Converting CCGs into Typed Feature Structure Grammars[*]

Hans-Ulrich Krieger and Bernd Kiefer
German Research Center for Artificial Intelligence (DFKI GmbH)
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
{krieger,kiefer}@dfki.de

Introduction

In this paper, we report on a transformation scheme that turns a Categorial Grammar (CG), more specifically, a Combinatory Categorial Grammar (CCG; see (Baldridge 2002)) into a derivation- and meaning-preserving typed feature structure (TFS) grammar. We describe the main idea which can be traced back at least to work by (Karttunen 1986), (Uszkoreit 1986), (Bouma 1988), and (Calder, Klein, & Zeevat 1988). We then show how a typed representation of complex categories can be extended by other constraints, such as modes, and indicate how the Lambda semantics of combinators is mapped into a TFS representation, using unification to perform α-conversion and β-reduction (Barendregt 1984). We also present first findings concerning runtime measurements, showing that the PET system, originally developed for the HPSG grammar framework, outperforms the OpenCCG parser by a factor of more than 10.

Motivation

The Talking Robots (TR) group here at the LT Lab of DFKI uses categorial grammars in several large EU projects in order to communicate with robots in spoken language. The grammars for English and Italian are written in the OpenCCG dialect of CCG.

Faster Parser. The main rationale for our transformation method is driven by the need that we are looking for a reliable and trainable (C)CG parser that is faster than the one which comes with the OpenCCG system. People from the DFKI LT group have co-developed the PET system (Callmeier 2000), a highly-tuned TFS parser written in C++, which originally grew out of the HPSG community. In order to use such a TFS parser in a CG setting, the (combinatory) rules and lexicon entries need to be transformed into a TFS representation.

Structured Language Model. Another major rationale for the transformation comes from the fact that the CCG grammars are used for spoken language, operating on the output of a speech recognizer. Although speech recognizers are based on trained statistical models, modern recognizers can be further tuned by supplying an additional structured language model. Given a TFS grammar for the transformed CCG grammar, we would like to use the corpus-driven approximation method described in (Krieger 2007) to generate a context-free approximation of the deep grammar. This approximation then serves as our language model for the recognizer. Again, as is the case for PET, software can be reused here, since the method described in (Krieger 2007) is implemented for the external chart representation of the PET system.

Cross-Fertilization. We finally hope that our experiment provides insights on how to incorporate descriptive means from CG (e.g., direct slash notation for categories) into the HPSG framework, even though they are compiled out in the end. Thus, specification languages for HPSG, such as $\mathcal{TDL}$ (Krieger 1995), might be extended by some kind of macro formalism, allowing a grammar writer to state such extended rules. However, we will not speculate on this in the paper.

Categorial Grammar

Categorial grammar started with Bar-Hillel in 1953 who adapts and extend Ajdukiewicz's work by adding directionality to what Ajdukiewicz (by referring to Husserl) called "Bedeutungskategorie". The grammatical objects in Bar-Hillel's system are called *categories*. The set of *complex* categories C can be defined inductively by assuming a set of *atomic* categories A (e.g., s or np) and a set of binary functor symbols F_2 (usually $/$ and $\backslash$ for one-dimensional binary grammar rules):

1. *if* $\mathsf{a} \in \mathsf{A}$ *then* $\mathsf{a} \in \mathsf{C}$
2. *if* $\mathsf{c}, \mathsf{c}' \in \mathsf{C}$ *and* $\mathsf{f} \in \mathsf{F}_2$ *then* $\mathsf{cfc}' \in \mathsf{C}$

The system of categories in its simplest form is usually equipped with two very fundamental binary rules (or better, rule schemes), viz., forward ($>$) and backward ($<$) *functional application*—this is called the AB calculus (for Ajdukiewicz & Bar-Hillel). Here and in the following, we use the notation from (Baldridge 2002), originating from the work of Mark Steedman:

$$(>\mathbf{A}) \quad \mathsf{X/Y} \;\; \mathsf{Y} \;\Rightarrow\; \mathsf{X}$$

[*]The research described here has been partly financed by the **TAKE** project (take.dfki.de), funded by the German Federal Ministry of Education and Research, and the European Integrated projects **CogX** (cogx.eu), **NIFTi** (nifti.eu), and **Aliz-e** (aliz-e.org) under contract numbers 01IW08003, FP7 ICT 215181, 247870, and 248116. We would like to thank our reviewers for their useful comments.

$(<\mathbf{A})\quad \mathsf{Y}\ \ \mathsf{X}\backslash\mathsf{Y}\ \Rightarrow\ \mathsf{X}$

Depending on the kind of slash, complex category symbols in these rules look to the right (forward) or to the left (backward) in order to derive a simpler category. Such a framework is in the truest sense *lexicalized*, since the categories in these rules are actually category schemes: there is no category $\mathsf{X/Y}$, only instantiations, such as, for instance, $(\mathsf{s}\backslash\mathsf{np})/(\mathsf{s}\backslash\mathsf{np})$ for modal verbs. Furthermore, and very importantly, concrete categories are only specified for lexicon entries ($\vdash$ maps the word to its category):

defeat $\vdash (\mathsf{s}\backslash\mathsf{np})/\mathsf{np}$

Not only are lexical entries equipped with a category, but also with a semantics. Since Montague, categorial grammarians have often used the Lambda calculus to make this explicit. Abstracting away from several important things such as tense, we can define what is meant by the transitive verb *defeat* (: is used to attach the semantic to a lexicon entry):

defeat $\vdash (\mathsf{s}\backslash\mathsf{np})/\mathsf{np} : \lambda x.\lambda y.\mathbf{defeat}(y, x)$

The above two rules for functional application in fact indicate how the semantics is supposed to be assembled, viz., by *functional application*:

$(>\mathbf{A})\quad \mathsf{X/Y}:f\ \ \mathsf{Y}:a\ \Rightarrow\ \mathsf{X}:fa$
$(<\mathbf{A})\quad \mathsf{Y}:a\ \ \mathsf{X}\backslash\mathsf{Y}:f\ \Rightarrow\ \mathsf{X}:fa$

f in the above two rules actually abbreviates $\lambda x.fx$, so that the resulting phrase on the right-hand side is in fact fa as a result of applying β-reduction to $(\lambda x.fx)(a)$.

Given these rule schemes, we can easily find a derivation for sentences, such as *Brazil defeats Germany*:

$$\dfrac{\mathsf{np}{:}\mathbf{Brazil}\quad \dfrac{(\mathsf{s}\backslash\mathsf{np})/\mathsf{np}{:}\lambda x.\lambda y.\mathbf{defeat}(y,x)\quad \mathsf{np}{:}\mathbf{Germany}}{\mathsf{s}\backslash\mathsf{np}{:}\lambda y.\mathbf{defeat}(y,\mathbf{Germany})}}{\mathsf{s}{:}\mathbf{defeat}(\mathbf{Brazil},\mathbf{Germany})}$$

A lot of linguistic phenomena can be perfectly handled by the two application rules. However, many researchers have argued that the AB calculus should be extended by rules that have a greater combinatory potential. CCG, for instance, employs rules for forward/backward (harmonic & crossed) composition, substitution, and type raising (we only list the forward versions):

Harmonic Composition $(>\mathbf{B})\quad \mathsf{X/Y}\ \ \mathsf{Y/Z}\ \Rightarrow\ \mathsf{X/Z}$

Crossed Composition $(>\mathbf{B}_\times)\quad \mathsf{X/Y}\ \ \mathsf{Y}\backslash\mathsf{Z}\ \Rightarrow\ \mathsf{X}\backslash\mathsf{Z}$

Substitution $(>\mathbf{S})\quad (\mathsf{X/Y})/\mathsf{Z}\ \ \mathsf{Y/Z}\ \Rightarrow\ \mathsf{X/Z}$

Type Raising $(>\mathbf{T})\quad \mathsf{X}\ \Rightarrow\ \mathsf{Y}/(\mathsf{Y}\backslash\mathsf{X})$

Related to these rules are the three combinators (e.g., higher-order functions) for *composition* **B**, *subsitution* **S**, and *type raising* **T** (see (Steedman 2000)):

- $\mathbf{B}fg \equiv \lambda x.f(gx)$
- $\mathbf{S}fg \equiv \lambda x.fx(gx)$
- $\mathbf{T}x \equiv \lambda f.fx$

In a certain sense, even functional application can be seen as a combinator, since argument a can be regarded as a nullary function:

- $\mathbf{A}fa \equiv \lambda x.fx(a)$

The three combinators above indicate how semantics should be assembled within the categorial rules. Semantics construction is addressed later when we move to the TFS representation of the CCG rules.

Idea

The TFS encoding below distinguishes between atomic and complex categories. Atomic categories such as s do not have an internal structure. However, atomic categories in CCG are usually part of a structured inheritance lexicon, quite similar to HPSG. Atomic categories here do have a flat internal structure, encoding morpho-syntactical feature-value combinations. Thus, atomic categories in our transformation will be realized as typed feature structures to fully exploit the potential of typed unification.

In contrast, the most general functor category type has two subtypes / (*slash*) and \ (*backslash*) and defines three appropriate features: 1ST (FIRST), 2ND (SECOND), and MODE (for modalities, explained later). This encoding is similar to the CUG encoding in (Karttunen 1986; Uszkoreit 1986); however, the DIR (direction) feature is realized as a type, and the ARG (argument) and VAL (value) features through features 1ST and 2ND. Our encoding is advantageous in that it (i) makes a complex functor hierarchy possible, even multi-dimensional functors; (ii) allows for functors of more than two arguments, thus going beyond the potential of binary rules; and (iii) need not look at the directionality of the functor in order to specify the proper values for ARG and VAL (as is the case in Lambek's notation).

Underspecified atomic categories in the CCG rules above are realized through logic variables (coreferences) in the TFS rules below. Moreover, a distinguished list-valued feature DTRS (daughters) is employed in the TFS representation to model the LHS arguments of CCG rules.

Examples

We start with the TFS encoding of a proper noun, a transitive verb, and a modal verb, followed by the basic representation of the forward versions of the CCG rules, including a form of Lambda semantics in order to indicate how the compositional semantic approach of categorial grammars translates into a TFS grammar.

Lexicon Entries

A proper noun, such as *Germany* $\vdash$ np : **Germany** is mapped to a flat feature structure with distinguished attributes CAT and SEM:

$$\begin{bmatrix} \textit{germany} \\ \textsc{cat}\ \textit{np} \\ \textsc{sem}\ \textbf{Germany} \end{bmatrix}$$

Actually, **Germany** is represented as a nullary function (i.e., a function with zero arguments), but this does not matter here. The value of SEM is either a function specification (type f) with NAME and ARGS features, or the representation of a Lambda term (type λ), encoded through VAR and BODY. The body of a Lambda term might again be a Lambda term or a function specification. Functional composition is encoded through an embedding of function specifications.

The representation of transitive verbs is a straightforward translation of the one-dimensional CCG specification *defeat* $\vdash \mathsf{(s\backslash np)/np} : \lambda x.\lambda y.\textbf{defeat}(y,x)$. Note that the de-curried representation suggests that β-reduction for x happens *before* y. Note further that even though x is bound first, it is the second argument of **defeat**.

$$\begin{bmatrix} \textit{defeat} & \\ \text{CAT} & \begin{bmatrix} / & \\ \text{1ST} & \begin{bmatrix} \backslash & \\ \text{1ST} & s \\ \text{2ND} & np \end{bmatrix} \\ \text{2ND} & np \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \lambda & \\ \text{VAR} & \boxed{x} \\ \text{BODY} & \begin{bmatrix} \lambda & \\ \text{VAR} & \boxed{y} \\ \text{BODY} & \begin{bmatrix} f & \\ \text{NAME} & \textbf{defeat} \\ \text{ARGS} & \left\langle \boxed{y}, \boxed{x} \right\rangle \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The representation of modal verbs is more complicated because P in the complex Lambda term below is not an argument like x (or x and y above), but instead a function that is *applied* to x—it might even be a Lambda term as the example *Brazil should defeat Germany* shows. Here is the categorial representation, followed by the TFS encoding: *should* $\vdash \mathsf{(s\backslash np)/(s\backslash np)} : \lambda P.\lambda x.\textbf{should}(Px)$

$$\begin{bmatrix} \textit{should} & \\ \text{CAT} & \begin{bmatrix} / & \\ \text{1ST} & \begin{bmatrix} \backslash & \\ \text{1ST} & s \\ \text{2ND} & np \end{bmatrix} \\ \text{2ND} & \begin{bmatrix} \backslash & \\ \text{1ST} & s \\ \text{2ND} & np \end{bmatrix} \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \lambda & \\ \text{VAR} & \begin{bmatrix} \lambda & \\ \text{VAR} & \boxed{x} \\ \text{BODY} & \boxed{b} \end{bmatrix} \\ \text{BODY} & \begin{bmatrix} \lambda & \\ \text{VAR} & \boxed{x} \\ \text{BODY} & \begin{bmatrix} f & \\ \text{NAME} & \textbf{should} \\ \text{ARGS} & \left\langle \boxed{b}\ ] \right\rangle \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Rules

Next comes the rule for **Forward Functional Application**:

$(>\mathbf{A})\quad \mathsf{X/Y} : f \quad \mathsf{Y} : a \;\Rightarrow\; \mathsf{X} : fa$

$$\begin{bmatrix} >\mathbf{A} & \\ \text{CAT } \boxed{X} & \\ \text{SEM } \boxed{f} & \\ \text{DTRS} & \left\langle \begin{bmatrix} \text{CAT} & \begin{bmatrix} / & \\ \text{1ST} & \boxed{X} \\ \text{2ND} & \boxed{Y} \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \lambda & \\ \text{VAR} & \boxed{a} \\ \text{BODY} & \boxed{f} \end{bmatrix} \end{bmatrix}, \begin{bmatrix} \text{CAT} & \boxed{Y} \\ \text{SEM} & \boxed{a} \end{bmatrix} \right\rangle \end{bmatrix}$$

Given this rule and the entries for *should*, *defeat*, and *Germany*, the twofold application of $(>\mathbf{A})$ yields the correct semantics for the VP *should defeat Germany*, viz., $\lambda x.\textbf{should}(\textbf{defeat}(x, \textbf{Germany}))$, or as a TFS, constructed via unification:

$$\begin{bmatrix} \lambda & \\ \text{VAR} & \boxed{x} \\ \text{BODY} & \begin{bmatrix} f & \\ \text{NAME} & \textbf{should} \\ \text{ARGS} & \left\langle \begin{bmatrix} f & \\ \text{NAME} & \textbf{defeat} \\ \text{ARGS} & \langle \boxed{x}, \textbf{Germany} \rangle \end{bmatrix} \right\rangle \end{bmatrix} \end{bmatrix}$$

The TFS representation of the three rules to follow next are **Forward Harmonic Composition**, **Forward Substitution**, and **Forward Type Raising**. The motivation for such kind of rules, can, e.g., be found in (Baldridge 2002).

$(>\mathbf{B})\quad \mathsf{X/Y} : f \quad \mathsf{Y/Z} : g \;\Rightarrow\; \mathsf{X/Z} : \lambda x.f(gx)$

$$\begin{bmatrix} >\mathbf{B} & \\ \text{CAT} & \begin{bmatrix} / & \\ \text{1ST} & \boxed{X} \\ \text{2ND} & \boxed{Z} \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \lambda & \\ \text{VAR} & \boxed{x} \\ \text{BODY} & \boxed{f} \begin{bmatrix} \text{ARGS}|\text{FIRST } \boxed{g} \end{bmatrix} \end{bmatrix} \\ \text{DTRS} & \left\langle \begin{bmatrix} \text{CAT} & \begin{bmatrix} / & \\ \text{1ST} & \boxed{X} \\ \text{2ND} & \boxed{Y} \end{bmatrix} \\ \text{SEM}|\text{BODY} & \boxed{f} \end{bmatrix}, \begin{bmatrix} \text{CAT} & \begin{bmatrix} / & \\ \text{1ST} & \boxed{Y} \\ \text{2ND} & \boxed{Z} \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \text{VAR} & \boxed{x} \\ \text{BODY} & \boxed{g} \end{bmatrix} \end{bmatrix} \right\rangle \end{bmatrix}$$

$(>\mathbf{S})\quad \mathsf{(X/Y)/Z} : f \quad \mathsf{Y/Z} : g \;\Rightarrow\; \mathsf{X/Z} : \lambda x.fx(gx)$

$$\begin{bmatrix} >\mathbf{S} & \\ \text{CAT} & \begin{bmatrix} / & \\ \text{1ST} & \boxed{X} \\ \text{2ND} & \boxed{Z} \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \lambda & \\ \text{VAR} & \boxed{x} \\ \text{BODY} & \boxed{f} \begin{bmatrix} \text{ARGS}|\text{REST}|\text{FIRST } \boxed{g} \end{bmatrix} \end{bmatrix} \\ \text{DTRS} & \left\langle \begin{bmatrix} \text{CAT} & \begin{bmatrix} / & \\ \text{1ST} & \begin{bmatrix} / & \\ \text{1ST} & \boxed{X} \\ \text{2ND} & \boxed{Y} \end{bmatrix} \\ \text{2ND} & \boxed{Z} \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \lambda & \\ \text{VAR} & \boxed{x} \\ \text{BODY} & \boxed{f} \end{bmatrix} \end{bmatrix}, \begin{bmatrix} \text{CAT} & \begin{bmatrix} / & \\ \text{1ST} & \boxed{Y} \\ \text{2ND} & \boxed{Z} \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \lambda & \\ \text{VAR} & \boxed{x} \\ \text{BODY} & \boxed{g} \end{bmatrix} \end{bmatrix} \right\rangle \end{bmatrix}$$

$(>\mathbf{T})\quad \mathsf{X} : x \;\Rightarrow\; \mathsf{Y/(Y\backslash X)} : \lambda f.fx$

$$\begin{bmatrix} >\mathbf{T} & \\ \text{CAT} & \begin{bmatrix} / & \\ \text{1ST} & \boxed{\text{Y}} \\ \text{2ND} & \begin{bmatrix} \backslash & \\ \text{1ST} & \boxed{\text{Y}} \\ \text{2ND} & \boxed{\text{X}} \end{bmatrix} \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \lambda & \\ \text{VAR} & \boxed{\text{f}} \\ \text{BODY} & \begin{bmatrix} f & \\ \text{NAME} & \boxed{\text{f}} \\ \text{ARGS} & \langle \boxed{\text{x}} \rangle \end{bmatrix} \end{bmatrix} \\ \text{DTRS} & \left\langle \begin{bmatrix} \text{CAT} & \boxed{\text{X}} \\ \text{SEM} & \boxed{\text{x}} \end{bmatrix} \right\rangle \end{bmatrix}$$

Extensions

In this section, we outline several extensions of the basic CG system and show what their TFSs representation look like.

$-Convention & Generalized Forward Composition

The VP *should defeat Germany* from the rule section can not only be analyzed by a twofold application of $(>\mathbf{A})$, but also by applying $(>\mathbf{B})$ to *should* and *defeat*, followed by $(>\mathbf{A})$. Now, $(>\mathbf{B})$ must be generalized in case we are even interested in ditransitive verbs, or even VPs with further PP attachments. Instead of describing every possible alternative, (Steedman 2000) devised a compact notation using $-schemes to characterize functions of varying numbers of arguments, or as (Baldridge 2002) puts it: "In essence, the $ acts as a stack of arguments that allows the rule to eat into a category". For example, the schema $\mathsf{s}/\$$ is a representative for the infinite set $\{\mathsf{s}, \mathsf{s}/\mathsf{np}, (\mathsf{s}/\mathsf{np})/\mathsf{np}, \ldots\}$.

Formally, the expansion of a $-category can be inductively defined as follows. Let C be the set of complex categories, as defined earlier, F_2 the set of binary functor symbols, and let $c \in \mathsf{C}$ and $\mathsf{f} \in \mathsf{F}_2$. Define $\mathsf{C}_\epsilon := C \cup \{\epsilon\}$, $\mathsf{cf}\epsilon := \mathsf{c}$, and $\mathsf{cfC}_\epsilon := \{\mathsf{cfd} \mid \mathsf{d} \in \mathsf{C}_\epsilon\}$. Then $\mathsf{cf}\$:= (\mathsf{cfC}_\epsilon)\mathsf{fC}_\epsilon$.

Let us move on to the rule for generalized forward composition $(>\mathbf{B}^n)$ which employs $ and its TFS counterpart:

$(>\mathbf{B}^n)$ $\mathsf{X/Y}$ $\mathsf{(Y/Z)/\$}$ $\Rightarrow$ $\mathsf{(X/Z)/\$}$

$$\begin{bmatrix} >\mathbf{B}^{n>1} & \\ \text{CAT} & \begin{bmatrix} / & \\ \text{1ST}^{n-1} & \begin{bmatrix} / & \\ \text{1ST} & \boxed{\text{X}} \\ \text{2ND} & \boxed{\text{Z}} \end{bmatrix} \\ \text{2ND} & \boxed{\$} \end{bmatrix} \\ \text{DTRS} & \left\langle \begin{bmatrix} \text{CAT} & \begin{bmatrix} / & \\ \text{1ST} & \boxed{\text{X}} \\ \text{2ND} & \boxed{\text{Y}} \end{bmatrix} \end{bmatrix}, \begin{bmatrix} \text{CAT} & \begin{bmatrix} / & \\ \text{1ST}^{n-1} & \begin{bmatrix} / & \\ \text{1ST} & \boxed{\text{Y}} \\ \text{2ND} & \boxed{\text{Z}} \end{bmatrix} \\ \text{2ND} & \boxed{\$} \end{bmatrix} \end{bmatrix} \right\rangle \end{bmatrix}$$

The above TFS uses a "coordinated" path expression 1ST^{n-1} at two places inside the rule structure and is, in a certain sense, even worse than *functional uncertainty* (Kaplan & Maxwell III 1988), since it involves counting. To the best of our knowledge, we are not aware of TFS formalisms which offer such descriptive means. We thus understand the above structure as a schema that can be compiled into $k-1$ different concrete rules for $1 < n \leq k$. Another way to carry over the meaning would be to add a unary and a binary helper rule for each $-rule which together simulate the expansion of a $-category. We have opted for the first solution, since the latter could blow up the search space of the parser.

We finally note that $>\mathbf{B}^1$ is equivalent to the original rule $>\mathbf{B}$. In case we define $\text{1ST}^0 := \epsilon$ and assume that $\text{2ND} \doteq \boxed{\text{Z}} \wedge \text{2ND} \doteq \boxed{\$}$ leads to $\boxed{\text{Z}} = \boxed{\$}$ (features are functional relations!), there is no need to specify $>\mathbf{B}^1$ separately.

In principle, other rule schemata might be generalized in such a way, but at the expense of further uncertainty and overgeneration during parsing.

Atomic Categories & Morpho-Syntax

As indicated earlier, atomic categories in CCG usually do have a flat internal structure. For instance, the category s_i refers to an inflection phrase (Baldridge 2002). The TFS representation then uses s_i as a type, having the following definition:

$$\mathsf{IP} \equiv \begin{bmatrix} s_i & \\ \text{SPEC} & \textit{boolean} \\ \text{ANT} & \textit{boolean} \\ \text{CASE} & \textit{case} \\ \text{VFORM} & \textit{fin} \\ \text{MARKING} & \textit{unmarked} \end{bmatrix}$$

Words in CCG usually refer to these more specialized categories; for instance, the ECM verb *believe* $\vdash (\mathsf{s}_i\backslash\mathsf{np})/\mathsf{s}_{fin}$. Given such specific category information, TFS unification takes care that the additional constraints are "transported" throughout the derivation tree.

Modes & Modalized CCG

Besides having more control through specialized atomic categories as shown above, *multi-modal CCG* incorporates means from *Categorial Type Logic* to provide a further fine-grained lexical control through so-called *modalities*; see (Baldridge & Kruijff 2003) for a detailed description. For example, the complex category of the coordination particle *and* $\vdash (\mathsf{s}_i\backslash\mathsf{s}_i)/\mathsf{s}_i$ which can lead to unwanted analyses is replaced by the modalized category $(\mathsf{s}_i\backslash_\star\mathsf{s}_i)/_\star\mathsf{s}_i$.

In principle, modes can be "folded" into subtypes of the very general complex category types / and \. We have, however, opted for an additional feature MODE which takes values from the following atomic mode type hierarchy:

$$\begin{tikzcd} & \cdot \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\ \star & \diamond & \times \end{tikzcd}$$

There are further modalities which are not of interest to us here. Let us finally present the TFSs for *and* and the multi-modal CCG forward type raising rule rule $(>\mathbf{T})$ which even enforces modes to be identical between the embedded and the outer slash.

$$\begin{bmatrix} \textit{and} & \\ \text{CAT} & \begin{bmatrix} / & \\ \text{1ST} & \begin{bmatrix} \backslash & \\ \text{1ST} & s_i \\ \text{2ND} & s_i \\ \text{MODE} & \star \end{bmatrix} \\ \text{2ND} & s_i \\ \text{MODE} & \star \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} >\mathbf{T} & \\ \text{CAT} & \begin{bmatrix} / & \\ \text{MODE} & \boxed{M} \\ \text{1ST} & \boxed{Y} \\ \text{2ND} & \begin{bmatrix} \backslash & \\ \text{MODE} & \boxed{M} \\ \text{1ST} & \boxed{Y} \\ \text{2ND} & \boxed{X} \end{bmatrix} \end{bmatrix} \\ \text{DTRS} & \langle [\,\text{CAT}\ \boxed{X}\,] \rangle \end{bmatrix}$$

First Measurements

We have compared the performance of the CCG parser and the PET system on a MacBook Pro (2GHz Core Duo, 32 bit architecture). The measurements were carried out against a hand-crafted artificial test corpus of 5,000 sentences with an average length of 7 and a maximal length of 12 words, including sentences with heavy use of different kinds of coordination, such as *Brazil will meet and defeat Germany* or *Brazil should defeat Germany and Italy and England.*

We have switched off the semantics and have only compared the syntactic coverage, using categorial information, including modes. We have also switched off the type raising rules in both parsers, since the OpenCCG parser seems to ignore them in analyses licensed by the grammar theory. Packing in both parsers has been switched on, supertagging switched off (in fact, PET does not provide a supertagging stage).

We further note that we have obtained about twice as much analyses for PET (approximately 15,000 analyses) as the OpenCCG system, the reason for this currently unclear. For instance, the CCG parser produces only **one** analysis for the sentence *Brazil should defeat Germany*, even though a careful inspection of the rules shows that **two** analyses are possible (as is the case for PET), viz.,

$$[(<\mathbf{A})\ \textit{Brazil}\ [(>\mathbf{A})\ \textit{should}\ [(>\mathbf{A})\ \textit{defeat}\ \ \textit{Germany}\,]\,]\,]$$
$$[(<\mathbf{A})\ \textit{Brazil}\ [(>\mathbf{A})\ [(>\mathbf{B})\ \textit{should}\ \ \textit{defeat}\,]\ \textit{Germany}\,]\,]$$

Even though we double the number of analyses, PET is about one magnitude faster (overall 2.67 vs. 28.9 seconds for the full set of 5,000 sentences).

Both PET and the OpenCCG system have implemented standard CYK parsers. We believe that the difference in the running time is related to the choice of the programming language (C++ vs. Java), but also to maintenance effort and the still ongoing development of the PET system by an active community, whereas the evolution of the core parsing engine in the OpenCCG library seems to have ended several years ago.

To some extend, the above mismatch is related to the fact that certain "settings" in the CCG are realized through *program code*, but **not** *declaratively* stated in the lingware. For instance, the type raising rules can in principle be applied to arbitrary categories, but, by default, the OpenCCG code limits them to NPs only. Given our treatment, such a restriction can be easily stated in the TFSs for the type raising rules, and we think that this is the right place to do so:

$$\begin{bmatrix} >\mathbf{T} & \\ \text{CAT} & \begin{bmatrix} / & \\ \text{1ST} & \boxed{Y} \\ \text{2ND} & \begin{bmatrix} \backslash & \\ \text{1ST} & \boxed{Y} \\ \text{2ND} & \boxed{X} \end{bmatrix} \end{bmatrix} \\ \text{DTRS} & \langle [\,\text{CAT}\ \boxed{X}\,\textit{np}\,] \rangle \end{bmatrix}$$

Other "adjusting screws" in OpenCCG, e.g., the specification of the atomic mode hierarchy (see last subsection) are also "casted" in program code (deeply nested *if-then-else* statements), whereas our treatment uses a type hierarchy, helping to better understand and manipulate the parser's output. Given these remarks, explaining missing analyses in OpenCCG has required a deep inspection of the program code. Besides the MODE dimension, we found a further orthogonal binary ABILITY dimension with values *inert* and *active* that was hidden in the program code (Java classes) for each categorial rule. The PET version of CCG still overgenerates (to a lesser extent) and we hope to unveil the secrets at the conference.

Moving Further

The transformation schema described in this paper has been manually constructed for the rules, the lexical types, and a small set of lexicon entires. In order to automatically transform the OpenCCG grammars from our Lab for English and Italian, we have implemented code that operates on the XML output of the `ccg2xml` converter for CCG's WebCCG input format. This includes files for rules, general types, and so-called families which are collections of lexical types and corresponding lexical entries.

Contrary to traditional CG and CCG, OpenCCG does not use Lambda semantics, but instead comes with a kind of Davidsonean event semantics, comparable to MRS, building on Blackburn's hybrid modal logic: Hybrid Logic Dependency Semantics (HLDS) (Baldridge & Kruijff 2002). Looking more closely on the seemingly different notation, it becomes quite clear that HLDS formulae can be straightforwardly translated into a TFS representation. We can only throw a glance on a small example at the end of this paper.

Originally, the HLDS representations were built up in tandem with the construction of the categorial backbone (Baldridge & Kruijff 2002), comparable to the construction of Lambda semantics in our rules before. (White & Baldridge 2003) has improved on this construction by attaching the semantics, i.e., the elementary predications (EPs), directly to the atomic categories from which a complex category is built up (see (Zeevat 1988) for a similar treatment in UCG).

Consider the sentence *Marcel proved completeness* (Kruijff & Baldridge 2004). Subscripts attached to atomic categories (the nominals) can be used to access them. The satisfaction operator @ that is equipped with a subscript e indicates that the formulae to follow hold at a state named e:

$$\begin{aligned}&\textit{proved} \vdash (\mathsf{s}_e\backslash \mathsf{np}_x)/\mathsf{np}_y : \\ &\quad @_e\textbf{prove} \wedge @_e\langle \text{TENSE}\rangle\textbf{past} \wedge @_e\langle\text{ACT}\rangle x \wedge @_e\langle\text{PAT}\rangle y \\ &\textit{Marcel} \vdash \mathsf{np}_m : @_m\textbf{Marcel} \\ &\textit{completeness} \vdash \mathsf{np}_c : @_c\textbf{completeness}\end{aligned}$$

By *conjoining* the EPs during the application of ($>\mathbf{A}$) and ($<\mathbf{A}$), we immediately obtain

$$\begin{aligned}&\textit{Marcel proved completeness} \vdash \mathsf{s}_e : \\ &\quad @_e\textbf{prove} \wedge @_e\langle \text{TENSE}\rangle\textbf{past} \wedge @_e\langle\text{ACT}\rangle m \wedge \\ &\quad @_e\langle\text{PAT}\rangle c \wedge @_m\textbf{Marcel} \wedge @_c\textbf{completeness}\end{aligned}$$

Exactly these effects can be achieved through unification in our framework. The CCG nominals are realized through logic variables (coreference tags), atomic categories, such as s or np are assigned a further feature INDEX, cospecified with the semantics, and the nominals are realized through ordinary features. In theory, SEM is a set-valued feature whose elements are combined conjunctively (as in HLDS or MRS). Since $\mathcal{TDL}$ (and PET) does not provide sets, the usual list implementation is used. This gives us the following TFSs (we have omitted the explicit representation of the name of the event variables e, m, and c in the individual EPs below):

$$\begin{bmatrix} \textit{proved} & \\ \text{CAT} & \begin{bmatrix} / & \\ \text{1ST} & \begin{bmatrix} \backslash & \\ \text{1ST} & \begin{bmatrix} \textit{s} \\ \text{INDEX } \boxed{e}\end{bmatrix} \\ \text{2ND} & \begin{bmatrix} \textit{np} \\ \text{INDEX } \boxed{x}\end{bmatrix}\end{bmatrix} \\ \text{2ND} & \begin{bmatrix}\textit{np}\\ \text{INDEX } \boxed{y}\end{bmatrix}\end{bmatrix} \\ \text{SEM} & \boxed{e}\ \langle\, \textbf{prove}, [\text{TENSE } \textit{past}], [\text{ACT } \boxed{x}], [\text{PAT } \boxed{y}]\, \rangle \end{bmatrix}$$

$$\begin{bmatrix} \textit{marcel} & \\ \text{CAT} & \begin{bmatrix} \textit{np} \\ \text{INDEX } \boxed{m}\end{bmatrix} \\ \text{SEM} & \boxed{m}\ \langle\, \textbf{Marcel}\, \rangle\end{bmatrix} \quad \begin{bmatrix} \textit{completeness} & \\ \text{CAT} & \begin{bmatrix} \textit{np} \\ \text{INDEX } \boxed{c}\end{bmatrix} \\ \text{SEM} & \boxed{c}\ \langle\, \textbf{completeness}\, \rangle\end{bmatrix}$$

Alternatively, the list representation of EPs might be replaced by a single complex feature structure. However, the list implementation makes it easy to implement relational information, e.g., the representation of several modifiers. Given the above encoding, there is no longer a need to specify semantics construction in each of the categorial rule schemata: semantics construction simply "happens" here when categorial information is unified. In a certain sense, this is easier and more elegant than representing the effects of the different combinators **A**, **B**, **S**, **T** in the different kinds of rule schemata, as we have described in the beginning of this paper. More complex constructions involving, e.g., coordination particles, stipulate that the list under SEM is in fact a difference list in order to ease the implementation of a list append that is not required in the example above.

References

Baldridge, J., and Kruijff, G.-J. M. 2002. Coupling CCG and hybrid logic dependency semantics. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 319–326.

Baldridge, J., and Kruijff, G.-J. M. 2003. Multi-modal combinatory categorial grammar. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics*, 211–218.

Baldridge, J. 2002. *Lexically Specified Derivational Control in Combinatory Categorial Grammar*. Ph.D. Dissertation, University of Edinburgh, Division of Informatics, Institute for Communicating and Collaborative Systems.

Barendregt, H. 1984. *The Lambda Calculus, its Syntax and Semantics*. Amsterdam: North-Holland.

Bouma, G. 1988. Modifiers and specifiers in categorial unification grammar. *Linguistics* 26:21–46.

Calder, J.; Klein, E.; and Zeevat, H. 1988. Unification categorial grammar: A concise, extendable grammar for natural language processing. In *Proceedings of the 12th International Conference on Computational Linguistics*, 83–86.

Callmeier, U. 2000. PET—A Platform for Experimentation with Efficient HPSG Processing. *Natural Language Engineering* 6(1):99–107.

Kaplan, R. M., and Maxwell III, J. T. 1988. An algorithm for functional uncertainty. In *Proceedings of the 12th International Conference on Computational Linguistics*, 297–302.

Karttunen, L. 1986. Radical lexicalism. Technical Report CSLI-86-68, Center for the Study of Language and Information, Stanford University.

Krieger, H.-U. 1995. *$\mathcal{TDL}$—A Type Description Language for Constraint-Based Grammars. Foundations, Implementation, and Applications*. Ph.D. Dissertation, Universität des Saarlandes, Department of Computer Science.

Krieger, H.-U. 2007. From UBGs to CFGs—a practical corpus-driven approach. *Natural Language Engineering* 13(4):317–351. Published online in April 2006.

Kruijff, G.-J. M., and Baldridge, J. 2004. Generalizing dimensionality in combinatory categorial crammar. In *Proceedings of the 20th International Conference on Computational Linguistics*.

Steedman, M. 2000. *The Syntactic Process*. Cambridge, MA: MIT Press.

Uszkoreit, H. 1986. Categorial unification grammars. In *Proceedings of the 11th International Conference on Computational Linguistics*, 187–194.

White, M., and Baldridge, J. 2003. Adapting chart realization to CCG. In *Proceedings of the 9th European Workshop on Natural Language Generation*.

Zeevat, H. 1988. Combining categorial grammar and unification. In Reyle, U., and Rohrer, C., eds., *Natural Language Parsing and Linguistic Theories*. Reidel, Dordrecht. 202–229.