group label of `poss` and the pronoun

4.3 Templates for Categorial Changes between SL and TL

In addition to insertions and deletions, there are cross-linguistic changes between categories, i.e., the meaning of a word or a phrase is conveyed by different categories. Most of these changes are syntactic ones with reflections on the semantic level. In the following sections, I consider among others, argument switching, argument-modifier switching and head switching, changes from predicative into verbal constructions and changes from adjectival or adverbial modifier to prepositional modifier.

4.3.1 Argument Switching

Argument switching concerns cases where the SL verb has a different argument frame as the TL verb, as e.g., in (133). The subject of the German verb *der Juli* is assigned the `arg3` role and its indirect object *mir* the `arg2` role. For the corresponding English verb *think of*, the subject *I* is linked to the `arg1` role and the prepositional object *July* to the `arg2`. Hence, when translating the verb its arguments have to be mapped appropriately, see (136).

(133) *Mir* schwebt da *der Juli* vor.
I think of *July*.

Not all uses of the argument switch mechanisms are motivated by actual divergences in the argument structures of the SL verb and its TL counterpart. Some of them are necessary solely for “technical reasons”: linking mechanisms differ between SL and TL grammar.

For defining the template for verbs that change their argument frame in (135) I introduce an auxiliary template in (134) that captures the argument changes and can be used in addition to (135) if more than one argument needs to be changed.

(134)
```
arg_switch(arg(arg),arg(TLarg),L,I,I1) :=
      SLarg(L,I,I1) <-> TLarg(L,I,I1).
```

(135a)
```
arg_switch_verb([SLverb,SLarg],[TLverb,TLarg],L,I,I1) :=
      @verb(SLverb,TLverb,L,I)
      @arg_switch(SLarg,TLarg,L,I,I1).
```

(135b)
```
arg_switch_verb([SLverb,SLarg],[TLverb,TLarg]) :=
      @arg_switch_verb([SLverb,SLarg],[TLverb,TLarg],_,_,_).
```

Template head:	`SLverb`	SL verb
	`SLarg`	Argument of `SLverb`
	`TLverb`	TL verb
	`TLarg`	Argument of `TLverb`
	`L`	Label of `SLverb` and `TLverb`
	`I`	Instance variable of `SLverb` and `TLverb`
	`I1`	Instance variable of `ArgSL` and `ArgTL`

(136) shows the rule for the transfer of *vorschweben* in (133). Here, both templates, (135a) and (134) are applied. The verb *vorschweben* is mapped to *think of* with the German indirect object `arg2` becoming the subject `arg1` of the TL verb, and the direct object `arg3` of the German verb being transformed into an `arg2` in English. The template expansions are shown in (137) and (138).

(136)
```
@arg_switch_verb([vorschweben,arg2],[think_of,arg1],L,I,_)
@arg_switch(arg3,arg2,L,I,_).
```

(137)
```
vorschweben(A,B),arg2(A,B,C),arg3(A,B,D) ->
      think_of(A,B),arg1(A,B,C),arg2(A,B,D).
```

(138)
```
think_of(A,B),arg1(A,B,C),arg2(A,B,D) ->
      vorschweben(A,B),arg2(A,B,C),arg3(A,B,D).
```

Another possible argument switch is that of an NP argument to a propositional argument, where the NP is an event nominalization, see (139). The templates for this switch are presented in (140) and (141).

(139) *Zugfahren* dauert zu lange.
To go by train takes too long.

(140)
```
arg_nomTOarg_prop([arg(Arg),Qua,Noun,Num],Verb,L,I,L1,I1,G1) :=
      Arg(L,I,I1),nom(L1,_) # in_g(L,G) <->
      Arg(L,I,H),Verb(L1,I1),in_g(L1,G1),leq(G1,H)
      @lhs_np(Qua,Noun,Num,L1,I1,G).
```

(141)
```
arg_propTOarg_nom(Verb,[arg(Arg),Qua,Noun,Num],L,I,L1,I1,G1) :=
      Arg(L,I,H),Verb(L1,I1),in_g(L1,G1),leq(G1,H) <->
      Arg(L,I,I1),nom(L1,_) # in_g(L,G)
      @rhs_np(Qua,Noun,Num,L1,I1,G).
```

Template head:	`Arg`	Argument type
	`Noun`	SL nominal predicate filler of `Arg`
	`Quant`	Quantifier of `Noun`
	`Verb`	TL verb
	`L`	Label of `Arg`
	`I`	Instance variable of `Arg`
	`L1`	Label of `Noun`, `nom` and `Verb`
	`I1`	Instance variable of `Noun` and `Verb`
	`G1`	Group label of `Verb`

(142) shows the mapping of the event nominalization *Zugfahren* to *to go by train* when it occurs as argument of the verb *dauern*. The template expansions are shown in (143) and (144).

(142)
```
@verb(dauern,take_adv,L,I)
@arg_nomTOarg_prop([arg3,udef,zugfahren,sg],go,L,I,I1,G)
@rhs_pp(by,no,udef,train,sg,_,I1,_,G).
```

(143)
```
zugfahren(A,B) # in_g(A,C),nom(A,D),top_hole(udef,C,E) ->
     go(A,B),train_n(F,G),by(H,B,G),udef(I,G,J,K),dir(H,no),
     in_g(H,C),in_g(F,J),in_g(I,L),leq(C,K),leq(L,E),num(G,sg).
```

(144)
```
go(A,B),take_adv(C,D),train(E,F),arg3(C,D,G),by(H,B,F),
     udef(I,F,J,K),in_g(A,L),leq(L,G),dir(H,no),in_g(H,L),
     num(F,sg),in_g(E,J),in_g(I,M),leq(M,N),leq(L,K)
     # top_hole(udef,O,P) ->
     dauern(C,D),arg3(C,D,B)udef(R,B,S,T),zugfahren(A,B),
     nom(A,Q),num(B,sg),in_g(A,S),in_g(R,U),leq(U,P),leq(O,T).
```

Here, in fact, we have a change of a noun into an infinitive that might occur in other functions as well, i.e., as subject of a copula or as internal argument of a preposition. The problem is that there is no general device for these kinds of mappings since the propositional embedding for these cases is represented differently.

4.3.2 Argument Modifier Switching

Another common divergence in the argument structure of SL and TL correspondences is caused by changes between arguments and modifiers, as illustrated in (145). Here, the *bei* modifier is realized by the indirect object `arg2` of the TL verb *ask*.

(145) Ich wollte mich *bei Ihnen* erkundigen, ob Sie kommen.
I wanted to ask *you* whether you'll come.

Not all switches of arguments into prepositional modifiers, and vice versa, are motivated by an underlying difference between SL and TL. Many changes are motivated by a different treatment of argument-like modifiers in the SL and TL grammars. For the changes observed so far, we provide two template variants which are different wrt. the number of arguments in the template head.

For argument-modifier switching templates, we need the auxiliary definitions in (146) and (147). They replace a preposition by an argument role, and vice versa.

(146)
```
prepTOarg([prep(Prep),YesNo],arg(Arg),L,I,I1,G) :=
    Prep(L1,I,I1),dir(L1,YesNo),in_g(L1,G) <-> Arg(L,I,I1) # in_g(L,G).
```

(147)
```
argTOprep(arg(Arg),[prep(Prep),YesNo],L,I,I1,G) :=
    Arg(L,I,I1) # in_g(L,G) <-> Prep(L1,I,I1),dir(L1,YesNo),in_g(L1,G).
```

Template head:	`Prep`	Preposition that becomes `Arg`
	`YesNo`	Directionality of `Prep`
	`Arg`	Argument that becomes `Prep`
	`L`	Label of `Arg`
	`I`	Instance variable of the external arg. of Arg and `Prep`
	`I1`	Instance variable of the internal arg. of Arg and `Prep`
	`G`	Group label of `L`

Now, the templates in (146) and (147) are combined with the verb mapping template. We obtain templates for verbs that change their arguments into modifiers, and vice versa, see (148) and (149).

(148a)
```
v_argTOv_prep([SLverb,Arg],[TLverb,TLprep,Yes_No],L,I,I1,G) :=
    @verb(SLverb,TLverb,L,I)
    @argTOprep(Arg,[TLprep,Yes_No],L,I,_,G).
```

(148b)
```
v_argTOv_prep([SLverb,Arg],[TLverb,TLprep,Yes_No]) :=
    @v_argTOv_prep([SLverb,Arg],[TLverb,TLprep,Yes_No],_,_,_,_).
```

Template head:	`SLverb`	SL verb
	`Arg`	Argument of `SLverb` that becomes `TLprep`
	`TLverb`	TL verb
	`TLprep`	Preposition of `TLverb`
	`YesNo`	Directionality of `TLprep`
	`L`	Label of `Arg`, `SLverb` and `TLverb`
	`I`	Instance variable of `Arg`, `SLverb` and `TLverb`
	`I1`	Instance variable of `Arg` and `TLprep`
	`G`	Group label of `L`

(149a)
```
v_prepTOv_arg([SLverb,SLprep,YesNo],[TLverb,Arg],L,I,I1,G) :=
    @verb(SLverb,TLverb,L,I)
    @prepTOarg([SLprep,YesNo],Arg,L,I,I1,G).
```

(149b)
```
v_prepTOv_arg([SLverb,SLprep,YesNo],[TLverb,Arg]) :=
    @v_prepTOv_arg([SLverb,SLprep,YesNo],[TLverb,Arg],_,_,_,_).
```

Template head:	`SLverb`	SL verb
	`SLprep`	Preposition of `SLverb` that becomes `Arg`
	`YesNo`	Directionality of `SLprep`
	`TLverb`	TL verb
	`Arg`	Argument of `TLverb`
	`L`	Label of `Arg`, `SLverb` and `TLverb`
	`I`	Instance variable of `SLverb` and `TLverb`
	`I1`	Instance variable of `Arg` and `SLprep`
	`G`	Group label of `L`

As an example, consider the application of the template in (149b) for the translation of *sich erkundigen bei jemanden* into *ask somebody* in (150). The rule expansions are shown in (151) and (152).

(150)
```
@v_prepTOv_arg([erkundigen_refl,perspective,no],[ask,arg2]).
```

(151)
```
erkundigen_refl(A,B),perspective(C,B,D),dir(C,no),in_g(C,E) ->
      ask(A,B),arg3(A,B,D).
```

(152)
```
ask(A,B),arg2(A,B,C) # in_g(A,D) ->
      erkundigen_refl(A,B),perspective(E,B,C),dir(E,no),in_g(E,D).
```

(154) shows another useful template for the transfer of subcategorized idiosyncratic prepositions that can be combined with the regular template for verbs or nouns, as required for the example in (153). Note that directionality information is entered into the template head of (154) because it might change with the translation of the preposition.

(153) Können wir das Treffen *auf* den Montag *legen*?
Could we *schedule* the meeting *for* Monday?

(154)
```
prepTOprep([prep(SLprep),Yes_No1],[prep(TLprep),Yes_No2],L,I,I1) :=
      SLprep(L,I,I1),dir(L,Yes_No1) <->
      TLprep(L,I,I1),dir(L,Yes_No2).
```

Template head:	`SLprep`	SL preposition
	`YesNo1`	Directionality of `SLprep`
	`TLprep`	TL preposition
	`YesNo2`	Directionality of `TLprep`
	`L`	Label of `SLprep` and `TLprep`
	`I`	Instance variable of external argument of `SLprep` and `TLprep`
	`I1`	Instance variable of internal argument of `SLprep` and `TLprep`

Then, verbs with idiosyncratic prepositions are translated as shown with the mapping for *legen auf* to *schedule for* in (155). The corresponding template expansions are shown in (156) and (157).

(155)
```
# sort_check(I1,time) <-> true
@verb(legen,schedule,_,I)
@prepTOprep([auf,yes],[for,no],_,I,I1).
```

(156)
```
legen(A,B),auf(C,B,D),dir(C,yes) # sort_check(D,time) ->
      schedule(A,B),for(C,B,D),dir(C,no).
```

(157)
```
schedule(A,B),for(C,B,D),dir(C,no) ->
      legen(A,B),auf(C,B,D),dir(C,yes).
```

4.3.3 Head Switching

Head switching, as illustrated with the example in (158), is first of all triggered by attitude adverbs, such as *gerne*, that are analyzed in Verb*mobil* as operators. They are often translated into a control verbs, such as *like* in our example. It embeds the proposition that was in the scope of the triggering adverb in the SL.

(158) Ich würde das Treffen *gerne* am Montag machen.
I would *like* to schedule the meeting on Monday.

For head switching, it is necessary to transmit the categorial information from the SL main verb to the inserted TL control verb. This is done by the auxiliary `temo_shift` template in (159). The values of the tense and mood information `D1` and `D2`, as well as the feature `ta_perf`, which marks the perfective aspect of a verb, are handed over to the instance of the control verb.

(159)
```
temo_shift(I1,I) :=
     ta_mood(I1,D1),ta_tense(I1,D2),ta_perf(I1,D3) <->
     ta_mood(I,D1),ta_tense(I,D2),ta_perf(I1,D3).
```

Template head:		
	`I`	Instance variable of the control verb
	`I1`	Instance variable of the main verb
	`D1`	Value of `ta_mood`
	`D2`	Value of `ta_tense`
	`D3`	Value of `ta_perf`

In (160), the scopal adverb is replaced by the TL verb which keeps the same label `L`. The TL verb embeds the proposition via an `arg3` which takes over the scope `H` of the SL adverb. Tense and mood information is shifted from the SL main verb to the inserted TL control verb by the attached `temo_shift` template. The `ta_pref` value of the embedded TL verb is set to `nonperf` which marks the infinitive. In order to ensure subject control in the TL, the arguments that are linked to the subject are anchored in the left condition part of (160). In most cases, the subject corresponds to an `arg1`. If there is no `arg1` the `arg3` is assumed to be the subject.

For the TL control verb only the existence of an `arg3` with propositional realization can be predicted. The argument type of the subject `Subj` has to be have to introduced by hand, since it is verb-specific.

When translating from English into German, it has to be guaranteed that the verb has a propositional argument to avoid the switch for nominal arguments, as in *I like this book.*

(160)
```
advTOverb(scop(Adv),[verb(TLverb),arg(Subj)],L,I) :=
     Adv(L,H) # rel(verb,L1,I1),L1=<H,
     (arg1(L1,I1,I2);(arg3(L1,I1,I2),not(arg1(L1,I1,I2)))) <->
     TLverb(L,I),arg3(L,I,H),Subj(L,I,I2),ta_perf(I1,nonperf)
     # rel(verb,L1,I1),L1=<H @temo_shift(I1,I).
```

Template head:		
	`Adv`	SL scopal adverb
	`TLverb`	TL control verb
	`Subj`	Argument type of the subject of `TLverb`
	`L`	Label of the `Adv` and `TLverb`
	`I`	Instance variable of the `TLverb`

The use of this template is shown with the translation of *gerne* into *like* in (161). The template expansions are given in (162) and (163).

(161)
```
@advTOverb(gern_scop,[like,arg1],_,_).
```

(162a)
```
gern_scop(A,B),ta_mood(C,D),ta_tense(C,E),ta_perf(C,F)
     # G=<B,arg1(G,C,H),rel(verb,G,C) ->
     like_v(A,I),arg1(A,I,H),arg3(A,I,B),ta_mood(I,D),
     ta_perf(I,F),ta_perf(C,nonperf),ta_tense(I,E).
```

(162b)
```
gern_scop(A,B),ta_mood(C,D),ta_tense(C,E),ta_perf(C,F)
     # not(arg1(G,C,H)),G=<B,arg3(G,C,H),rel(verb,G,C) ->
     like_v(A,I),arg1(A,I,H),arg3(A,I,B),ta_mood(I,D),
     ta_perf(I,F),ta_perf(C,nonperf),ta_tense(I,E).
```

(163)
```
like_v(A,B),arg1(A,B,C),arg3(A,B,D),ta_perf(E,nonperf),
     ta_mood(B,F),ta_tense(B,G),ta_perf(B,H)
     # I=<D,rel(verb,I,E) ->
     gern_scop(A,D),ta_mood(E,F),ta_perf(E,H),ta_tense(E,G).
```

4.3.4 Changes from Predicative into Verbal Constructions

In this section, I consider predicative constructions that are transformed into verbs, and vice versa. There are two cases to be distinguished: the change of a copula with a adjectival predicative into a verb, see (164), and the change a copula with a prepositional predicative into a verb, see (165).

(164) Mir *ist* das City-Hotel *angenehmer*.
I *prefer* the City hotel.

(165) *Bin* ich da bei Ihnen *an der richtigen Adresse*?
Am I *talking to the right person*?

(166) shows the mapping of a copula construction with an adjectival predicative to a verb. The subject of the copula becomes an argument of the verb. This is indicated by the instance variable `I1` which occurs as an instance variable of the predicative adjective on the rule's left hand side. It is bound by an argument role on the right hand side of the rule. The definitions in (166c) and (166d) capture the cases in which the predicative adjective occurs in comparative or superlative, see the rule in (168).

(166a)
```
acop_subjTOv_arg(mod(SLadj),[verb(TLverb),arg(Arg)],L,I,I1) :=
     stat(L,I),SLadj(L,I1) <-> TLverb(L,I),Arg(L,I,I1).
```

(166b)
```
acop_subjTOv_arg([SLadj,D],[TLverb,Arg]) :=
     @acop_subjTOv_arg([SLadj,D],[TLverb,Arg],_,_,_).
```

(166c)
```
acop_subjTOv_arg([mod(SLadj),deg(D)],[verb(TLverb),arg(Arg)],L,I,I1) :=
     stat(L,I),SLadj(L,I1),D(L,I1,_) <-> TLverb(L,I),Arg(L,I,I1).
```

(166c)
```
acop_subjTOv_arg([mod(SLadj),deg(D)],[verb(TLverb),arg(Arg)],L,I,I1) :=
     stat(L,I),SLadj(L,I1),D(L,I1,_) <-> TLverb(L,I),Arg(L,I,I1).
```

Template head:		
	`SLadj`	SL predicative adjective
	`TLverb`	TL verb
	`L`	Label of `SLadj`, `stat` and `TLverb`
	`I`	Instance variable of `SLadj` and `stat`
	`I1`	Instance variable of `SLadj` and `Arg`
	`D`	`comp` or `superl`

The reverse change is defined in (167), i.e., the mapping of a verb into a copula construction, where an argument becomes the subject of the copula.

```
(167a) v_argTOacop_subj([verb(SLverb)L,arg(Arg)],mod(TLadj),L,I,I1) :=
           SLverb(L,I),Arg(L,I,I1) <-> stat(L,I),TLadj(L,I1).
```

```
(167b) v_argTOacop_subj([SLverb,Arg],TLadj) :=
              @v_argTOacop_subj([SLverb,Arg],TLadj,_,_,_).
```

Template head:		
	`SLverb`	SL verb
	`Arg`	Argument of `SLverb` that becomes the copula subject
	`TLadj`	TL predicative adjective
	`L`	Label of `SLverb`, `stat` and `TLadj`
	`I`	Instance variable of `SLverb` and `stat`
	`I1`	Instance variable of `Arg` and `TLadj`

I illustrate the application of these templates with the translation of *angenehmer sein* into *prefer*, see (164). The corresponding rule is shown in (168). The comparative of the predicative adjective `angenehm` is mapped to the verb `prefer`. The German dative NP which is analyzed as a `perspective` modifier is transformed into an argument by the attached `prepTOarg` template. (169) and (170) show the expanded rules for both translation directions.

```
(168) @acop_subjTOv_arg([angenehm,comp],[prefer,arg3],L,I,_)
      @prepTOarg([perspective,no],arg1,L,I,_,_).
```

```
(169) angenehm(A,B),stat(A,C),comp(A,B,D),
          perspective(E,C,F),dir(E,no),in_g(E,G) ->
          prefer(A,C),arg3(A,C,B),arg1(A,C,F).
```

```
(170) prefer(A,B),arg1(A,B,C),arg3(A,B,D) # in_g(A,E) ->
          stat(A,B),angenehm(A,D),comp(A,D,F),
          perspective(G,B,C),dir(G,no),in_g(G,E).
```

The templates in (166) and (167) are also used for the mapping of stative passive verbs into predicative constructions, and vice versa. Here the `stat` predicate has to be attached to the verb, as shown in (171) with the mapping of *beschäftigt sein* to *be busy*.

```
(171) stat(L,_) <-> true
      @v_argTOacop_subj([beschaeftigen,arg3],busy,L,_,_).
```

Similarly, a prepositional predicative can be transformed into a verb, and vice versa. This is shown in the templates in (172)and (173), respectively.

```
(172a) pcop_subjTOv_arg(prep(SLprep),[verb(TLverb),arg(Arg)],L,I,I1,I2) :=
           stat(L,I),SLprep(L,I1,I2),dir(L,no) <-> TLverb(L,I),Arg(L,I,I1).
```

```
(172b) pcop_subjTOv_arg(SLprep,[TLverb,Arg]) :=
           @pcop_subjTOv_arg(SLprep,[TLverb,Arg],_,_,_,_).
```

Template head:	`SLprep`	SL predicative preposition
	`TLverb`	TL verb
	`Arg`	Argument of `TLverb` that was the copula subject
	`L`	Label of `SLprep`, `stat`, `TLverb` and `Arg`
	`I`	Instance variable of `stat`, `TLverb` and `Arg`
	`I1`	Instance variable of `SLprep` and `Arg`
	`I2`	Instance variable of internal arg of `SLprep`

(173a)
```
v_argTOpcop_subj([verb(SLverb),arg(Arg)],prep(TLprep),L,I,I1,I2) :=
     SLverb(L,I),Arg(L,I,I1) <-> stat(L,I),TLprep(L,I1,I2),dir(L,no).
```

(173b)
```
v_argTOpcop_subj([SLverb,Arg],TLprep) :=
     @v_argTOpcop_subj([SLverb,Arg],TLprep,_,_,_,_).
```

Template head:	`SLverb`	SL verb
	`Arg`	Argument of `SLverb` that becomes the copula subject
	`TLprep`	TL predicative preposition
	`L`	Label of `SLverb`, `Arg`, `stat` and `TLprep`
	`I`	Instance variable of `SLverb` and `stat`
	`I1`	Instance variable of `Arg` and `TLprep`
	`I2`	Instance variable of internal argument of `TLprep`

As an example, I give the rule in (174) for the idiomatic expression *an der richtigen Adresse sein* meaning *to talk to the right person*, see (165). It is composed of three templates, see also (67) in Section 4.2.2, and (124) in Section 4.2.7. The expansions of this rule are shown in (175) and (176).

(174)
```
@pcop_subjTOv_arg(an,[talk_v,arg1],L,I,I1,I2)
@lhs_mod_np(udef,richtig,adresse,sg,_,I2,G)
@rhs_mod_pp(to,yes,def,right_isect,person,sg,_,I,I1,G).
```

(175)
```
adresse(A,B),richtig(C,B),stat(D,E),an(D,F,B),udef(G,B,H,I),
     dir(D,no),num(B,sg),in_g(A,H),in_g(C,H),in_g(G,J),leq(J,K),
     leq(L,I) # top_hole(def,L,M) ->
     talk_v(D,E),arg1(D,E,F),to(N,E,F),def(O,F,P,Q),person(R,F),
     right_isect(S,F),dir(N,yes),in_g(N,L),num(F,sg),in_g(R,P),
     in_g(S,P),in_g(O,T),leq(T,M),leq(L,Q).
```

(176)
```
person(A,B),right_isect(C,B),talk_v(D,E),arg1(D,E,B),to(F,E,B),
     def(G,B,H,I),dir(F,yes),in_g(F,J),num(B,sg),in_g(A,H),
     in_g(C,H),in_g(G,K),leq(K,L),leq(J,I) # top_hole(udef,J,M) ->
     stat(D,E),an(D,B,N),udef(O,N,P,Q),adresse(R,N),richtig(S,N),
     dir(D,no),num(N,sg),in_g(R,P),in_g(S,P),in_g(O,T),leq(T,M),leq(J,Q).
```

Another structural change is the translation of a verb with a modifier into a copula construction where the modifier becomes the predicative, and the argument of the SL verb becomes the subject of the copula, see (177). Naturally this change occurs only with verbs that have a weak lexical meaning or a support verb character, since the verb is substituted by the copula. The corresponding templates for adverbial modifiers are shown in (178), and those for prepositional ones in (183). I do not discuss the reversion of these templates here, since there was no occurrence of this kind of change in our domain.

(177) Sie *sehen* das (ganz) *richtig*.
You *are* (really) *right*.

In the templates (178a) and (178b), the SL modifier and the TL predicative have different labels, because of the `stat` representation. In this way, the attachment of occurring intensifiers gets lost, since they share the label of the modifier. To avoid this, the additional templates in (178c) and (178d) have to be provided. They include the intensifier role `dim` and have the same template heads as in (178a) and (178b). They are applied prior to them whenever the modifier or predicative in question is intensified. In theory, we would need the same device for `comp` and `superl`, but so far no examples of this kind have been found in our domain.

(178a)
```
verb_modTOacop([verb(SLverb),arg(Arg),mod(Sadx)],TLadx,L,L1,I,I1,G) :=
     SLverb(L,I),Arg(L,I,I1),Sadx(L1,I),in_g(L1,G)
     <-> stat(L,I),TLadx(L,I1) # in_g(L,G).
```

(178b)
```
verb_modTOacop([SLverb,Arg,SLadx],TLadx) :=
     @verb_modTOacop([SLverb,Arg,SLadx],TLadx,_,_,_,_,_).
```

(178c)
```
verb_modTOacop([verb(SLverb),arg(Arg),mod(SLadx)],TLadx,L,L1,I,I1,G) :=
     SLverb(L,I),Arg(L,I,I1),SLadx(L1,I),dim(L1,I4),in_g(L1,G) <->
     stat(L,I),TLadx(L,I1),dim(L,I4) # in_g(L,G).
```

(178d)
```
verb_modTOacop([SLverb,Arg,SLadx],TLadx) :=
     @verb_modTOacop([SLverb,Arg,SLadx],TLadx,_,_,_,_,_).
```

Template head:	`SLverb`	SL verb
	`Arg`	Argument of `SLverb` that becomes the copula subject
	`SLadx`	Modifier of `SLverb` that becomes `TLadx`
	`TLadx`	TL predicative
	`L`	Label of `SLverb`
	`L1`	Label of `SLadx`
	`I`	Instance variable of `SLverb` and `stat`
	`I1`	Instance variable of `Arg` and `TLadx`
	`G`	Group label of `L` and `L1`

As an example, consider the rule in (179) that manages the translation of the utterance in (177). It expands to the rules in (180) and (181).

(179)
```
@verb_modTOacop([sehen,arg1,richtig_isect],right_isect,L,_,I,_,G).
@lhs_arg_pron(arg3,third,event,sg,L,I,_,G).
```

(180a)
```
richtig_isect(A,B),sehen(C,B),pron(D,E),arg1(C,B,F),arg3(C,B,E),
     in_g(A,G),prontype(E,third,event),num(E,sg),in_g(D,G) ->
     right_isect(C,F),stat(C,B).
```

(180b)
```
richtig_isect(A,B),sehen(C,B),dim(A,D),pron(E,F),arg1(C,B,G),arg3(C,B,F),
     in_g(A,H),prontype(F,third,event),num(F,sg),in_g(E,H) ->
     dim(C,D),right_isect(C,G),stat(C,B).
```

(181a)
```
right_isect(A,B),stat(A,C) # in_g(A,D) ->
     pron(E,F),richtig_isect(G,C),sehen(A,C),arg1(A,C,B),
     arg3(A,C,F),in_g(G,D),in_g(E,D),num(F,sg),prontype(F,third,event).
```

(181b)
```
right_isect(A,B),dim(A,C),stat(A,D) # in_g(A,E) ->
      dim(F,C),pron(G,H),richtig_isect(F,D),sehen(A,D),arg1(A,D,B),
      arg3(A,D,H),in_g(F,E),in_g(G,E),num(H,sg),prontype(H,third,event).
```

In (183), I present the template that transforms a verb with a prepositional modifier into a copula with a prepositional predicative, as illustrated in (182). Directionality information is ignored here, since only stative prepositions can undergo this change, i.e., the `dir` value remains the same.

(182) Was *steht* denn *zur Disskusion*?
What *is on the agenda*?

(183a)
```
verb_modTOpcop([verb(SLverb),arg(Arg),prep(SLprep)],TLprep,L,L1,I,I1,I2,G)) :=
      SLverb(L,I),Arg(L,I,I1),SLprep(L1,I,I2),in_g(L1,G) <->
      stat(L,I),TLprep(L,I1,I2) # in_g(L,G).
```

(183b)
```
verb_modTOpcop([SLverb,Arg,SLprep],TLprep) :=
      @verb_modTOpcop([SLverb,Arg,SLprep],TLprep,_,_,_,_,_,_).
```

Template head:		
	`SLverb`	SL verb
	`Arg`	Argument of `SLverb` that becomes the copula subject
	`SLprep`	Modifier of `SLverb` that becomes `TLprep`
	`TLprep`	TL predicative
	`L`	Label of `SLverb`
	`L1`	Label of `Sprep`
	`I`	Instance variable of `SLverb` and `stat`
	`I1`	Instance variable of `Arg` and `TLprep`
	`I2`	Internal argument of `SLprep` and `TLprep`
	`G`	Group label of `L` and `L1`

As an example for the application of the template in (183a), consider the translation of *zur Diskussion stehen* into *be on the agenda* in (184) with the expanded rules in (185) and (186).

(184)
```
@verb_modTOpcop([stehen,arg3,zu],on,_,_,_,_,I,_)
@noun(diskussion,agenda,_,I).
```

(185)
```
diskussion(A,B),stehen(C,D),arg3(C,D,E),zu(F,D,B),in_g(F,G) ->
      agenda(A,B),stat(C,D),on_prep(C,E,B).
```

(186)
```
agenda(A,B),stat(C,D),on_prep(C,E,B) # in_g(C,F) ->
      diskussion(A,B),stehen(C,D),arg3(C,D,E),zu(G,D,B),in_g(G,F).
```

4.3.5 Changes from Adverbial to Prepositional Modifiers

Now let us turn to the change of an adverbial or adjectival modifier into a prepositional one, as illustrated in (187). The corresponding templates are presented in (188) and (189). The template definitions make use of the insertion/deletion templates for nouns, see Section 4.2.2.

(187) *Meinerseits* ist der Termin in Ordnung.
For my part the time is allright.

(188a)
```
adxTOpp(mod(Adv),[prep(Prep),Yes_No,Qua,Noun,Num],L,I,I1,G) :=
     Adv(L,I) # in_g(L,G) <-> Prep(L,I,I1),dir(L,Yes_No)
     @rhs_np(Qua,Noun,Num,_,I1,G).
```

(188b)
```
adxTOpp(Adv,[Prep,Yes_No,Qua,Noun,Num]) :=
     @adxTOpp(Adv,[Prep,Yes_No,Qua,Noun,Num],_,_,_,_).
```

(189a)
```
ppTOadx([prep(Prep),YesNo,Qua,Noun,Num],mod(Adv),L,I,I1,G) :=
     Prep(L,I,I1), dir(L,YesNo) <-> Adv(L,I) # in_g(L,G)
     @lhs_np(Qua,Noun,Num,_,I1,G).
```

(189b)
```
ppTOadx([Prep,YesNo,Qua,Noun,Num],Adv) :=
     @ppTOadx([Prep,YesNo,Qua,Noun,Num],Adv,_,_,_,_).
```

Template head:		
	`Prep`	Preposition
	`YesNo`	Directionality of `Prep`
	`Qua`	Quantifier of `Noun`
	`Noun`	Internal argument of `Prep`
	`Num`	Number of `Noun`
	`Adv`	TL adverb
	`L`	Label of `Prep` and `Adv`
	`I`	Instance variable of `Prep` and `Adv`
	`I1`	Instance variable of `Noun`
	`G`	Group label of `L`

As an example, I show the translation of *meinerseits* into *for my part*, which also shows the insertion of a possessive pronoun, see (126) in Section 4.2.7. The template expansion is shown in (191).

(190)
```
@adxTOpp(meinerseits,[for,no,def,part_n,sg],_,_,I,_)
@rhs_poss_pron(I,_,sp,std,sg,_).
```

(191)
```
meinerseits(A,B) # in_g(A,C),top_hole(def,C,D) ->
     part_n(E,F),pron(G,H),for_prep(A,B,F),poss(I,F,H),def(J,F,K,L),
     dir(A,no),dir(I,no),in_g(E,K),in_g(J,M),in_g(I,C),in_g(G,C),
     leq(C,L),leq(M,D),num(F,sg),num(H,sg),prontype(H,sp,std).
```

For the change of an *adverbial modifier into a PP with a modified internal argument*, see (192), I provide the templates in (193) and (194) which make use of the templates for insertion and deletion of modified NPs, see Section 4.2.2.

(192) Dann könnten wir *gleichzeitig* unser Treffen besprechen.
Then we could talk about our meeting *at the same time*.

(193a)
```
adxTOmod_pp(mod(Adv),[prep(Prep),Yes_No,Qua,Adj,Noun,Num],L,I,I1,G) :=
     Adv(L,I) # in_g(L,G) <-> Prep(L,I,I1),dir(L,Yes_No)
     @rhs_mod_np(Qua,Adj,Noun,Num,_,I1,G).
```

(193b)
```
adxTOmod_pp(Adv,[Prep,Yes_No,Qua,Adj,Noun,Num]) :=
     @adxTOmod_pp(Adv,[Prep,Yes_No,Qua,Adj,Noun,Num],_,_,_,_).
```

(194a)
```
mod_ppTOadx([prep(Prep),Yes_No,Qua,Adj,Noun,Num],mod(Adv),L,I,I1,G) :=
     Prep(L,I,I1),dir(L,Yes_No)  <-> Adv(L,I) # in_g(L,G)
     @lhs_mod_np(Qua,Adj,Noun,Num,_,I1,G).
```

(194b)
```
mod_ppTOadx([Prep,Yes_No,Qua,Adj,Noun,Num],Adv) :=
        @mod_ppTOadx([Prep,Yes_No,Qua,Adj,Noun,Num],Adv,_,_,_,_).
```

Template head:		
	`Prep`	Preposition
	`YesNo`	Directionality of `Prep`
	`Qua`	Modifier of `Noun`
	`Adj`	Quantifier of `Noun`
	`Noun`	Internal argument of `Prep`
	`Num`	Number of `Noun`
	`Adv`	TL adverb
	`L`	Label of `Prep` and `Adv`
	`I`	Instance variable of `Prep` and `Adv`
	`I1`	Instance variable of `Noun`
	`G`	Group label of `L`

As an example, I show the application of the template in (193b) for the translation of *gleichzeitig* into *at the same time* in (195) with the expanded rules in (196) and (197).

(195)
```
@adxTOmod_pp(gleichzeitig,[at,no,def,same,time,sg]).
```

(196a)
```
gleichzeitig(A,B) # in_g(A,C),top_hole(def,C,D) ->
        same(E,F),time(G,F),at(A,B,F),def(H,F,I,J),dir(A,no),
        in_g(G,I),in_g(E,I),in_g(H,K),leq(C,J),leq(K,D),num(F,sg).
```

(197)
```
same(A,B),time(C,B),temp_loc(D,E,B),def(F,B,G,H),dir(D,no),
        num(B,sg),in_g(C,G),in_g(A,G),in_g(F,I),leq(I,J),leq(K,H) ->
        gleichzeitig(D,E).
```

4.3.6 Restructuring of Prepositional Modifiers

Now I present some templates that concern the restructuring of prepositional modifiers. In (199), the template for the mapping of stacked prepositions to a single one is defined. The first preposition embeds the second one as its argument, see (198). So far, this phenomenon occurs only in the translation direction German-English.

(198) Ich bin *bis zum* Montag unterwegs.
I'll be out of town *until* Monday.

(199a)
```
prep_seqTOprep([prep(SLprep1),prep(SLprep2),YesNo],prep(TLprep),I,I2)) :=
        SLprep1(L,I,I1),SLprep2(L1,I1,I2),dir(L1,YesNo),in_g(L1,G) <->
        TLprep(L,I,I2) # in_g(L,G).
```

(199b)
```
prep_seqTOprep([SLprep1,SLprep2,YesNo],TLprep)  :=
        @prep_seqTOprep([SLprep1,SLprep2,YesNo],TLprep,_,_).
```

Template head:		
	`SLprep1`	SL preposition
	`SLprep2`	Embedded SL preposition
	`TLprep`	TL preposition
	`YesNo`	Directionality of `SLprep2`
	`I`	Instance variable of `SLprep1` and `TLprep`
	`I2`	Internal argument of `SLprep2` and `TLprep`

(200) exemplifies the use of the template in (199a) with the mapping of *bis zu* to the meaning abstraction `temp_span_end` under the condition that the internal argument of *zu* is a temporal entity. The expansion of this rule is shown in (201).

(200)
```
# sort_check(I,temporal) <-> true
@prep_seqTOprep([bis,zu,yes],temp_span_end,_,I).
```

(201)
```
bis(A,B,C),zu(D,C,E),dir(D,yes),in_g(D,F) # sort_check(E,temporal) ->
    temp_span_end(A,B,E).
```

Another common change is the reduction of a complex PP to a single preposition, see (202). (203) and (204) show the templates for this kind of change. A complex PP has an internal argument to which another preposition is attached.

(202) Wir können *im Bereich von* neun Uhr anfangen.
We can start *around* nine o'clock.

(203a)
```
complex_ppTOprep([prep(SLprep1),Qua,Noun,Num,prep(SLprep2),YesNo],
                 prep(TLprep),L,I,I3) :=
    SLprep1(L,I,I2),SLprep2(L1,I2,I3),dir(L1,YesNo),in_g(L1,G1) <->
    TLprep(L,I,I3) # in_g(L2,G1),in_g(L,G)
    @lhs_np(Qua,Noun,Num,L2,I2,G).
```

(203b)
```
complex_ppTOprep([SLprep1,Qua,Noun,Num,SLprep2,YesNo],TLprep) :=
    @complex_ppTOprep([SLprep1,Qua,Noun,Num,SLprep2,YesNo],TLprep,_,_,_).
```

Template head:		
	`SLprep1`	SL preposition
	`Qua`	Quantifier of `Noun`
	`Noun`	Internal argument of `SLprep1`
	`Num`	Number of `Noun`
	`SLprep2`	Preposition attached to `Noun`
	`YesNo`	Directionality of `SLprep2`
	`TLprep`	TL preposition
	`I`	Instance variable of `SLprep1` and `TLprep`
	`I3`	Instance variable of `SLprep2` and `TLprep`

(204a)
```
prepTOcomplex_pp(prep(SLprep),[prep(TLprep1),Qua,Noun,Num,
                              prep(TLprep2),YesNo],L,I,I1) :=
    SLprep(L,I,I1) # in_g(L2,G1),in_g(L,G) <->
    TLprep1(L,I,I2),TLprep2(L1,I2,I1),dir(L1,YesNo),in_g(L1,G1)
    @rhs_np(Qua,Noun,Num,L2,I2,G).
```

(204b)
```
prepTOcomplex_pp(SLprep,[TLprep1,Qua,Noun,Num,TLprep2,YesNo]) :=
    @prepTOcomplex_pp(SLprep,[TLprep1,Qua,Noun,Num,TLprep2,YesNo],_,_,_).
```

Template head:		
	`SLprep`	SL preposition
	`TLprep1`	TL preposition
	`Qua`	Quantifier of `Noun`
	`Noun`	Internal argument of `TLprep1`
	`Num`	Number of `Noun`
	`TLprep2`	Preposition attached to `Noun`
	`YesNo`	Directionality of `TLprep2`
	`I`	Instance variable of `SLprep` and `TLprep1`
	`I3`	Instance variable of `SLprep` and `TLprep2`

I illustrate the application of the template in (203b) with the mapping of *im Bereich von* with a temporal internal argument to the meaning abstraction `temp_approx` in (205). This refinement rule expands to (206).

(205)
```
# sort_check(I,time) -> true
@complex_ppTOprep([in,def,bereich,sg,von,no],temp_approx,_,_,I).
```

(206)
```
bereich(A,B),in(C,D,B),von(E,B,F),def(G,B,H,I),dir(E,no),
     in_g(E,J),num(B,sg),in_g(A,H),in_g(G,K),leq(K,L),leq(M,I)
     # sort_check(F,time) -> temp_approx(C,D,F).
```

4.3.7 Nouns and Event Nominalizations

In our domain, certain event nominalizations are represented as verbs with a nominalization role `nom(L,I)` that shares the label of the verb but not its instance variable. This representation is assigned in German only to nominalizations that are identical to the verb infinitive, and in English to the gerund form of the verb. Since this sort of nominalization cannot always be preserved in translation, see (207), we need templates which manage the mapping between regular nouns and event nominalizations. These are shown in (208) and (209).

(207) Danke für Ihr *Entgegenkommen*.
Thanks for your *kindness*.

(208a)
```
nounTOevent_nom(noun(SLnoun),verb(TLverb),L,I) :=
     SLnoun(L,I) <-> TLverb(L,_),nom(L,I).
```

(208b)
```
nounTOevent_nom(SLnoun,TLverb) :=
     @nounTOevent_nom(SLnoun,TLverb,_,_).
```

Template head:	`SLnoun`	SL noun
	`TLverb`	TL verb the noun is derived from
	`L`	Label of `SLnoun`, `TLverb` and `nom`
	`I`	Instance variable of `SLnoun` and `nom`

(209a)
```
event_nomTOnoun(verb(SLverb),noun(TLnoun),L,I) :=
     SLverb(L,_),nom(L,I) <-> TLnoun(L,I).
```

(209b)
```
nounTOevent_nom(SLnoun,TLverb) :=
     @nounTOevent_nom(SLnoun,TLverb,_,_).
```

Template head:	`SLverb`	SL verb the noun is derived from
	`TLnoun`	TL noun
	`L`	Label of `SLverb`, `TLnoun` and `nom`
	`I`	Instance variable of `TLnoun` and `nom`

As an example, I show the transfer mapping of *Entgegenkommen* to *kindness*, see (207), in (210). The corresponding rule expansions are shown in (211) and (212).

(210)
```
@event_nomTOnoun(entgegenkommen,kindness).
```

(211)
```
entgegenkommen(A,B),nom(A,C) ->
     kindness(A,C).
```

(212)
```
kindness(A,B) ->
          entgegenkommen(A,C),nom(A,B).
```

4.3.8 Decomposition of Compounds

German compounds are not yet decomposed on the input to the transfer module. They are regular nominal predicates with no internal structure. But most of them have to be decomposed in their English translation. There are two ways of treating them. They could either be decomposed in the source language, and then transferred by regular rules for non-compound words. Or they are decomposed and translated in one step. In Verb*mobil* the second, certainly less elegant way is taken. However, for a restricted domain, it is possible to keep all compounds in the lexicon. Moreover, there are many cases in which the translation of the parts is particular to their occurrence in the compound.

Table 3 exemplifies the noun decomposition classes that occur in the Verb*mobil* domain. In the following, you find the template definitions for all of these cases.

Decomposition Class	Example
Compound ↔ Noun + Noun	Notizbuch - note book
Compound ↔ Adj + Noun	Haupteingang - main entrance
Compound ↔ Noun + PP	Zugfahrt - trip on the train
Compound ↔ Noun + of + Noun	Ausgangspunkt - point of departure
Compound ↔ RelNoun + of + Noun	Jahresende - end of the year
Compound ↔ Noun + Noun + Noun	Landesrechenzentrum - state computer center
Compound ↔ Verb + ArgNoun	Terminplanung - appointment scheduling
Compound ↔ Noun + Verb	Terminschwierigkeit - difficulty to find a time

Table 3: Compound Decomposition Classes

Compound ↔ Noun + Noun. The German compound of this type has a head modifier structure. The first noun is the modifier, and the second noun is the head of the compound. In its translation, it is decomposed into the head noun, with the modifier noun being the internal argument of a unspecified relation `unspec`, which acts as a prepositional modifier to the head noun. Since this is a preposition, directionality information has to be added. The lack of the determiner of the modifier noun is indicated by the `udef` quantifier. Both `unspec` and `udef` are not lexicalized in English. The template definition in (213) uses the insertion template for prepositional modifiers in (119) with fixed values for the prepositional relation, directionality, and the quantifier of its internal argument.

(213a)
```
compoundTOnoun_noun(noun(Cmp),[ModNoun,Num,noun(HeadNoun),L,I]) :=
      Cmp(L,I) # in_g(L,G) <-> HeadNoun(L,I)
      @rhs_pp(unspec,no,udef,ModNoun,Num,_,I,_,G).
```

(213b)
```
compoundTOnoun_noun(Cmp,[ModNoun,Num,HeadNoun]) :=
      @compoundTOnoun_noun(Cmp,[ModNoun,Num,HeadNoun],_,_).
```

Template head:	`Cmp`	SL compound
	`ModNoun`	TL modifier noun of `Cmp`
	`Num`	Number of `ModNoun`
	`HeadNoun`	TL head of `Cmp`
	`L`	Label of `Cmp` and `HeadNoun`
	`I`	Instance variable of `Cmp` and `HeadNoun`

The application of the template in (213b) to the noun *Notizbuch* is shown in (214). The corresponding template expansions are given in (215) and (216).

(214)
```
@compoundTOnoun_noun(notizbuch,[note,sg,book]).
```

(215)
```
notizbuch(A,B) # in_g(A,C),top_hole(udef,C,D) ->
     book_n(A,B),note_n(E,F),unspec(G,B,F),udef(H,F,I,J),dir(G,no),
     in_g(G,C),in_g(H,K),in_g(E,I),leq(C,J),leq(K,D),num(F,sg).
```

(216)
```
book_n(A,B),note_n(C,D),unspec(E,B,D),udef(F,D,G,H),dir(E,no),
     in_g(E,I),num(D,sg),in_g(C,G),in_g(F,J),leq(J,K),leq(I,H) ->
     notizbuch(A,B).
```

Compound ↔ Adjective + Noun. A German compound with a head modifier structure can also be decomposed into a head noun with the modifier noun being transformed into an adjectival modifier in the TL, see (217). The template definition uses the insertion template for adjectival modifiers in (72) in Section 4.2.3.

(217a)
```
compoundTOmod_noun(noun(Cmp),[Mod,noun(HeadNoun)],L,I) :=
     Cmp(L,I) # in_g(L,G) <-> HeadNoun(L,I)
     @rhs_adx(Mod,_,I,G).
```

(217b)
```
compoundTOmod_noun(Cmp,[Mod,HeadNoun]) :=
     @compoundTOmod_noun(Cmp,[Mod,HeadNoun],_,_).
```

Template head:	`Cmp`	SL compound
	`Mod`	TL adjectival modifier
	`HeadNoun`	TL head of `Cmp`
	`L`	Label of `Cmp` and `HeadNoun`
	`I`	Instance variable of `Cmp` and `HeadNoun`

If we apply the template in (217b) to the word *Haupteingang* as in (218), we get the expanded rules in (219) and (220) .

(218)
```
@compoundTOmod_noun(haupteingang,[main,entrance]).
```

(219)
```
haupteingang(A,B) # in_g(A,C) ->
     entrance(A,B),main(D,B),in_g(D,C).
```

(220)
```
entrance(A,B),main(C,B),in_g(C,D) ->
     haupteingang(A,B).
```

Compound ↔ Noun + PP. In its translation, the compound is decomposed into its head noun and a preposition that takes the non-head noun as its internal argument. For this decomposition pattern, the template definition in (221) uses the insertion template for prepositional modifiers in (119), see Section 4.2.7. Since the TL preposition and quantifier of its internal argument cannot be predicted, as in

(213), they have to be entered into template head.

(221a)
```
compoundTOnoun_pp(noun(Cmp),[noun(HeadNoun),Prep,YesNo,Det,Noun,Num],L,I) :=
         Cmp(L,I) # in_g(L,G) <-> HeadNoun(L,I)
         @rhs_pp(Prep,YesNo,Det,Noun,Num,_,I,_,G).
```

(221b)
```
compoundTOnoun_pp(Cmp,[HeadNoun,Prep,YesNo,Det,Noun,Num]) :=
         @compoundTOnoun_pp(Cmp,[HeadNoun,Prep,YesNo,Det,Noun,Num],_,_).
```

Template head:		
	Cmp	SL compound
	Prep	TL prepositional modifier of HeadNoun
	Noun	TL internal argument of Prep
	Num	Number of Noun
	HeadNoun	TL head of Cmp
	L	Label of Cmp and HeadNoun
	I	Instance variable of Cmp and HeadNoun

If we apply the template in (221b) to the noun *Zugfahrt*, as in (222), we get the expanded rules in (223) and (224).

(222)
```
@compoundTOnoun_pp(zugfahrt,[trip,on,no,def,train_n,sg]).
```

(223)
```
zugfahrt(A,B) # in_g(A,C),top_hole(def,C,D) ->
     train_n(E,F),trip(A,B),loc_int(G,B,F),def(H,F,I,J),dir(G,no),
     in_g(G,C),in_g(E,I),in_g(H,K),leq(C,J),leq(K,D),num(F,sg).
```

(224)
```
train_n(A,B),trip(C,D),loc_int(E,D,B),def(F,B,G,H),dir(E,no),
     in_g(E,I),num(B,sg),in_g(A,G),in_g(F,J),leq(J,K),leq(I,H) ->
     zugfahrt(C,D).
```

Non-head nouns of compounds of the considered decomposition pattern might be three-place nominal predicates as well. That is why they are not captured by the templates in (221). (225) defines a transfer template for them. To the mapping of the head noun we attach the template for inserting a preposition, see (84) in Section 4.2.4, and the insertion template for three-place noun predicates, see (69) in Section 4.2.2.

(225a)
```
compoundTOnoun_pp3(noun(Cmp),[noun(HeadNoun),Prep,YesNo,Noun3,D],L,I) :=
         Cmp(L,I) # in_g(L,G) <-> HeadNoun(L,I)
         @rhs_prep(Prep,YesNo,_,I,I1,G)
         @rhs_np3(def,Noun3,D,sg,_,I1,G).
```

(225b)
```
compoundTOnoun_pp3(Cmp,[HeadNoun,Prep,YesNo,Noun3,D]) :=
         @compoundTOnoun_pp3(Cmp,[HeadNoun,Prep,YesNo,Noun3,D],_,_).
```

Template head:		
	Cmp	SL compound
	HeadNoun	TL head of Cmp
	Prep	TL prepositional modifier of HeadNoun
	Noun3	TL internal argument of Prep
	D	Designator of Noun3
	L	Label of Cmp and HeadNoun
	I	Instance variable of the Cmp and HeadNoun

As an example, consider the rule for the translation of *Februartermin* into *meeting in February* in (226). The expanded rules are shown in (227) and (228).

(226)
```
@compoundTOnoun_pp3(februartermin,[meeting,temp_loc,no,mofy,feb]).
```

(227)
```
februartermin(A,B) # in_g(A,C),top_hole(def,C,D) ->
     meeting(A,B),mofy(E,F,feb),temp_loc(G,B,F),def(H,F,I,J),dir(G,no),
     in_g(G,C),in_g(H,K),in_g(E,I),leq(C,J),leq(K,D),num(F,sg).
```

(228)
```
meeting(A,B),mofy(C,D,feb),temp_loc(E,B,D),def(F,D,G,H),dir(E,no),
     in_g(E,I),num(D,sg),in_g(C,G),in_g(F,J),leq(J,K),leq(I,H) ->
     februartermin(A,B).
```

Compound ↔ Noun + of + Noun. The head noun of this type of compound denotes the whole and the non-head noun a part of it. In the English translation, the part-whole relation between them is expressed by the prepositional relation `poss`. The transfer template for this type of compound in (229) is built up in essentially the same way as (221).

(229a)
```
compoundTOhead_poss(noun(Cmp),[noun(HeadNoun),Qua,PossNoun,Num],L,I) :=
     Cmp(L,I) # in_g(L,G) <-> HeadNoun(L,I)
     @rhs_pp(poss,no,Qua,PossNoun,Num,_,I,_,G).
```

(229b)
```
compoundTOhead_poss(Cmp,[HeadNoun,Qua,PossNoun,Num]) :=
     @compoundTOhead_poss(Cmp,[HeadNoun,Qua,PossNoun,Num],_,_).
```

Template head:	`Cmp`	SL compound
	`HeadNoun`	Head of `Cmp` denoting the whole
	`Qua`	Quantifier of `PossNoun`
	`PossNoun`	Noun denoting a part of `HeadNoun`
	`Num`	Number of `PossNoun`
	`L`	Label of `Cmp` and `HeadNoun`
	`I`	Instance variable of `Cmp` and `HeadNoun`

As an example for the application of (229b), consider the rule for the translation of *Ausgangspunkt* into *point of departure* in (230) with the expansions in (231) and (232).

(230)
```
@compoundTOhead_poss(ausgangspunkt,[point,udef,departure,sg]).
```

(231)
```
ausgangspunkt(A,B) # in_g(A,C),top_hole(udef,C,D) ->
     departure(E,F),point(A,B),poss(G,B,F),udef(H,F,I,J),dir(G,no),
     in_g(G,C),in_g(E,I),in_g(H,K),leq(C,J),leq(K,D),num(F,sg).
```

(232)
```
departure(A,B),point(C,D),poss(E,D,B),udef(F,B,G,H),dir(E,no),
     in_g(E,I),num(B,sg),in_g(A,G),in_g(F,J),leq(J,K),leq(I,H)
     -> ausgangspunkt(C,D).
```

The noun denoting the part can also be realized by a three-place noun. For this we need the extra template in (233).

(233a)
```
compoundTOhead_poss3(noun(Cmp),[noun(HeadNoun),Noun3,D],L,I) :=
     Cmp(L,I) # in_g(L,G) <-> HeadNoun(L,I)
     @rhs_prep(poss,no,_,I,I1,G)
     @rhs_np3(def,Noun3,D,sg,_,I1,G).
```

(233b)
```
compoundTOhead_poss3(Cmp,[HeadNoun,Noun3,D]) :=
     @compoundTOhead_poss3(Cmp,[HeadNoun,Noun3,D],_,_).
```

Template head:	`Cmp`	SL compound
	`HeadNoun`	Head of `Cmp` denoting the whole
	`Noun3`	Noun denoting a part of `HeadNoun`
	`D`	Designator of `Noun3`
	`L`	Label of `Cmp` and `HeadNoun`
	`I`	Instance variable of `Cmp` and `HeadNoun`

Compound ↔ Relational Noun + of + Noun. The head noun of this type of compound is a relational noun. In the English translation of this compound, the related noun fills the argument position (`arg3`) of the relational noun. It occurs with a definite article in singular. The definition of this pattern in (234) makes use of the NP insertion template in (62) in Section 4.2.2.

(234a)
```
compoundTOrelnoun(noun(Cmp),[noun(RelNoun),Noun],L,I) :=
        Cmp(L,I) # in_g(L,G) <-> RelNoun(L,I),arg3(L,I,I1)
        @rhs_np(def,Noun,sg,_,I1,G).
```

(234b)
```
compoundTOrelnoun(Cmp,[RelNoun,Noun]) :=
        @compoundTOrelnoun(Cmp,[RelNoun,Noun],_,_).
```

Template head:	`Cmp`	SL Compound
	`RelNoun`	Relational noun, head of `Cmp`
	`Noun`	Related noun bound by `arg3`
	`L`	Label of `Cmp` and `RelNoun`
	`I`	Instance variable of `Cmp` and `RelNoun`
	`L1`	Label of `Noun`
	`I1`	Instance variable of `Noun`

As an example, consider the translation of *Jahresende* into *end of the year* in (235) with the expansion in (236) and (237).

(235)
```
@compoundTOrelnoun(jahresende,[end_n,year]).
```

(236)
```
jahresende(A,B) # in_g(A,C),top_hole(def,C,D) ->
        end_n(A,B),year(E,F),arg3(A,B,F),def(G,F,H,I),
         in_g(G,J),in_g(E,H),leq(C,I),leq(J,D),num(F,sg).
```

(237)
```
end_n(A,B),year(C,D),arg3(A,B,D),def(E,D,F,G),num(D,sg),
        in_g(C,F),in_g(E,H),leq(H,I),leq(J,G) ->
        jahresende(A,B).
```

Compound ↔ Noun + Noun + Noun. A compound noun is decomposed into three nouns in the TL. For this, the templates in (238) are used.

(238a)
```
compoundTOnoun_noun_noun(noun(Cmp),
        [ModNoun,HeadNoun1,Num,noun(HeadNoun2)],L,I) :=
        Cmp(L,I) # in_g(L,G) <-> HeadNoun2(L,I)
        @rhs_pp(unspec,no,udef,HeadNoun1,Num,_,I,I1,G)
        @rhs_pp(unspec,no,udef,ModNoun,sg,_,I1,_,G).
```

(238b)
```
compoundTOnoun_noun_noun(Cmp,[ModNoun,HeadNoun1,Num,HeadNoun2]) :=
        @compoundTOnoun_noun_noun(Cmp,[ModNoun,HeadNoun1,Num,HeadNoun2],_,_).
```

Template head:	`Cmp`	SL compound
	`ModNoun`	Modifier noun of `HeadNoun1`
	`HeadNoun1`	First head noun of `Cmp`
	`Num`	Number of `HeadNoun2`
	`HeadNoun2`	Second head noun of `Cmp`
	`L`	Label of `Cmp`
	`I`	Instance variable of `Cmp`

As an example, consider the application of (238b) in rule (239). It manages the translation of *Landesrechnungszentrum* into *state computer center*. The expanded rules are shown in (240) and (241).

(239)
```
@compoundTOnoun_noun_noun(landesrechenzentrum,[state,computer,sg,center]).
```

(240)
```
landesrechenzentrum(A,B) # in_g(A,C),top_hole(udef,C,D),top_hole(udef,C,E) ->
      center(A,B),computer(F,G),state(H,I),unspec(J,B,G),unspec(K,G,I),
      udef(L,G,M,N),udef(O,I,P,Q),dir(J,no),dir(K,no),in_g(J,C),in_g(L,R),
      in_g(F,M),in_g(K,C),in_g(O,S),in_g(H,P),leq(C,N),leq(C,Q),leq(R,D),
      leq(S,E),num(G,sg),num(I,sg).
```

(241)
```
center(A,B),computer(C,D),state(E,F),unspec(G,B,D),unspec(H,D,F),
      udef(I,D,J,K),udef(L,F,M,N),dir(G,no),in_g(G,O),num(D,sg),
      in_g(C,J),in_g(I,P),leq(P,Q),leq(O,K),dir(H,no),in_g(H,O),
      num(F,sg),in_g(E,M),in_g(L,R),leq(R,S),leq(O,N) ->
      landesrechenzentrum(A,B).
```

Compound ↔ Noun + Verb. An example for this decomposition pattern is shown in Section 4.2.6 with the translation of *Terminschwierigkeit* into *difficulty to find a time*, see (115).

Compound ↔ Noun + Argument Noun. The head noun of this type of compound is a verb nominalization. In the English translation, it is represented as a verb with the `nom` (nominalize) predicate attached. The non-head noun is represented as its `arg3` argument. The template for this pattern is presented in (242). It makes use of the NP insertion template defined, see (62) in Section 4.2.2.

(242a)
```
compoundTOarg_noun(noun(Cmp),[ArgNoun,Num,verb(HeadVerb)],L,I) :=
      Cmp(L,I), in_g(L,G) <-> HeadVerb(L,I1),nom(L,I),arg3(L,I1,I2)
      @rhs_np(udef,ArgNoun,Num,_,I2,G).
```

(242b)
```
compoundTOarg_noun(Cmp,[ArgNoun,Num,HeadVerb]) :=
      @compoundTOarg_noun(Cmp,[ArgNoun,Num,HeadVerb],_,_).
```

Template head:	`Cmp`	SL compound
	`ArgNoun`	arg3 of the `HeadVerb`
	`Num`	Number of `ArgNoun`
	`HeadVerb`	Verb derived of the head of `Cmp`
	`L`	Label of `Cmp` and `HeadVerb`
	`I`	Instance variable of `Cmp` and `HeadVerb`

(243) illustrates the application of (242b) with the translation of *Terminplanung* into *appointment scheduling*. The expanded rules are shown in (244) and (245).

(243)
```
@compoundTOarg_noun(terminplanung,[appointment,sg,schedule_v]).
```

(244)
```
terminplanung(A,B),in_g(A,C) # top_hole(udef,C,D) ->
        appointment(E,F),nom(A,B),schedule_v(A,G),arg3(A,G,F),
        udef(H,F,I,J),in_g(H,K),in_g(E,I),leq(C,J),leq(K,D),num(F,sg).
```

(245)
```
appointment(A,B),schedule_v(C,D),nom(C,E),arg3(C,D,B),num(B,sg),
        udef(F,B,G,H),in_g(A,G),in_g(F,I),leq(I,J),leq(K,H)
        ->terminplanung(C,E),in_g(C,K).
```

5 Contextual Constraints

To restrict the transfer mapping of ambiguous SL lexical items, we follow the principle of *cascading disambiguation*, i.e., the local information available in the actual input VIT is used first, see Section 5.1. When it is exhausted global contextual information is exploited, see Section 5.2.

5.1 Information from the Local Utterance Context

The information shown in (246) is used for the resolution of translational ambiguities. It is encoded in the VIT input representation and is directly available for the specification of contextual constraints.

(246)
- Structural information
- Sortal information
- Co-occurrence restrictions
- Semantic predicate classes
- Categorial information
- Prosodic information
- Surface order

5.1.1 Structural Information

As structural information I consider the kind of connection between semantic predicates. On the one hand, it is needed to access the predicates on which we want to test particular information, see Section 5.1.2 - Sections 5.1.6. On the other hand, structural information is useful to identify certain kinds of structures that have an impact on translation. Structural connections are described by the use of semantic roles, coindexation of labels and instance variables and subordination constraints.

The Use of Semantic Roles. Sometimes a certain argument frame might be decisive for the translation of a verb. Then, the relevant argument role/s is/are simply stated in the condition part. In (247), for example, the verb *führen* is translated into *proceed*. This is allowed if the German verb does not occur with a theme argument but with a PP denoting path instead. Otherwise, its translation differs. The rule in (248) shows how the `arg3` occurrence is excluded.

(247) Die Stadtrundfahrt *führt* durch Hannover.
The city tour *proceeds* through Hannover.

(248)
```
# not(arg3(L,I,_)),path(_,I,_) -> true
@verb(fuehren,proceed,L,I).
```

Argument roles are also used to access the concrete noun that fills the argument position if we need to specify conditions on it (i.e., on their second instance variable). Consider the example in (249), where the translation of *erreichen* depends on the sort of its theme argument, see Section 5.1.2. If the `arg3` refers to a means of public transportation, *erreichen* is translated into *catch*, and vice versa. The corresponding rule is shown in (250).

(249) Sie werden den Zug nach München noch *erreichen.*
You will *catch* the train to Munich.

(250)
```
# arg3(L,I,I1),sort_check(I1,vehicle) <->
# arg3(L,I,I1), sort_check(I1,vehicle)
@verb(erreichen,catch,L,I).
```

In order to find out whether an adjective or a preposition is used as predicative of a copula verb, the `stat` role can be exploited. (251) shows an example where the predicative usage of *kurzfristig* is relevant for its translation into *soon.* The rule in (252) anchors the copula role `stat` in the condition part. It shares the label of *kurzfristig.* In other contexts *kurzfristig* is translated into *at short notice* or into *short-time.*

(251) Dieser Termin ist zu *kurzfristig.*
This meeting is too *soon.*

(252)
```
# stat(L,_) <-> true
@mod(kurzfristig,soon,L,_).
```

For the translation of some adverbs and adjectives we might need to know whether they are intensified. For this purpose the `dim, comp` and `superl` roles are used. They share the label `L` of the predicate they are related to. For example, *früh* has two different translations: *early,* as shown in (253), and *in the morning.* To make a decision, among others, the rule in (254) is applied. It says that *early* is appropriate if the German adjective is either intensified or is used in the comparative or superlative form. In these contexts a translation into *in the morning* would be wrong.

(253) Können wir uns *füher*/sehr *früh* treffen?
Could we meet *earlier*/very *early*?

(254)
```
# (dim(L,_);comp(L,_,_);superl(L,_,_)) -> true
@mod(frueh,early,L,_).
```

Other Methods of Coindexation. For the translation of modifiers we often need to have more information about the predicates they modify. The modified predicates can be easily accessed by their instance variable that is shared by the modifier. Consider the example in (255). It shows the translation of the predicate *schlecht* into *good* in cases where it modifies verbs expressing a positive attitude. They are represented by the type `abstr_passen` in the condition part of the rule in (256). In English, negative attitude adverbs cannot be combined with positive attitude verbs, so that *schlecht* has to be mapped to its TL antonym *good*, and the attitude verb has to be put in the scope of negation.

(255) Das geht/paßt/klappt bei mir *schlecht*.
That does not suit *well*.

(256)
```
# abstr_passen(L,I),in_g(L,G) -> true
@mod(schlecht,good,_,I)
@rhs_scop(neg,_,_,G).
```

Subordination Constraints. In order to access predicates that are connected by scopal relations, coordination, or propositional embedding, or to state these kinds of semantic relations as contextual constraints, we make use of subordination constraints. These are less than or equal to relations between variables, i.e., labels and hole variables. The equation in (257a) says that Variable1 (a label) and Variable2 (another label or a hole variable) are identical (direct subordination) or Variable1 is below Variable2 (indirect subordination). (257b) is used to express that Variable1 and Variable2 are not identical.

(257a) `Variable1 =< Variable2`
(257b) `Variable1 \== Variable2`

Consider the example in (258). The adverb *da* is translated into *out of town* if it occurs in the scope of the negation as predicative of the copula with a human subject. These constraints are stated in the condition part of the rule in (259). The scopal relation is expressed by the less than or equal to constraint between the label `L` of the copula and the hole variable `H` of the negation `neg`.

(258) Morgen bin ich *nicht da*.
Tomorrow I'll be *out of town*.

(259)
```
# stat(L,_),L=<H,sort_check(I,human) -> true
@lhs_scop(neg,_,H,_)
@mod(da_isect,out_of_town,L,I).
```

(260) illustrates the test on a propositional argument. The verb *finden* is translated into *think* if it has a propositional argument. In other contexts, it corresponds to *find*. The rule for the translation in (260) is presented in (261). The propositional argument is anchored in the condition part in the following way: the third argument `H` of the `arg3` of *finden* is either a label or a hole variable, which means that a verbal construction is embedded. For nouns, this argument would be an instance variable.

(260) Ich *finde*, wir sollten früher anfangen.
I *think* we should start earlier.

(261)
```
# arg3(L,I,H),(label(H);hole(H)) <-> true
@verb(finden,think,L,I).
```

5.1.2 Sortal Information

Sortal information is assigned to lexical entries in the semantic lexicon and stored in the VIT SORT SLOT, see Section 2. Sorts are frequently needed for the disambiguation of verbs, prepositions and adjectives. Figure 5 shows the sort hierarchy used by the transfer module. Almost 10% of all rules contain sortal restrictions.

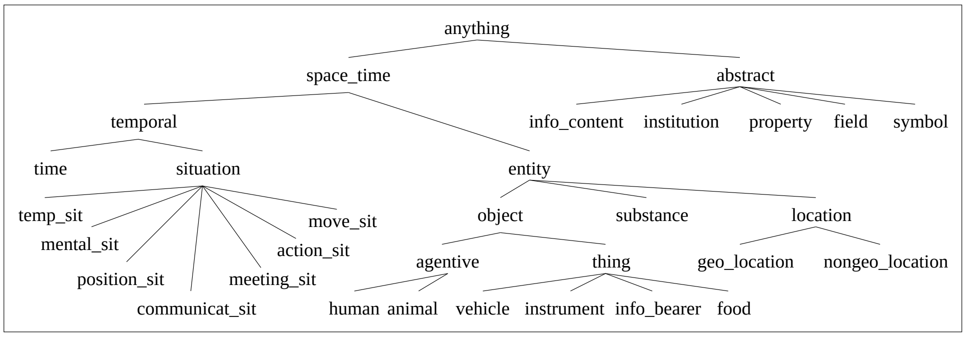

Figure 5: Sort Hierarchy for the Disambiguation of Translational Ambiguities

(262) shows the sort test predicate. `I` is the instance variable of the predicate in question and `Sort` a sortal expression of the sort hierarchy. Sortal expressions can be specified disjunctively (indicated by ";") or by use of the negation (indicated by "~").

(262) `sort_check(I, Sort)`

(263) illustrates the usage of the sort test `sort_check` with an example. The preposition *vor* is mapped to a meaning abstraction called `temp_ant(erior)` if the sort of its internal argument is subsumed by the sort `temporal`, see (264). With a location or a thing as internal argument it would get a spatial interpretation that is covered by the English preposition *in front of*.

(263) Sollen wir uns *vor* 11 Uhr treffen?
Should we meet *before* 11 o'clock?

(264)
```
# sort_check(I,temporal) -> true
@prep(vor,temp_ant,_,_,I).
```

5.1.3 Co-occurrence Restrictions

A very common method for selecting TL correspondences is the specification of certain predicates or previously introduced abstract meanings that occur as arguments or modifiers of an ambiguous word or phrase, see Section 5.1.1.

In the condition part, all predicates of the semantic lexicon and abstract predicates that are declared by a type definition in the transfer module can be used. They can be combined by conjunction and disjunction or excluded by negation. Consider the example in (265). (266) shows the corresponding bidirectional mapping between *bereiten* and *cause* if they are used as support verbs with the nouns *Problem* and *problem*, respectively.

(265) Ich denke, daß das keine Probleme *bereiten* wird.
I think this won't *cause* any problems.

(266)
```
# arg3(L,I,I1),problem(_,I1) <-> # arg3(L,I,I1),problem(_,I1)
@verb(bereiten,cause_v,L,I).
```

(267) shows the time span interpretation of the prepositions *für* and *for*. This meaning is assigned to them in the refinement module. The refinement rule in (268) maps the preposition *für* to the meaning abstraction `temp_span` if its internal argument is realized by a time measure noun. It is either in plural or has a cardinal number modifier. The abstract predicate `time_measure` is a pre-defined type for nouns like hour, month, etc.

(267) Aber ich muß *für* zwei Wochen nach Japan.
But I have to go to Japan *for* two weeks.

(268)
```
# time_measure(_,I),(num(I,pl);card(_,I,_)) ->  true
@prep(fuer,temp_span,_,_,I).
```

5.1.4 Semantic Predicate Classes

There are cases in which the semantic class of a predicate allows different translation candidates to be chosen from. Table 4 enumerates the tests on semantic classes. They are more abstract than the semantic classes in the semantic lexicons, but they are in correspondence with them.

Semantic Class Tests	Predicate with Label `L` is a
`rel(verb,L,I)`	verbal predicate
`rel(noun,L,I)`	nominal predicate
`rel(mod,L,I)`	intersective adverbial/adjectival modifier
`rel(mod,L,I,I1)`	intersective prepositional modifier
`rel(quant,L,I,L1,H)`	quantifier
`rel(scop,L,H)`	scopal adverb

Table 4: Tests on Semantic Classes

Examples for the use of semantic predicate classes are shown with the translation of the modifier *anschließend* in (269) and (270).

(269) Wie wäre es denn mit der mit der *anschließenden* Woche?
How about the *following* week?

(270) Haben Sie denn *anschließend* Zeit?
Do you have time *after that*.

(271) and (272) show the mapping rules for *anschließend*. If the modified predicate is a noun it is translated into *following*. If it is a verb *anschließend* corresponds to *after that*.

(271)
```
# rel(noun,_,I) <-> true
@mod(anschliessend,following,_,I).
```

(272)
```
# rel(verb,_,I) <-> true
@adxTOpp(anschliessend,[temp_post,no,that,abstr_nom,sg],_,I,_,_).
```

For scopal adverbs, the semantics module provides further refined subclasses that are used for disambiguation in transfer, see Table 5. This information is assigned to the `s_class` slot of the VIT.

Subclass Tests	Predicate with Label `L` is a
`s_class(L,sa)`	sentence adverb
`s_class(L,fp)`	focus particle
`s_class(L,mp)`	modal particle
`s_class(L,dra)`	discourse relation adverb

Table 5: Tests on Subclasses of Scopal Adverbs and Particles

The examples in (273) and (274) show that the German particle *bloß* is translated either into *only* or into *but*.

(273) Es geht bei mir *bloß* im März.
It suits me *only* in March.

(274) Am Montag geht es bei mir *bloß* nicht.
But it does not suit me on Monday.

The rules in (275) and (276) state the conditions under which the English counterparts are assigned. If *bloß* occurs as focus particle (`fa`) it is mapped to *only*. If the German particle is a discourse relation adverb (`dra`) it corresponds to *but*.

(275)
```
# s_class(L,fp) -> true
@scop(bloss,only_scop,L,_).
```

(276)
```
# s_class(L,dra) -> true
@scop(bloss,but_scop,L,_).
```

For more information on the semantic classification of particles, I refer to [Bos and Schiehlen, 1997].

5.1.5 Categorial Information

Sometimes certain kinds of categorial features, such as tense, mood, aspect and number are useful for choosing the appropriate TL word or phrase. They are tested with the predicates and values as they are represented in the MORPHO-SYNTAX and TENSE & ASPECT slots of the VIT, see Figure 2 in Section 2.

Sentence mood is tested with the special predicate in (277). `L` is the label of the predicate the translation of which depends on the sentence mood. Values of `SentMood` are `decl` (declarative), `quest` (question) and `imp` (imperative).

(277) `sent_mood(L,SentMood)`

(278) illustrates a case where mood information allows two alternatives to be chosen from. If the verb *denken an* is used in the imperative mood it is translated into *not to forget*. Otherwise it corresponds to *think of* which is the regular English counterpart of *denken an*. The corresponding transfer rule is given in (279). The condition part contains a test on the sentence mood.

(278) *Denken* Sie *daran*, die Unterlagen mitzubringen!
Don't forget to bring the documents!

(279)
```
# sent_mood(L,imp),in_g(L,G) -> true
@verb(denken_an,forget,L,I).
@rhs_scop(neg,_,_,G).
```

5.1.6 Surface Order

To treat clusters of German particles and to identify readings of adverbs and particles we often need to know their surface order because it can get lost in the semantic representation. This information is derived from the order of the predicate labels that point to the words of the input utterance. Naturally, it can be used only for predicates that are directly related to words.

Adjacency Tests. In order to test whether the words that correspond to two semantic predicates have been adjacent in the input utterance the predicate in (280) is used. `L` is the label of the predicate that precedes the predicate with the label `L1` in the utterance.

(280) `adjac(L,L1)`

Consider the translation of the sequence of the German particles *noch mal* in (281). The rule in (282) uses the adjacency test to map *noch mal* to *again*. This is done by mapping *noch* to *again* and removing *mal*.

(281) Wir sollten wir uns *noch mal* treffen.
We should meet *again*.

(282)
```
# adjac(L,L1) -> true
@scop(noch,again,L,_)
@lhs_scop(mal_scop,L1,_,_).
```

Tests on Sentence-Initial and Sentence-Final Occurrence might be relevant for the translation of adverbs and particles, see the examples in (294) and (295) in Section 5.2.1, as well as for identifying greeting and exclamative formulas, see (285). For this, the test predicates in (283) and (284) are used. They say that the word that corresponds to the predicate with label `L` occurs in the very beginning (283) or in the very end (284) of an utterance.

(283) `sent_init(L)`
(284) `sent_final(L)`

(285) shows an example where sentence-initial and sentence-final occurrence are used to identify the German greeting formula *mach's gut*. If uttered with no further context it is translated into *take care*. The rule in (286) shows the transfer mapping that includes the test predicates in (283) and (284).

(285) *Mach's gut!*
Take care!

(286)
```
# arg3(L,I,I1),sent_init(L),sent_final(L1) <->
# arg3(L,I,I1),sent_init(L),sent_final(L1)
@verb(machen,take_v,L,I)
@rhs_np(udef,care_n,sg,L1,I1,G)
@lhs_arg_pron(arg3,third,event,sg,L,I,_,G)
@lhs_adx(gut_isect,_,I,G).
```

5.1.7 Prosodic Information

Prosodic information is available from the PROSODY slot of the VIT. We make use of prosodic accent for the disambiguation of German particles. Prosodic mood, i.e., intonation, is chiefly used to generate the appropriate English question or imperative structures for verb-initial German sentences. Consider the examples in (287) and (288) which are identical wrt. to the uttered words.

(287) Dann würde das *DOCH* gehen.
Then it would be possible, *after all*.

(288) Dann würde das *doch* gehen?
Then it would be possible, *wouldn't it?*

In (287), the particle *doch* occurs in the middle of an utterance with declarative mood and bears an accent. Here, it is translated into *after all*. It functions as a pointer to a previous dialog stage. Something that was impossible before turned out to be feasible at the utterance time. For the mapping of *doch* to *after all*, the rule in (289) contains constraints on its position, the prosodic mood of the utterance it occurs in, and its prosodic accent.

In (288), *doch* occurs in exactly the same context as in (287), but here it is not stressed and the utterance has a rising intonation. The particle signals the speaker's expectation of the hearer's approving response. In English, this meaning is conveyed by a question tag. The corresponding rule in (290) restricts the mapping to utterances with the intonation of a yes-no question, in which *doch* occurs unstressed in a non-sentence-initial position.

(289)
```
# not(init(L)),pros_accent(L),pros_mood(L,decl) -> true
@scop(doch,after_all,L,_).
```

(290)
```
# not(init(L)),not(pros_accent(L)),pros_mood(L,quest) -> true
@scop(doch,quest_tag,L,_).
```

5.2 Information from the Broader Context

There are cases in which the transfer component needs global information, such as that listed in (291), in order to choose a particular TL correspondence. Global information is obtained from the dialog processing and the context evaluation modules via interface predicates. A survey of problems that cannot be solved by the transfer module alone is given in [Buschbeck-Wolf, 1997].

(291)
- Antecedents of anaphors and ellipsis
- Illocution or dialog act
- Dialog stage
- Propositional utterance content
- Temporal referents
- Probability measures of particular co-occurrences

For anaphors and ellipses, the transfer requires the identification of their antecedents. On the one hand, the antecedent might be required for the appropriate translation of the anaphor or the ellipsis itself. On the other hand, information about antecedents might be necessary to verify constraints in the condition part of other transfer rules. The requests for the identification of an antecedent are generated automatically. They are triggered if the required information, such as the sort of a referential predicate, is lacking in the semantic construction output.

Dialog stage and propositional content information is exploited for lexical disambiguation. Illocutional information is often relevant for the translation of discourse particles and set phrases [Stede and Schmitz, 1997].

The identification of temporal referents is used, among others, to translate holiday names into dates if they are unknown in the culture of the addressee.

5.2.1 Dialog Information

The Dialog Act encodes the illocutional function of an utterance, which is sometimes helpful for disambiguation. The dialog act request is shown in (292). `L` is the label of the predicate in question, `DialogAct` is the dialog act of the utterance in which it occurs. (293) shows the request of the dialog act of the previous utterance. In both cases, a disjunctive specification of dialog act values is possible (indicated by the sign ";" between the values of `DialogAct`, see below).

(292) `dialog_act(L,DialogAct)`

(293) `preceding_da(L,DialogAct)`

Values of `DialogAct` `accept, clarify, clarify_answer, confirm, thank, convention, deliberate, deviate_scenario, digress, init, explained_reject, feedback, feedback_positive, feedback_negative, feedback_backchanneling, inform, give_reason, greeting, greeting_begin, suggest, greeting_end, introduce, politeness_formula, refer_to_setting, reject, request, request_clarify, request_comment,request_suggest`

For the definition of the dialog acts, I refer to [Alexandersson et al., 1997].

As an example, consider the translation of the particle *doch* in its sentence-initial occurrence in (294) and (295).

(294) *DOCH* das geht bei mir.
Yes, this would suit me.

(295) *Doch* das geht bei mir nicht.
But this doesn't suit me.

In (294), *doch* is stressed and expresses a positive feedback to a previously made suggestion. It emphasises the speakers attitude and might be translated into *yes*. The rule in (296) fixes the context for this translation, i.e., `sent_init(L)` tests the sentence-initial occurrence of the particle, and `pros_accent(L)` checks whether it bears prosodic accent. The dialog act `pos_feedback` is verified by the context evaluation module.

The meaning of *doch* changes if it is unstressed and comes along with a dialog act other than positive feedback, as in (295). Although being in the same position as in (294), it now denotes contrast, and is consequently translated into *but*. Rule (297) restricts the context of this mapping correspondingly.

(296)
```
# init(L),pros_accent(L),dialog_act(L,pos_feedback) -> true
@scop(doch,yes,L,_).
```

(297)
```
# init(L),not(pros_accent(L)),dialog_act(L,~pos_feedback) -> true
@scop(doch,but_scop,L,_).
```

Dialog Phase Request. For cases in which the information about the actual discourse stage allows the appropriate translation to be determined, the request in (298) is used.

(298) `dialog_phase(DialogPhase)`

Below, the values of `DialogPhase` are listed. For their definition, I refer to [Alexandersson et al., 1997].

Values of `DialogPhase` `hello, opening, negotiation, closing, good_bye`

The use of dialog stages as context restriction is illustrated with the example in (299). Both verbs *repeat* and *summarize* in (299) and (300) are possible translations of *wiederholen*. Usually, *summarize* is used to give a summary of a discussion or a part of it, while *repeat* means to do something again. In order to disambiguate *wiederholen*, its place of occurrence in the dialog can be utilized. While *summarize* is usually uttered at the end of a successive appointment scheduling circle, i.e., in the closing part, *repeat* can be used at any stage of the dialog. The rule for the translation of *wiederholen* into *summarize* in given in (301).

(299) Ja, gut, dann *wiederhole* ich jetzt nochmal.
All right, I'll *summarize*.

(300) Könnten Sie das bitte *wiederholen*? (Ich hab's nicht verstanden.)
Could you *repeat* that, please? (I didn't get it.)

(301)
```
# dialog_phase(L,closing) <-> true
@verb(wiederholen,summarize,L,_).
```

Test on the Propositional Utterance Content. Some ambiguities can be resolved by information about the propositional content of an utterance. It is requested by the predicate in (302).

(302) `prop_content(L,PropContent)`

`L` is the label of the predicate in question, `PropContent` is the propositional content of the utterance in which it occurs. The list of all values of `PropContent` is not yet available, but `location` and `time` are two of them.

The use of the information about the propositional content is shown with the ambiguity of the preposition *bei* in (303). Depending on the topic of negation the PP has either an experiencer meaning (translated into or *for you*), or it has a locative meaning (translated into *at your place*). To make a decision between these two meanings of *bei*, we state the rather special context of this ambiguity[7] and add the test on the propositional content, see (304). If a location is negotiated the locative interpretation is assigned, otherwise the experiencer reading is assumed to be appropriate.

(303) Ist es *bei Ihnen* möglich?
Is it possible *for you*?
Is it possible *at your place*?

(304)
```
# sort_check(I1,human),stat(L2,I),abstr_possible(L2,_),
pron(_,I1),prop_content(L,location) -> true
@prep(bei,loc_derived,L,I,I1).
```

5.2.2 Discourse Function of Particles

For particles, the context evaluation component calculates the discourse function, which is helpful for their translation or deletion. It is entered in the discourse slot and requested by the predicate in (305). `L` is the label of the particle in question.

(305) `disc_func(L,DiscFunc)`

Values of `DiscFunc`
```
structure, push, pop, check, edit, surprise,
exemplify, hesitate, coherence, known, revised,
attitude, positive, negative, indifferent,
uptake, smoothen, repair, prefer, emphasize
```

For explanation of these values, see [Stede and Schmitz, 1998].

An example is given in (306). If (in contrast to the examples (287), (288), (294) and (295)) the particle *doch* is used as a pure filler word, it is removed. In (306), it somehow smoothens the utterance, but it has no contribution to its propositional content. Since for transfer, it is impossible to exhaustively consider all combinations of contexts in which this might occur, we utilize the result of the discourse particle analysis in order to constrain the deletion of *doch*.

(306) Dann nehmen wir *doch* den Dienstag.
Then, let's take Tuesday.

---

[7]That is, the preposition *bei* has a pronominal internal argument. The *bei* PP is a modifier of the copula verb *sein* with a human subject and the adjectives *möglich* or *machbar* as predicative.

The rule in (307) shows the deletion of the particle *doch* if it is analyzed as a smoothening particle.

(307)
```
# disc_func(L,smoothening) -> true
@lhs_scop(doch,L,_,_).
```

5.2.3 Requests for External Disambiguation

If disambiguation requires either world knowledge or context information beyond the current utterance boundaries, we send the request in (308) to the context evaluation component. `L` is the label of the ambiguous predicate and `AmbigPred_Translation` a value consisting of the name of the predicate in question and its expected TL correspondence. If the TL reading of the SL word can be verified by the context evaluation module the corresponding mapping is executed.

(308)
```
disambig_lex(L,AmbigPred_Translation)
```

For example, in the utterance in (309), there are two words we cannot resolve in the local context. *Da* is translated either into *then* or *there*, and *Essen* corresponds either to *lunch* or *dinner*. The rules for these words in (310) and (311) are supplied with disambiguation requests that allow the corresponding mappings to be applied.

(309) *Da* können wir uns nicht zum *Essen* treffen.
We can't meet *then/there* for *lunch/dinner*.

(310)
```
# disambig_lex(L,da_isect_there) <-> true
@mod(da_isect,there,L,_).
```

(311)
```
# disambig_lex(L,essen_lunch) <-> true
@noun(essen_n,lunch,L,_).
```

5.3 Templates for Frequently Used Contextual Constraints

In order to treat frequently occurring restrictions more efficiently, I also introduce templates for conditions. They are called by the `+` operator and can be freely combined with the conditions introduced in Sections 5.1 and 5.2.

5.3.1 Sortal Restrictions on Arguments

The templates in (312) are used for argument sort checking.

(312a) `arg1_sort(L,I1,Sort) := arg1(L,I1,I2),sort_check(I2,Sort).`
(312b) `arg2_sort(L,I1,Sort) := arg2(L,I1,I2),sort_check(I2,Sort).`
(312c) `arg3_sort(L,I1,Sort) := arg3(L,I1,I2),sort_check(I2,Sort).`

Template head:	`L`	Label of the verb and the argument
	`I1`	Instance variable of the verb and the argument
	`I2`	Instance variable of the argument filler
	`Sort`	Sort or disjunctions of sorts

With the template in (312c), the rule for *erreichen* presented in (250) of Section 5.1.1 can be rewritten as shown in (313). Note that both the label and the instance variable have to be coindexed to combine the verb with the condition template.

(313)
```
# +arg3_sort(L,I,vehicle) <-> # +arg3_sort(L,I,vehicle)
@verb(erreichen,catch,L,I).
```

5.3.2 Specific Argument Realization

The concrete realization of an argument is often relevant for the translation of support verbs. For this purpose, the templates in (314) can be used.

```
(314a) arg1_noun(L,I1,I2,Noun) := arg1(L,I1,I2),Noun(_,I2).
(314b) arg2_noun(L,I1,I2,Noun) := arg2(L,I1,I2),Noun(_,I2).
(314c) arg3_noun(L,I1,I2,Noun) := arg3(L,I1,I2),Noun(_,I2).
```

Template head:	`L`	Label of the verb and the argument
	`I1`	Instance variable of the verb and the argument
	`I2`	Instance variable of `Noun`
	`Noun`	Noun that fills the argument role

As an example, consider the transfer rule in (315) that relates *ein Problem bereiten* to *cause a problem.* It is an abbreviated version of the rule presented in (266) of Section 5.1.3.

(315)
```
# +arg3_noun(L,I,_,problem) <-> # +arg3_noun(L,I,_,problem)
@verb(bereiten,cause_v,L,I).
```

5.3.3 Pronominal Argument Realization

If we remove or insert reflexive pronouns, we often have to check coreference features with pronominal arguments, see (109) in Section 4.2.6. For this, it is useful to have templates such as those in (316) that shorten the writing of the condition.

```
(316a) arg1_pron(L,I1,I2,D1,D2) :=  arg1(L,I1,I2),pron(_,I2),
                                    prontype(I2,D1,D2).
(316b) arg2_pron(L,I1,I2,D1,D2) :=  arg2(L,I1,I2),pron(_,I2),
                                    prontype(I2,D1,D2).
(316c) arg3_pron(L,I1,I2,D1,D2) :=  arg3(L,I1,I2),pron(_,I2),
                                    prontype(I2,D1,D2).
```

Template head:	`L`	Label of the verb and the argument
	`I1`	Instance variable of the verb and the argument
	`I2`	Instance variable of `pron`
	`D1`	Reference designator of `pron`
	`D2`	Sort designator of `pron`

The use of the template in (316a) allows us to rewrite the rule that was presented in (109) as shown in (317).

(317)
```
# +arg1_pron(L,I,I2,D,std),num(I2,pl) <->
# +arg1_pron(L,I,I2,D,std),num(I2,pl), in_g(L,G)
@verb(sehen,meet,L,I)
@lhs_arg_pron(arg3,D,refl,pl,L,I,_,_,G).
```

5.3.4 Propositional Argument Realization

Another useful abbreviation of conditions is that for propositional argument realization, shown in (318).

(318)
```
proposit_arg3(L,I) := arg3(L,I,H),(label(H);hole(H)).
```

Template head:	L	Label of the verb and the argument
	I	Instance variable of the verb and the argument

The use of this template is illustrated with the rule for the translation of *finden* into *think* in (319) which is yet another formulation of the rule in (261) in Section 5.1.1.

(319)
```
# +proposit_arg3(L,I) <-> true
@verb(finden,think,L,I).
```

5.3.5 Test on Situation Anaphors

Another frequently used condition is the test on a situation anaphor, i.e., *das* and *es* in German, and *this, that* and *it* in English. To avoid an explicit disjunctive specification of the particular pronoun type values, I provide the templates in (320) for German situation anaphors, and (321), for English situation anaphors. I is the instance variable of the anaphor.

(320)
```
g_sit_anaphor(I) :=  prontype(I,third,demon);
                     prontype(I,third,event).
```

(321)
```
e_sit_anaphor(I) :=  (abstr_nom(_,I),(this(_,I,_,_);that(_,I,_,_)));
                     prontype(I,third,event).
```

An example is given in (323). If *genauso* is used as predicative of the copula with a situation anaphor in the subject role, it corresponds to *the same*, see (322).

(322) Das ist bei mir *genauso*.
That is *the same* for me.

(323)
```
# stat(L,_),+g_sit_anaphor(I) <-> true
@adxTOmod_pp(genauso,[equal,no,def,same,abstr_nom,sg],L,I,_,_).
```

5.3.6 Test on Semantic Classes

The tests on semantic classes, see Table 4 in Section 5.1.4, can be shortened with the templates given in (324).

```
(324a) is_verb(I)      := rel(verb,_,I).
(324b) is_noun(I)      := rel(noun,_,I).
(324c) is_prep(I1,I2)  := rel(mod,_,I1,I2).
(324d) is_mod(I)       := rel(mod,_,I);card(_,I,_);ord(_,I,_).
(324e) is_quant(I)     := rel(quant,_,I,_,_).
(324f) is_grad(L)      := rel(mod,L,I),dim(_,I).
```

Examples are shown in (325) and (326). They correspond to the rules presented in (271) and (272) in Section 5.1.4.

(325)
```
# +is_noun(I) <-> true
@mod(anschliessend,following,_,I).
```

(326)
```
# +is_verb(I) <-> true
@adxTOpp(anschliessend,[temp_post,no,that,abstr_nom,sg],_,I,_,_).
```

5.3.7 Test on Temporal Relational Nouns

Certain relational nouns are not assigned a sort of their own. They inherit the sort of their related noun. For the translation of prepositions we often need to know whether a relational noun has a temporal interpretation. This constraint is abbreviated by the template in (327). `I` is the instance variable of the relational noun that is represented by the type `relnoun` which subsumes all nouns of this sort.

(327)
```
temp_relnoun(I) := relnoun(L,I),arg3(L,I,I1),sort_check(I1,temporal).
```

An example is given in (328). The preposition *an* is mapped to the meaning abstraction `temp_loc`, see Section 3.2, if its internal argument is a relational noun with a temporal interpretation. The corresponding rule is shown in (329).

(328) *Am* Ende der Woche bin ich in Berlin.
At the end of the week I'll be in Berlin.

(329)
```
# +temp_relnoun(I) -> true
@prep(an,temp_loc,_,_,I).
```

5.3.8 Test on Directional Verbs

For the translation of English prepositions and certain adverbs into German, we need to know whether they are used directionally or not in order to generate the appropriate case of the German internal argument. In English, there is no case marking for directionality as in German, where the use of dative with case alternating prepositions usually gives a stative interpretation, and accusative a directional one. In English, the sort of the external argument gives us the desired information. It is requested by the templates in (330). It tests the sort of the verb or noun that is modified by the preposition. The sort `move_sit` stands for movements (verbs and nouns). `I` is the instance variable of the external argument of the preposition.

(330a)
```
dir_verb(I)    := +is_verb(I),sort_check(I,move_sit).
```
(330b)
```
nondir_verb(I) := +is_verb(I),sort_check(I,~move_sit).
```

Consider the examples in (331) and (332). The directionality information is required in order to generate the appropriate case for the NP *meine Tasche.*

(331) I'll put the documents *in my briefcase.*
Ich stecke die Dokumente *in meine Tasche.*

(332) I have the documents *in my briefcase.*
Ich habe die Dokumente *in meiner Tasche.*

The refinement rules in (333) maps the English preposition *in* to the meaning abstraction `loc_int` (localization in the interior of something). Depending on the sort of the external argument the directionality value `dir` is set to `yes` or `no`.

(333a)
```
dir(L,_) # sort_check(I1,thing),+dir_verb(I) -> dir(L,yes)
@prep(in,loc_int,L,I,I1).
```

(333b)
```
dir(L,_) # sort_check(I1,thing),+nondir_verb(I) -> dir(L,no)
@prep(in,loc_int,L,I,I1).
```

References

[Abney, 1996] Steven Abney. 1996. Partial parsing via finite-state cascades. In *Proceedings of the ESSLLI '96 Robust Parsing Workshop.*

[Alexandersson et al., 1997] Jan Alexandersson, Bianka Buschbeck-Wolf, Tsutomu Fujinami, Elisabeth Maier, Norbert Reithinger, Birte Schmitz, and Melanie Siegel. 1997. Dialogue Acts in VERBMOBIL-2. Verb*mobil* Report 204, DFKI GmbH, Universities of Berlin, Saarbrücken and Stuttgart, May.

[Becker et al., 1997] T. Becker, W. Finkler, and A. Kilger. 1997. Generation in Dialog Translation: Requirements, Techniques, and their Realization in Verbmobil. Submitted to "Computational Linguistics".

[Block and Ruland, 1998] Hans Ulrich Block and Tobias Ruland. 1998. The Verb*mobil* Stochastic LR-Parser. Siemens AG, Munich, Germany. Undocumented software.

[Bos and Schiehlen, 1997] J. Bos and M. Schiehlen. 1997. Klassifikation der deutschen Partikeln in Verbmobil. Verb*mobil* Memo XX, Universities of Saarbrücken and Stuttgart, July.

[Bos et al., 1998] Johan Bos, C.J. Rupp, Bianka Buschbeck-Wolf, and Michael Dorna. 1998. Managing Information at Linguistic Interfaces. In *Proceedings of the 17th International Conference on Computational Linguistics (Coling-ACL'98)*, pages 160–166, Montréal, Canada, August.

[Bos, 1996] Johan Bos. 1996. Predicate Logic Unplugged. In *Proceedings of the 10th Amsterdam Colloquium*, pages 133–143, University of Amsterdam. ILLC/Department of Philosophy.

[Bub et al., 1997] T. Bub, W. Wahlster, and A. Waibel. 1997. Verb*mobil*: The Combination of Deep and Shallow Processing for Spontaneous Speech Translation. In *Proceedings of the 22nd International Conference on Acoustics, Speech, and Signal Processing (ICASSP'97)*, Munich, Germany, April.

[Buschbeck-Wolf and Dorna, 1997] B. Buschbeck-Wolf and M. Dorna. 1997. Using Hybrid Methods and Resources in Semantic-based Transfer. In *Proceedings of the 2nd International Conference "Recent Advances in Natural Language Processing" (RANLP'97)*, Tzigov Chark, Bulgaria, September.

[Buschbeck-Wolf and Dorna, 1998] Bianka Buschbeck-Wolf and Michael Dorna. 1998. Quality and Robustness in MT - a Balancing Act. In *Machine Translation and the Information Soup: Proceedings of the 3rd International Conference of the Association for Machine Translation in the Americas (AMTA-98)*, Langhorne, PA.

[Buschbeck-Wolf and Tschernitschek, 1996] B. Buschbeck-Wolf and C. Tschernitschek. 1996. What You Always Wanted to Know about Semantic Transfer. Verb*mobil* Report 114, IBM Deutschland Informationssysteme GmbH, Heidelberg, Germany.

[Buschbeck-Wolf, 1997] Bianka Buschbeck-Wolf. 1997. Resolution on Demand. Verb*mobil* Report 196, IMS, Universität Stuttgart, Germany.

[Copestake et al., 1995] A. Copestake, D. Flickinger, R. Malouf, S. Riehemann, and I. Sag. 1995. Translation using Minimal Recursion Semantics. In *Proceedings of the 6th International Conference on Theoretical and Methodological Issues in Machine Translation (TMI'95)*, Leuven, Belgium.

[Dorna and Emele, 1996a] Michael Dorna and Martin C. Emele. 1996a. Efficient Implementation of a Semantic-based Transfer Approach. In *Proceedings of the 12th European Conference on Artifical Intelligence (ECAI'96)*, pages 567–571, Budapest, Hungary.

[Dorna and Emele, 1996b] Michael Dorna and Martin C. Emele. 1996b. Semantic-based Transfer. In *Proceedings of the 16th International Conference on Computational Linguistics (Coling'96)*, pages 316–321, Copenhagen, Denmark.

[Dorna, 1998a] M. Dorna, 1998a. *Documentation of the IMS Transfer Rule Compiler: Formalism, Compilation, etc.* ims.

[Dorna, 1998b] Michael Dorna. 1998b. The ADT-Package for the Verb*mobil* Interface Term. Verb*mobil* Report 104X, IMS, Universität Stuttgart, Germany. Available via URL `http://www.ims.uni-stuttgart.de/projekte/verbmobil/vitADTX/`.

[Kasper et al., 1997] Walter Kasper, Johan Bos, Michael Schiehlen, and Christine Thielen. 1997. Definition of Abstract Semantic Classes. Verb*mobil* Technical Report 61, DFKI GmbH, Saarbrücken, Germany and Universität des Saarlandes, Germany and IMS, Universität Stuttgart, Germany, August.

[Kay et al., 1994] M. Kay, M. Gawron, and P. Norwig. 1994. *Verbmobil: a Translation System for Face-to-Face Dialogs.* Number 33 in CSLI Lecture Notes. University of Chicago Press.

[Kroner and Schwinn, 1997] Hans-Jörg Kroner and Johannes Schwinn. 1997. Fallbasierte Übersetzung. DFKI GmbH, Kaiserslautern, Germany. Undocumented software.

[Light, 1996] M. Light. 1996. CHUMP: Partial Parsing and Underspecified Representations. In *Proceedings of the ECAI-96 Workshop: Corpus-Oriented Semantic Analysis*, pages 28–30, Budapest, Hungary.

[Nirenburg, 1995] Sergei Nirenburg. 1995. The PANGLOSS Mark III Machine Translation System. CMU-CMT-95-145, Joint Technical Report by NMSU CRL, USC ISI and CMU CMT.

[Reyle, 1993] Uwe Reyle. 1993. Dealing with Ambiguities by Underspecification: Construction, Representation and Deduction. *Journal of Semantics*, 10(2):123–179.

[Schiehlen, 1998] Michael Schiehlen. 1998. Robust Semantic Construction on Partial Syntactic Analyses. IMS, Stuttgart University, Germany. Undocumented software.

[Stede and Schmitz, 1997] M. Stede and B. Schmitz. 1997. Discourse Particles and Routine Formulas in Spoken Language Translation. In *Proceedings of the ACL/EACL'97 Workshop on Spoken Language Translation*, Madrid.

[Stede and Schmitz, 1998] M. Stede and B. Schmitz. 1998. Partikeln: Diskursfunktionen für die Übersetzung. Verb*mobil* Memo XX, Technical University of Berlin, July.

[Tillman et al., 1997] C. Tillman, S. Vogel, H. Ney, H. Sawaf, and A. Zubiaga. 1997. Accelerated DP-based Search for Statistical Translation. In *Proceedings of the 5th European Conference on Speech Communication and Technology (EuroSpeech'97)*, pages 2667–2670, Rhodes, Greece, September.