TERM-REWRITING AS A BASIS FOR A UNIFORM ARCHITECTURE IN MACHINE TRANSLATION

Wilhelm WEISWEBER
Technical University of Berlin
Institute for Software and Theoretical Computer Science
Project KIT-FAST, Sekr. FR 5-12
Franklinstr. 28/29, D-1000 Berlin 10
E-mail: ww@kit.cs.tu-berlin.de or weisweb@tubvm.cs.tu-berlin.de

Abstract

In machine translation (MT) different levels of representation can be used to translate a source language sentence onto its target language equivalent. These levels have to be related to each other. This paper describes a declarative formalism on the basis of term-rewriting which maps one representation onto an equivalent adjacent one. The different levels (e.g. represented by derivational trees, feature structures or expressions of a knowledge representation language) can be represented as terms. The equivalences between them are stated as axioms which are directed to form a non-confluent and terminating term-rewrite system. A complete and coherent algorithm has been developed which interprets these systems and is able to handle default rules.

1 Introduction

In general there are different models of machine translation (MT). Regardless of the model used as the basis for an MT system, the architecture looks like the following (see [Arnold et al. 86] and [Sharp 88]):

$$\begin{tikzcd}
 & G_1 \arrow[d, dash] & G_2 \arrow[d, dash] & & G_{n-1} \arrow[d, dash] & G_n \arrow[d, dash] & \\
S_S \arrow[r, "{}_ST_1"'] & R_1 \arrow[r, "{}_1T_2"'] & R_2 \arrow[r, "{}_2T_3"'] & \dots \arrow[r, "{}_{n-2}T_{n-1}"'] & R_{n-1} \arrow[r, "{}_{n-1}T_n"'] & R_n \arrow[r, "{}_nT_T"'] & S_T
\end{tikzcd}$$

The R_i's are representations, e.g. derivational trees (for example the syntactic stuctures) or directed acyclic graphs (for example f-structures of LFG or KL-ONE based conceptual representations), the G_i's are the formalisms which generate these representations, e.g. context free grammars or signatures, the T_i's are mappings from one representation to an equivalent adjacent one and S_S, S_T are the source and target language sentences, respectively.
If the MT system is interlingua based, one of the R_i is the interlingua and if it is transfer based one of the ${}_iT_{i+1}$ is the transfer system. Usually the first and last mappings have different status and are realized by a parser and a generator, respectively.
The MT system of our project KIT-FAST[1] is based on a transfer model and i = 4, where R_1 is the source GPSG structure (see [Gazdar et al. 85] and [Busemann/ Hauenschild 88]), R_2 is the source Functor-Argument-Structure (FAS)[2], R_3 is the target FAS and R_4 is the target GPSG structure. G_1 to G_4 are context-free grammars. In the current phase of our project we are taking the first steps towards the solution of textual phenomena, i.e. the interpretation of anaphorical relations (see [Schmitz et al. 91]). In order to achieve this, the source FAS is mapped onto a conceptual representation for the text content, which is represented by the ABox (assertional knowledge) of the KL-ONE like representation language BACK. The knowledge representation system BACK has been developed by our neighbour project KIT-BACK (see [Peltason et al. 89]). The representation for the text content is used to determine the discourse consistency of possible antecedents for anaphoric pronouns.
Currently ${}_ST_1$ and ${}_4T_T$ are realized by a GPSG parser and a morphological synthesis component, respectively. The mappings ${}_1T_2$ (semantic analysis), ${}_2T_3$ (transfer), ${}_3T_4$ (generation) and the mapping from R_2 (FAS representations) onto conceptual representations are realized by one algorithm on the basis of term-rewriting. The mapping ${}_ST_1$ (parsing) also is intended to be implemented with a term-rewrite system. A short introduction to term-rewriting is given in section 3.
It seems reasonable to represent all different R_i with the help of one data structure and to specify all ${}_iT_{i+1}$ with the help of the same formalism (including parsing and generation). Some proposals in this direction have already been made. Two of these systems, namely CAT2 and TFS, and their properties are outlined in section 2. The following sections present an alternative approach, which remedies some problems of these systems. In Section 4 a term representation, which is generated by a signature for a term algebra, is introduced with the help of which all R_i can be represented. The representation of GPSG or FAS derivational trees, feature structures and KL-ONE like conceptual structures as terms is shown by example. The algorithm, i.e. the termination condition and the application relations, which are automatically computed from the rewrite rules by the examination of the interdependencies of the rules with the help of

[1] KIT = Künstliche Intelligenz und Textverstehen (artificial intelligence and text understanding), FAST = functor-argument-structure for translation. The project KIT-FAST constitutes the Berlin component of the complementary research of EUROTRA-D. It receives grants by the Federal Minister for Research and Technology.

[2] The FAS is a semantic representation for sentences which has been developed in the preceding phase of our project. Among others it contains functor-argument-relations, information about the thematic structuring of sentences and semantic relations (argument roles) and semantic features. For a more detailed description see [Hauenschild/ Umbach 88], [Busemann/ Hauenschild 89] and [Busemann 90].

superposition, is given in section 5. Section 6 concludes the paper and gives an outlook for further research.

2 Other approaches

Several proposals in the direction of a uniform architecture of MT systems have been made. Within the EUROTRA-D project the CAT2 system was developed by [Sharp 88]. This approach uses derivational trees as representations and is characterized by the compositionality of the mapping rules according to [Arnold et al. 86]. The Typed Feature Structure (TFS) system as presented in [Emele/ Zajac 89] and [Zajac 89,90,91] is outlined in subsection 2. It uses typed feature structures as representations.
Another approach, which is similar to Emele and Zajac's, is given in [Russell et al. 91].

2.1 The CAT2 system

The representations R_i in the CAT2 system are derivational trees generated by context-free grammars G_i which are made up of a pair $\langle C_i, A_i \rangle$, where C is a set of Constructors (structural rules) and A a set of Atoms (lexical rules). The mappings from one tree to another are called Translators, which are sets of t-rules. The translators ${}_S T_1$ and ${}_n T_T$ are realized by a parser and a tree-to-string transducer, respectively. The interpreter for t-rules processes the input tree top-down by matching the input tree with the left-hand side (lhs) of a rule and the subobjects are recursively mapped. On the way bottom-up the subtrees are reordered according to the right-hand side (rhs) of the given t-rule. The interpreter terminates when the whole input structure has been traversed and mapped. The t-rules have the form lhs $\Rightarrow$ rhs. The lhs and rhs are structural descriptions of the source and target structure, respectively, which are expressed by trees of the form $\langle$node$\rangle$.$\langle$subtrees$\rangle$. The nodes are pairs (C, F), where C is a distinguished feature and F is a set of feature-value pairs. The subtrees of the lhs and rhs can be combined with the help of conjunction or disjunction. Each subtree can be labeled with a tag $N and can be marked as optional or with the Kleene-star operator. When a tag, which occurs on the lhs, is missing on the rhs, the corresponding subtree is deleted, otherwise it is replaced by its mapping (translation). The t-rules maintain the partial compositionality[3] of the translator. An example of a t-rule for semantic analysis is the mapping from the surface cases of the English verb *generate* onto the corresponding deep cases:

```
(s,{}).[$1:(np,{cas=nom}),
        (vp,{}).[(v,{stem=generate}).[],
                 $2:(np,{cas=acc})]]
⇒
(_,{}).[(process,{lu=generate}),
        (agent,{}).[$1],
        (affected,{}).[$2]]
```

The advantages of CAT2 are its simplicity and efficiency. Furthermore CAT2 is supposed to be reversible. The efficiency is a consequence of the (partial) compositionality of the translators. This leads directly to an efficient structure-driven[4] interpreter. The disadvantage is that it does not allow for directed acyclic graphs (DAGs) as representations. Since the translators are compositional, the t-rules cannot express the fact that the translation of one constituent depends on the translation of other constituents. For example if a predicate (verb, noun or adjective) is to be translated, the translation typically depends on the semantic roles and features (selectional restrictions) of its target language arguments.

2.2 The TFS system

The TFS system uses typed feature structures (TFSes) as representations, which can be represented as DAGs. TFSes are defined recursively. A TFS is an atomic or complex type. An atomic type consists of a type symbol and a complex type of a type symbol with a set of pairs of features and TFSes (values). The feature structures of PATR, LFG or HPSG are examples of TFSes. The set of type symbols P, which always includes the special type symbols $\top$ (top) and $\bot$ (bottom), is partially ordered and $\top \geq T \geq \bot$ holds for all $T \in P$. This partial ordering defines a lattice structure on P and can be extracted from the definitions (axioms). Definitions have the form $T = TFS_1 \vee \ldots \vee TFS_m$:- C, where T is a type symbol, the TFS_i are TFSes of type F_i ($T \geq F_i$) and C is a conditional constraint, which may be omitted and is expressed by a logical conjunction of TFSes. The unification of two type symbols is their greatest lower bound. A "rewrite step"[5] on an input TFS is performed by looking for a subTFS of type T of the input TFS and a definition of the form T = F :- C. In that case the subTFS of the input TFS and the TFS F are unified, the conjunction C of TFSes is solved and the result of the unification is inserted in place of the subTFS. The TFS rule for the semantic analysis rule of the previous subsection is:

```
SYN-S =
SEM-S[syn: S[np: NP[cas:NOM|X],
             vp: VP[v:GENERATE,
                    np: NP[cas:ACC|Y]]
      sem: REL[process: GENERATE,
               agent: X',
               affected: Y']]
:- SYN-NP[syn: NP[cas:NOM|X], sem: X'],
   SYN-NP[syn: NP[cas:ACC|Y], sem: Y']
```

The advantage of TFS is that all different levels, i.e. S_S, R_1 to R_n and S_T, are accessible for all mappings ${}_i T_{i+1}$. The disadvantage is that the algorithm for the application of definitions is not complete.

[3] The translation of an expression consists of the translation of its subexpressions (see [Arnold et al. 86]).

[4] 'Structure-driven' means that the input structure is processed in a certain strategy (in this case top-down). Structure-driven processes are normally more efficient than data-driven processes, which process the input structure according to the rules which manipulate it.

[5] This is not 'rewriting' in the sense of this paper, but the input TFS is extended by unification, which is a monotonic operation, i.e. it is "blown up" with additional information.

Another problem of TFS can arise if the definitions are applied in the wrong order. Let us for example assume that a predicate has alternative translations depending on the selectional restrictions of its target language arguments and the definition which gives the wrong translation is applied before the target language arguments exist. This would lead to a wrong or no translation, even if the correct translation would be possible. This can only be avoided if the alternatives are specified in one definition with the help of disjunction, which, however, may be very inefficient if the correct translation is the last alternative.

3 Term-rewriting

A term-rewrite system (TRS)[6] is a set of term-rewrite rules (TR rules) $\lambda \rightarrow \rho$ with left-hand side (lhs) λ and right-hand side (rhs) ρ, in which cospecified variables occur. The applicability of a TR rule to an input term t is checked by superposing t with λ.

Definition*: *Superposition
The term t_1 is superposable with the term t_2, iff a subterm t_1/u of t_1 and t_2 are unifiable with minimal unifier (or substitution) $\sigma \neq \{\}$.

The elements of σ are pairs of substitutions $X \leftarrow t_x$, where the variable X is substituted by the corresponding term t_x. The substitution of variables occurring in t/u and λ according to σ (notation: $(t/u)\sigma$ and $\lambda\sigma$, respectively) yields two identical terms ($(t/u)\sigma = \lambda\sigma$). If the term t is superposable with λ at the subterm t/u with the substitution σ, the TR rule is applied by replacing t/u in t by $\rho\sigma$ yielding the target term $t' = t[u \leftarrow \rho\sigma]$. This is called a derivation step (notation: $t \rightarrow t'$).
Originally TRSes are used to prove the equality of terms. In this context the Knuth-Bendix algorithm has been developed (see [Knuth/ Bendix 70]), which computes the normal form of a given TRS, if the TRS is confluent and terminating.
A TRS is confluent if the application of the rewrite rules to an input term yields exactly one target term, no matter in which sequence the rules are applied.
In order to guarantee the termination of TRSes, an ordering on the corresponding terms has to be defined and at least one minimal term exists. Such an ordering guarantees the termination of a derivation sequence $t_1 \rightarrow t_2 \rightarrow \ldots \rightarrow t_{n-1} \rightarrow t_n$ if and only if $t_1 > t_2 > \ldots > t_{n-1} > t_n$, where t_1 is the input term, t_n the target term and $t_i \rightarrow t_{i+1}$ a derivation step, in which the resulting term t_{i+1} is derived from the original term t_i by the application of one TR rule.
In order to prove the termination of a TRS, theoretically all possible derivation sequences have to be checked. Another possibility is to define a total ordering on the terms on the lhs and rhs of the TR rules, which guarantees that in a derivation step $t_i \rightarrow t_{i+1}$ the original term t_i is reduced according to this ordering ($t_i > t_{i+1}$), then the TRS is terminating because after a finite number of derivation steps either a minimal term is reached or no TR rule is applicable to the resulting term.
In order to guarantee the termination of TRSes by considering each single TR rule, some criteria have to be defined for the terms on the lhs and rhs of each single TR rule so that its application reduces the input term according to the total ordering. The ordering may reduce the size of the input term after the application of a rule (a quantitative ordering, i.e. deleting a subterm on the rhs) or substitute a subterm of the input term in such a way that this substitution is never reversed by another rewrite rule (a qualitative ordering, i.e. an operator precedence ordering). For the use in MT we assume a qualitative ordering (see section 5.5). For details about the termination of TRSes see [Dershowitz 82 and 85].
TRSes in normal form are complete and coherent. They can efficiently be applied to deduce the normal form of an input term. In order to prove the equality of two terms, their normal forms are deduced and compared for literal equality.
In order to use TRSes for mappings between a source and a target representation in MT, the source representation can be viewed as an input term for a TRS and the target representation as its normal form. For this reason a term algebra for each representation R_i has to be defined which generates the corresponding terms for the given representation. The mapping rules are considered as TR rules. But using TRSes for the mappings in MT causes a problem. Normally analysis, transfer and generation of natural languages in MT may have more than one result, i.e. TRSes used for mappings in MT usually are not confluent. For this reason an interpreter for TRSes has been developed in our project (see section 5, [Weisweber 89] and [Weisweber/ Hauenschild 90]), which is complete and coherent and applies terminating and non-confluent TRSes in a very efficient way.

4 The term representation

In order to have one process for the interpretation of the mapping rules of the different ${}_nT_{n+1}$, all structures R_n have to be represented with the help of one data structure. The data structure used by the TRS interpreter consists of terms which represent directed acyclic graphs (DAGs) with complex categories as node labels. Derivational trees are special instances of DAGs in which no re-entrancy of nodes is allowed and the edges leaving one node are ordered. The terms are generated by the following signature[7]:

Signature for DAGs

dag: CAT LIST $\rightarrow$ DAG
list: DAG LIST $\rightarrow$ LIST
[]: $\rightarrow$ LIST

At present this signature is fixed for the interpreter, but if more expressive representations are necessary for

[6] In the following a notation according to [Huet/ Oppen 80] is used, which gives a detailed introduction to TRSes.

[7] Signatures are very similar to context-free rules. The operator definitions 'op': $S_1 \ldots S_{n-1} \rightarrow S_n$ can be viewed as the context-free rule $S_n \rightarrow$ 'op(' S_1 ',' ... ',' S_{n-1} ')', where the sorts S_i are interpreted as non-terminals.

MT, the interpreter can be adapted. The sort CAT, which represents complex categories, is also generated by a signature which has to be specified for the particular representations.

Scheme for signatures for complex categories

```
C: LANG F_1 ... F_n → CAT
g-gpsg, g-fas, e-fas, e-gpsg, g-atl, ...: → LANG
V_i: → F_i
```

The C are the main categories (in the sense of the distinguished features of [Sharp 88]). LANG is a special sort which represents the language to which a category belongs. With the help of the instances of this sort the categories occurring in terms can be distinguished to belong to the source or target representation. This fact is used to allow the TRS interpreter to process the input structure in a flexible strategy (see section 5.4) and to guarantee its termination (see section 5.5).
The sorts F_i represent the features which are associated to the main category. The V_i are 0-ary operators (constants) which represent the values for the features represented by F_i ($V_i \in F_i$). Nodes N of a DAG which have no out-going edges (in trees terminal nodes) are represented by dag(N,[]).

Example signature for German nominal phrases

```
np: LANG PER PLU GEN CAS → CAT
g-gpsg: → LANG
1,2,3: → PER
+,-: → PLU
fem,masc,neut: → GEN
nom,gen,dat,acc: → CAS
```

An example of a term for a German nominal phrase is np(g-gpsg,3,-,fem,nom).
The TRS interpreter uses the signature for DAGs to traverse the input representation in order to find a subterm which is unifiable with the lhs of a TR rule. Our TRS editor uses the signature for DAGs to produce terms from a graphical input and to perform consistency checks on the input.
In order to show, for example, how conceptual structures can be represented as terms, a small fragment of the syntax of the ABox tell language (ATL) of the knowledge representation system BACK (see [Peltason et al. 89]), which is used in the experimental MT system of our project, is given:

Context-free rules for a fragment of the ATL:

```
⟨abox-tell⟩ → ⟨obj-ref⟩ = ⟨atl-conc⟩
            | ⟨variable⟩ = ⟨atl-conc⟩
⟨obj-ref⟩ → uc_i
⟨atl-conc⟩ → ⟨concept⟩
            | ⟨concept⟩ with ⟨atl-role⟩
⟨atl-role⟩ → ⟨role⟩ : (⟨abox-tell⟩)
            | ⟨atl-role⟩ andwith ⟨atl-role⟩
```

The non-terminal ⟨concept⟩ represents the concepts used in a discourse and ⟨role⟩ represents the semantic roles of the arguments of a predicative concept. ⟨variable⟩ represents variables which are instantiated with a new unique discourse object reference uc_i, if there is no object reference in the ABox for the given concept.

In order to represent ATL expressions as terms the following signature is used:

Signature for ATL categories:

```
equal: LANG OBJ-REF → CAT
with: LANG ROLE → CAT
g-atl: → LANG
```

The ATL expressions are represented by trees and since terms represent DAGs, a context-free syntax is needed to check whether the target terms of the conceptual analysis represent ATL trees or not. The context-free syntax of ATL trees is given in the following.

Context-free syntax for ATL trees

```
atl(g-atl) → equal(g-atl,_) with(g-atl,_)*
with(g-atl,_) → equal(g-atl,_) with(g-atl,_)*
equal(g-atl,_) → ⟨concept⟩ (lexical rules)
```

For example the German ATL expression uc_1 = generate with agent : (uc_2 = generator) andwith affected : (uc_3 = sentence) is represented by the following ATL term:

Example for an ATL term

```
dag(atl(g-atl), [
   dag(equal(g-atl,uc_1), [
      dag(generate, [])]),
   dag(with(g-atl,agent), [
      dag(equal(g-atl,uc_2), [
         dag(generator, [])])]),
   dag(with(g-atl,affected), [
      dag(equal(g-atl,uc_3), [
         dag(sentence, [])])])])
```

At the end of this section we give the TR rule of the semantic analysis which corresponds to the rules of CAT2 and TFS presented in the sections 2.1 and 2.2, respectively:

```
dag(s(e-gpsg), [
   dag(v-pred(e-fas,nom-acc,active), [
      dag(generate, [])]),
   dag(term(e-fas,nom), X),
   dag(term(e-fas,acc), Y)])
⇒
dag(clause(e-fas), [
   dag(v-pred(e-fas,ag-af,active), [
      dag(generate, [])]),
   dag(term(e-fas,agent), X),
   dag(term(e-fas,affected), Y)])
```

The TR rule contains the cospecified variables X and Y. Additionally some conditions on variable feature values can be defined with the help of the operators =, ≠, < and ≥ which can be combined with the logical operators *and* or *or*.
In order to handle for example the long distance dependencies of GPSG or LFG conveniently, the expressive power of TR rules has been increased. The categories occurring on the lhs and rhs of a rule may be labeled with the +-operator, which is similar to the Kleene-star operator. The occurrence of the category C^+ means that C is the root node of the corresponding subDAG, which may dominate another category C, which again may dominate another category C and so

on. This is similar to functional uncertainty in LFG.

5 Interpretation of TRSes

As mentioned in section 3, the TRSes for the mappings in MT are not usually confluent. For this reason the Knuth-Bendix algorithm cannot be used. In order to apply non-confluent TRSes efficiently, the interdependencies between their TR rules have to be determined. The rewrite process is data-driven and in order to check each TR rule only once for application, an order is computed (subsection 1). The rewrite process should be complete and coherent. Therefore more general TR rules should be checked for application after more specific ones and subsection 2 shows an order for some kind of default TR rules. As the TRSes are not confluent, the interpreter has to control the branching of the derivation sequence. This is done with the help of alternative rules (subsection 3). The TRS interpreter is outlined in subsection 4. Since the rewrite process is data-driven, the termination of the interpreter cannot be guaranteed by the interpreter itself. Subsection 5 gives an adequate termination condition for TRSes used in MT systems.
In the following subsections, the existence of two TR rules $\langle n_1, \lambda_1 \rightarrow \rho_1 \rangle$ and $\langle n_2, \lambda_2 \rightarrow \rho_2 \rangle$ in the TRS is assumed, where $\{n_1, n_2\} \subseteq R_N$ (the set of numbers of all TR rules), λ_1, λ_2 are the left-hand sides (lhs) and ρ_1, ρ_2 are the right-hand sides (rhs) of the TR rules.

5.1 The application order

In order to check each TR rule for application only once, an order has to be computed. Generally there are cycles in the application order and the TR rules of a cycle have to be checked more than once.

***Definition*: *Application order relation* $>_{app}$**
If ρ_1 is superposable with λ_2 **or**
λ_2 is superposable with ρ_1
then $n_1 >_{app} n_2$, where $>_{app} \subseteq R_N \times R_N$.

The relation $>_{app}$ is transitive and $n_1 >_{app} n_2$ means that TR rule n_1 has to be applied before rule n_2. This relation may have cycles $n_1 >_{app} \ldots >_{app} n_m >_{app} n_1$. In order to compute the cycles of $>_{app}$, the transitive closure $>^{+}_{app}$ is computed, which may contain an equivalence relation $>_{cyc}$ ($>_{cyc}$ is reflexive, symmetric and transitive, $>_{cyc} \subseteq >^{+}_{app}$ and $>_{cyc} \subseteq R_C \times R_C$, where $R_C \subseteq R_N$ is the set of numbers of cyclic TR rules).

***Definition*: *Cycles of* $>_{app}$**
A cycle of $>^{+}_{app}$ is an equivalence class $[n] = \{m \mid n >_{cyc} m\}$ and $>_{cyc}$ is the greatest equivalence relation in $>^{+}_{app}$.

The cycles (equivalence classes) are either equal or disjoint and constitute a partition of R_C. The efficiency of the rewrite process crucially depends on the number and size of the cycles.

5.2 Default TR rules

In some situations it is useful to have some kind of default TR rule. For example if there are several different translations for one source language terminal which depend on certain (structural) conditions and there is a "default" translation, if none of these conditions holds, e.g. if the German verb *schwimmen* has an inanimate argument, it has to be translated into the English *float* and if there is no information available, it has to be translated into *swim*.
These default TR rules can be computed by superposing the lhs of two TR rules and one lhs occurs completely in the other lhs. In that case the more special TR rule has to be checked for application first and the more general one last.

***Definition*: *Default relation* $>_{def}$**
If λ_1 is superposable with λ_2 with substitution σ **and**
λ_1 and λ_2 are not identical **and**
all variables X of $(X \leftarrow t) \in \sigma$ occur in λ_2
then $n_1 >_{def} n_2$, where $>_{def} \subseteq R_D \times R_D$.

The relation $>_{def}$ is reflexive, anti-symmetric and transitive, i.e. a (partial) order relation in R_D (the set of numbers of default rules) and $n_1 >_{def} n_2$ means that the lhs of rule n_1 is more special than the lhs of rule n_2 and n_1 has to be checked for application before n_2, even if they are part of a cycle. If λ_1 and λ_2 are identical without the names for variables, the TR rules are alternatives (see next subsection).
The set R_D contains subsets $C_i \subseteq R_D$, which are called chains, because for every x,y $\in C_i$ either x $>_{def}$ y or y $>_{def}$ x. Every chain has an infimum (the most special TR rule) and a supremum (the most gemeral TR rule).

5.3 Alternative TR rules

In order to get alternative solutions the derivation sequence has to branch at certain points which can also be computed by superposing the lhs of two TR rules. This is just the situation, in which the Knuth-Bendix algorithm computes a critical pair.

***Definition*: *Alternative relation* $\vee_{alt}$**
If λ_1 is superposable with λ_2 at subterm λ_1/u with substitution σ **and**
ρ_1 is not superposable with λ_2 **and**
if $\lambda_1/u = \lambda_1$ **then** λ_1 is not unifiable with ρ_2
then $\{\langle n'_1, \lambda_1\sigma \rightarrow \rho_1\sigma \rangle, \langle n'_2, \lambda_1\sigma \rightarrow \lambda_1[u \leftarrow \rho_2\sigma] \rangle\} \subseteq$ TRS **and** $n'_1 \vee_{alt} n'_2$, where $\vee_{alt} \subseteq R_A \times R_A$.

The relation $\vee_{alt}$ is reflexive, symmetric and transitive, i.e. an equivalence relation in R_A (the set of numbers of alternative rules). $n'_1 \vee_{alt} n'_2$ means that everytime TR rule n'_1 is applicable to an input term, then n'_2 is applicable and vice versa and the derivation sequence branches at this point. The additional condition that the rhs ρ_1 of one rule is not superposable with the lhs λ_2 of the other is necessary to exclude branches caused by rules in which the subterm λ_1/u is used as structural condition. The other condition is needed for the same reason in the special case when λ_1 and λ_2 are unifiable.
The rules n'_1 or n'_2 may already exist in the TRS. In that case either the lhs of n_1 and n_2 have been identical and $n'_1 = n_1$ and $n'_2 = n_2$ or $n_1 >_{def} n_2$ and $n'_1 = n_1$ or $n_2 >_{def} n_1$ and $n'_2 = n_2$.
If the rules n'_1 or n'_2 are not in the TRS, they are added and all other relations are computed. In most cases the new lhs is more special than the two other lhs and the corresponding default relations hold.
The lhs of the new TR rules n'_1 and n'_2 is the "superposition" of lhs of the rules n_1 and n_2. The rhs of n'_1 is the rhs of n_1, in which the variables are replaced according to the substitution σ. The rhs of n'_2 is the lhs of n_1, in which the subterm λ_1/u is replaced by the rhs of n_2, the variables of which are replaced according to

the superposition. The efficiency of the rewrite process crucially depends on the number and size of the equivalence classes of R_A.

5.4 The TRS interpreter

In order to apply the TR rules in an efficient order, the ordered set APP is precomputed:

$$\begin{aligned} APP = \; & R_N - R_C - R_D - R_A \\ & \cup \{n \mid [n] \subseteq R_C\} \\ & \cup \{n \mid n \text{ is infimum of a chain } C \subseteq R_D\} \\ & \cup \{n \mid [n] \subseteq R_A\} \end{aligned}$$

The set APP is ordered in the way that the sequence does not contradict to $>_{app}$. The interpreter for TRSes takes an input term t and checks all TR rules $\langle n, \lambda \rightarrow \rho \rangle$ in the order of APP.

If $n \in R_C$, then all TR rules $m \in [n] \subseteq R_C$ have to be checked for application in an arbitrary order as long as one rule of the cycle [n] is applicable, otherwise if t is superposable with λ, then the corresponding TR rule is applied else the next TR rule is checked for application.

If the applicable TR rule n is a default rule ($n \in R_D$), then the TR rules m ($n \geq_{def} m$) of the chain $C \subseteq R_D$ are applied in the order of $>_{def}$.

If the applicable TR rule n is an alternative rule ($n \in R_A$), then all the TR rules $m \in [n] \subseteq R_A$ are applied alternatively. Every alternative is a new branch in the derivation sequence.

If a TRS has a normal form, the algorithm interpretes it as efficiently as normalized TRSes.

The interpreter strategy of processing the input term, which represents a DAG, is flexible, i.e. the TR rule writer determines whether to proceed top-down, bottom-up, from left to right or vice versa. For example if the daughter categories of the lhs of the TR rules are target language (TL) categories and the mother category is a source language (SL) category which is to be translated into a TL category by the corresponding rule, then the interpreter will process the input structure bottom-up (see example of the TR rule in section 4).

5.5 Termination

For the use in our TRSes a quantitative ordering has been defined that guarantees the termination of all the TRSes used in our MT system. This ordering uses the different vocabularies of the SL and TL terms and requires that the number of SL categories occurring in the input term has to be reduced. This means that either a SL category has to be deleted or it has to be replaced by one or more TL categories. The minimal terms are all terms in which no SL categories occur. This ordering guarantees the termination of that sequence because the number of SL categories occurring in the input term is finite and after a finite number of applications there will be no SL categories left in the resulting term or no TR rule is applicable.

In order to prove the termination of a TRS, every single TR rule has to be checked. The lhs and rhs may contain occurrences of variables for terms, i.e. for (sub)DAGs, sets of (sub)DAGs, categories or feature values of categories. If for example a variable for a term representing a DAG is occurring once on the lhs and doubled on the rhs, then the number of SL categories occurring in the input term may be increased in the resulting term. For this reason an additional condition has to be defined which has to be fulfilled by the lhs and rhs of each TR rule. Every variable occurring on the rhs has to occur on the lhs. The two restrictions on TR rules allow for checking each rule for termination after it has been defined or modified so that the termination of the TRS can be guaranteed, because in every derivation step the number of SL categories is reduced. The derivation ends successfully if no SL category occurs in the resulting term and the corresponding tree can be generated by the context-free TL grammar.

The given termination condition fulfils the three constraints for termination (the input representation must be built in a well-behaved compositional way, recursive input representations have to be considered and the input should not be extended) discussed in [van Noord 90].

If the termination condition should prove to be inadequate for the use in MT, qualitative orderings or combinations of quantitative and qualitative orderings may be defined. Up to now the given quantitative ordering has proved to be adequate for the TRSes of our MT system.

6 Conclusion

The TRS interpreter and an editor for TRSes are implemented in Arity Prolog on an AT compatible PC. The editor allows for a graphical input in DAG notation, performs consistency checks on TR rules, checks the termination condition on TR rules, generates the corresponding terms and computes the application order by the means of the definitions given in section 5. TRSes have successfully been used for semantic analysis, transfer, generation and for the mapping from sentence semantic representations to conceptual representations in the experimental MT system of our project. TRSes are to be used to implement parsing in the near future. First experiments in that direction have been encouraging.

In the future the possibility of merging all TRSes defined for the translation of one language into another to form one single TRS is to be investigated. The advantage would be that the analysis depth will become flexible in the way that if the translation of a fragment of the source language syntactic representation is unambiguous, it can be directly translated into the target language syntactic representation, without the detour via the semantic representation. This would be possible for all levels of representation.

Another point will be to check the possibility of extracting reversible parts from one TRS and to use them for the other translation direction.

The interpreter, which uses the basic unification algorithm of [Eisele/ Dörre 86] to superpose the input term with the the lhs of the TR rules, is intended to be expanded with disjunction according to [Dörre/ Eisele 90].

With these additional features, term-rewriting is a powerful, elegant, complete and coherent device to describe the relations between all levels of representation in machine translation systems.

7 References

[Busemann 90]: S. Busemann: "Generierung natürlicher Sprache mit Generalisierten Phrasenstruktur-Grammatiken", KIT-Report 87, Technical University of Berlin 1990

[Arnold et al. 86]: D.J. Arnold, S. Krauwer, M. Rosner, L. de Tombe, G.B. Varile: "The ⟨C,A⟩,T Framework in EUROTRA: A Theoretically Committed Notation for MT", in: Procs. 11th COLING-86, Bonn 1986, pp. 297-303

[Busemann/Hauenschild 88]: S. Busemann, Ch. Hauenschild: "A Constructive View of GPSG or How to Make it Work", in: Procs. 12th COLING-88, Budapest 1988, pp. 77-82

[Busemann/Hauenschild 89]: S. Busemann, Ch. Hauenschild: "From FAS Representations to GPSG Structures", in: S. Busemann, Ch. Hauenschild and C. Umbach (eds.): "Views of the Syntax/Semantics Interface", Procs. of the Workshop "GPSG and Semantics", KIT-Report 74, Technical University of Berlin 1989, pp. 17 - 43

[Dershowitz 82]: N. Dershowitz: "Orderings for Term-Rewriting Systems", Theoretical Computer Science 17 (1982), North-Holland, pp. 279 - 301

[Dershowitz 85]: N. Dershowitz: "Termination", in: G.Goos, J. Hartmanis (eds.): "Rewriting Techniques and Applications", LNCS 202, Dijon, France 1985, pp. 180 - 224

[Dörre/Eisele 90]: J. Dörre, A. Eisele: "Feature Logic with Disjunctive Unification", in: Procs. 13th COLING-90 (Vol. 2), Helsinki 1990, pp. 100-105

[Eisele/Dörre 86]: A. Eisele, J. Dörre: "A Lexical Functional Grammar System in Prolog", in: Procs. 11th COLING-86, Bonn 1986, pp. 551-553

[Emele/Zajac 89]: M. Emele, R. Zajac, "RETIF: A Rewriting System for Typed Feature Structures", ATR Technical Report TR-I-0071 1989

[Gazdar et al. 85]: G. Gazdar, E. Klein, G. Pullum and I. Sag: "Generalized Phrase Structure Grammar", Oxford, Blackwell 1985

[Hauenschild/Busemann 88]: Ch. Hauenschild, S. Busemann: "A constructive version of GPSG for machine translation", in: E. Steiner, P. Schmidt and C. Zellinsky-Wibbelt (eds.): "From Syntax to Semantics - Insights From Machine Translation", London, Frances Pinter 1988, pp. 216-238

[Hauenschild/Umbach 88]: Ch. Hauenschild, C. Umbach: "Funktor-Argument-Struktur, Die satzsemantische Repräsentations- und Transferebene im Projekt KIT-FAST", in: J. Schütz (ed.): "Workshop Semantik und Transfer", EUROTRA-D Working Papers No.6, Saarbrücken 1988, pp. 16-35

[Huet/Oppen 80]: G. Huet, D. Oppen: "Equations and Rewrite Rules", in: R.V. Book (ed.): "Formal Language Theory, Perspectives and Open Problems", Academic Press 1980, pp. 349-405

[Knuth/Bendix 70]: D. Knuth, P. Bendix: "Simple Word Problems in Universal Algebras", in: J. Leech (ed.): "Computational Problems in Abstract Algebra", Pergamon Press 1970, pp. 263-297

[Peltason et al. 75]: C. Peltason, A. Schmiedel, C. Kindermann, J. Quantz, "The BACK System Revisited", KIT-Report 75, Technical University of Berlin 1989

[Russell et al. 91]: G. Russell, A. Ballim, D. Estival, S. Warwick-Amstrong, "A Language for the Statement of Binary Relations over Feature Structures", in: Procs. of the 5th Conference of the European Chapter of the ACL, Berlin 1991, pp. 287-292

[Schmitz et al. 91]: B. Schmitz, S. Preuß, C. Hauenschild: "Textrepräsentation und Hintergrundwissen für die Anaphernresolution im Maschinellen Übersetzungssystem KIT-FAST", KIT-Report 93, Technical University of Berlin 1991

[Sharp 88]: R. Sharp, "CAT2 - Implementing a Formalism for Multi-Lingual MT", in: Procs. of the 2nd International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages, CMU Pittsburg 1988

[van Noord 90]: G. van Noord: "Reversible Unification Based Machine Translation", in: Procs. 13th COLING-90 (Vol. 2), Helsinki 1990, pp. 299-304

[Weisweber 89]: W. Weisweber: "Transfer in Machine Translation by Non-Confluent Term-Rewrite Systems", in: Procs. of the 13th German Workshop on Artificial Intelligence (GWAI-89), Eringerfeld, September 1989, pp. 264 - 269

[Weisweber/Hauenschild 90]: W. Weisweber, Ch. Hauenschild: "A model of Multi-Level Transfer for Machine Translation and Its Partial Realization", KIT-Report 77, Technical University of Berlin 1990 and to appear in: Procs. of the Seminar "Computers & Translation '89", Tbilisi 1989

[Zajac 89]: R. Zajac, "A Transfer Model Using a Typed Feature Structure Rewriting System with Inheritance", in: Procs. of the 27th Annual Meeting of the ACL, Vancouver 1989

[Zajac 90]: R. Zajac, "A relational approach to translation", in: Procs. of the 3nd International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages, Austin 1990

[Zajac 91]: R. Zajac, "A Uniform Architecture for Parsing, Generation and Transfer", in: T. Strzalkowski (ed.), Procs. of the Workshop on Reversible Grammar in Natural Language Processing, Berkeley 1991, pp. 71-80