Argument Structure as an Interface Between Form and Meaning

MARCUS KRACHT

Abstract. In this paper we argue for a specific form of semantic representations, which pair together a DRS with a list of statements that declare how the variables occurring in the DRS are to be handled under merge. This eliminates the need for an external accounting device. Additionally, it greatly simplifies the amount of structure needed to account for the behaviour of contextually determined variables as well as argument identification in general. Factoring out structural from semantic information greatly simplifies the representations, and therefore it is expected to also simplifiy learning these structures.

1. Introduction

Learning from corpora has been a major domain in computational linguistics. Its use is twofold: the practical use is that it eliminates the need to design actual systems, say grammars, in favour of letting the machine create them by itself. The second, theoretical use is that it may show us how humans may learn languages. Largely, the contention has been that it is not feasible to learn structural properties of languages without prior, inborn knowledge of their basic structural traits. This theory has been supported by formal proofs that languages are virtually unlearnable (see [5] and references therein). The study of learning algorithms may be able to refute that claim if it can come up with an algorithm that learns language from scratch.

In this paper I shall suggest that the standard view of the matter is too simplistic in that it focuses exclusively on *syntactic* properties. If other kind of input was available, in particular semantic input, the situation might change significantly. At present, however, it is unclear what it is that should be learned and how it can be learned from the available data. To remedy this, I propose a particular theory of argument structure, which provides an interface between syntax and semantics. Argument structure associates basic morphological, syntactic and semantic properties with arguments and relates them to each other. In this way, the structures are built up simultaneously, using input from all three sources at once. This simplifies the task at many levels:

① The burden of structural description is not put on one level alone. Thus syntax is not responsible for interpretation, and semantics is not responsible for structure building.
② Each of the descriptions (morphological, syntactic, semantic) are relatively simple; the complications arise mainly from their interaction.
③ Learning these structures is simplified, since at each level the information needed to fix their behaviour is relatively small.

To see the advantage, just look at a full specification of lexical items in categorial grammar, or at the sentential structures in the Minimalist Program. In both instances the syntactic structure is everything: once it is built, the interpretation is a trivial matter. The syntactic structures are however very intricate and so are the individual lexical entries.

There is even a fundamental philosophy behind this, called *bootstrapping*. It says in this case that once the fundamental properties of syntactic structures are uncovered, semantics becomes trivial. It is, so to speak, a mere epiphenomenon of syntax. Even if that were true, there are several problems with that approach: for one, the structures that need to be learned are largely invisible (LF) so the language learner needs to know which of the hidden structures he has to associate with a given overt sentence. Second, the current structures are very complex, it is unclear how they can actually be learned. Third, the association between LFs and semantic representations is still not determined (assuming here that LF does not do any semantics). Fourth, in actual fact the many categories assumed in syntactic theory do have semantic labels (virtually all of them), and the association between these labels and semantic categories is not random. It is claimed that the positions are completely formal, but I am not sure that this is factually the case.

2. Facing the Theoretical Problem: Alignment Based Learning

Learning grammars from corpora is a delicate affair. The standard practice in Data Oriented Parsing, for example, has been to start with an already existing analysis, that is, a set of sentences already parsed according to some grammar of the language. The system is trained on that set and is expected to learn weights for the rules, and may also come up with refinements of the grammar. (See, for example, the collection [1].) But it cannot change the fundamental assumptions on which the corpus is based.

The problem with this approach is evident: the corpora are not the actual input for the human learner, but they have been manufactured to ease the learning task. In doing so, the system is made dependent on certain arbitrary choices on the part of the designers of the corpus. This is not a small affair.

Linguists have battled for decades over the structures of sentences, so it is not clear that the issue of sentence structure has been settled for good. Additionally, even if the structures proposed seem uncontroversial, there may evidence that they are incorrect or insufficient.

Alignment Based Learning (ABL, see for example [7]) has come to rescue here. Its aim is to propose learning methods from the raw input, the string. The theoretical underpinnings are the theory by Harris on distribution classes. This is a very interesting development since it puts the structuralist dogma to test. Whether or not we are right in teaching our students that constituent classes are derived from distributional data is finally assessed experimentally rather than theoretically. As much as I favour the pursuit of this paradigm, I have doubts as to whether this can ever be effectively used in establishing the structure of sentences.

My basic worry is this: there is overwhelming evidence that structure determines meaning. Chomsky himself has provided support for this. This means that the same sentence can have several concurrent structures. Mostly, the choice between them is arbitrary, and it means that the sentence is ambiguous. However, purely alignment based strategies may see constituents where there aren't any. Consider the infamous cross-serial dependencies of Dutch. On the face of it, we can assign them the same structure as the German counterparts, namely a nesting order. This would violate the association of verbs and their subject but distributionally I see nothing wrong with it. Thus, if the crossing versus nesting order is a matter of semantics and not syntax there is no way it can be unambiguously learned. Basically, the system will generate endless concurrent analyses with no means to discriminate between them. I have argued at length in [4] that with a notion of semantics as encoding truth conditions only the sentence structure of Dutch is provably crossing and not nesting.

The remedy for this, I claim, cannot and should not be looked for in refining the strategy. Rather, we should seriously start looking around for extraneous sources of learning. In particular, I claim, adding a modicum of semantic information will actually make a big difference.

3. Referent Systems

Referent systems were developed by Kees Vermeulen in [8] to overcome a difficulty in dynamic semantics, namely the "outsourcing" of the choice of variable names. DRT and Dynamic Semantics both operate with free variables but have no means to assign them. In both frameworks, variables names are global, and if two DRSs use the same variable name, they are talking about the same object. This has many drawbacks. In Referent Systems, by contrast, variable names are local and needed only during merge.

Internally, the DRT may use variables names that are different ("anonymous variables" or "referents") but they are hidden to the outside world. When two systems merge, variables are made disjoint by default. However, during merge the systems may negotiate identity for two of their referents in which case they will end up being the same (anonymous) variable. In programming languages, this is a known scenario. Procedures are defined using variables whose scope is unless otherwise declared, local. A procedure is invoked by plugging in values, and the position of the values indicates with which variables of the procedure they are to be associated.

We shall first give a reduced picture. An **identifier string** is a binary string preceded by a letter. Identifier strings are used to identify variables. We shall use Roman font for them; a variable in italics is actually a metavariable, it is only proxy for an identifier string. This is in order to stress that there is no additional substitutional device necessary. A **referent system** is a set of pairs $\langle x : N\rangle$ such that (a) $\langle x : N\rangle, \langle x : N'\rangle \in S$ implies $N = N'$, and (b) $\langle x : N\rangle, \langle x' : N\rangle \in S$ implies $x = x'$. The merge of referent systems is defined as follows.

$$\begin{aligned} S \circledR T := \; & \{\langle x0 : N\rangle : \langle x : N\rangle \in S\} \\ & \cup \{\langle x1 : N\rangle : \langle x : N\rangle \in T \text{ and for no } y : \langle y : N\rangle \in S\} \end{aligned} \tag{1}$$

This definition does not reveal enough of what is going on. Let us therefore define the following *substitution*:

$$\begin{aligned} \sigma_{S;T} := \; & \{\langle x, x0\rangle : \text{there is } N{:}\langle x : N\rangle \in S\} \\ & \cup \{\langle x, y0\rangle : \text{there is } N{:}\langle x : N\rangle \in T, \langle y : N\rangle \in S\} \\ & \cup \{\langle x, x1\rangle : \text{there is } N{:}\langle x : N\rangle \in T, \text{but no } y{:}\langle y : N\rangle \in S\} \end{aligned} \tag{2}$$

Let σ be a substitution; write

$$\sigma(S) := \{\langle \sigma(x) : N\rangle : \langle x : N\rangle \in S\} \tag{3}$$

Then we can write

$$S \circledR T := \sigma_{S;T}(S) \cup \sigma_{S;T}(T) \tag{4}$$

With S a referent system, we say N is the **name** of x iff $\langle x : N\rangle \in S$. By the definition of referent systems, names are unique for a variable, but not every variable needs to have a name. Also, variables are unique for names, but not every name needs to be assigned to some variable.

Referent systems can be used in connection with DRT or Dynamic Semantics as follows. A **semantic structure** is a pair $\langle S, \Delta\rangle$, where S is a referent system and Δ is a DRS. It is not required that S gives a name to every variable of Δ nor do we ban names for variables that do not occur in

Δ. Lets provide a realistic example, an entry for the Latin verb `videre`.

(5)
$$\begin{array}{|ll|}\multicolumn{2}{c}{\texttt{/videt/}}\\\hline \multicolumn{2}{|c|}{\langle x : 3.\textsc{nom.sg}\rangle, \langle y : \textsc{acc}\rangle}\\\hline \multicolumn{2}{c}{}\\\hline \multicolumn{2}{|c|}{e}\\\hline \mathsf{see}'(e); & \mathsf{exp}'(e, x);\\ \mathsf{thm}'(e, y). & \\\hline\end{array}$$

The lower part is a DRS, whose content is that there is an event *e* of seeing whose experiencer is x and whose theme is y. The upper part is a referent system which declares that the name of x is 3.SG.NOM, while the name of y is ACC. The upper part is what we call **argument structure**; it consists in turn of one or several **argument identifier statements**. If this structure is merged with another structure the outcome will depend on whether or the variables share names. By way of example, let us consider the following three entries:

(6)
$$\begin{array}{|c|}\multicolumn{1}{c}{\texttt{/senator/}}\\\hline \langle x : 3.\textsc{nom.sg}\rangle\\\hline \multicolumn{1}{c}{}\\\hline x\\\hline \mathsf{senator}'(x).\\\hline\end{array}\qquad \begin{array}{|c|}\multicolumn{1}{c}{\texttt{/rosam/}}\\\hline \langle x : \textsc{acc}\rangle\\\hline \multicolumn{1}{c}{}\\\hline x\\\hline \mathsf{rose}'(x).\\\hline\end{array}\qquad \begin{array}{|c|}\multicolumn{1}{c}{\texttt{/consuli/}}\\\hline \langle x : \textsc{dat}\rangle\\\hline \multicolumn{1}{c}{}\\\hline x\\\hline \mathsf{consul}'(x).\\\hline\end{array}$$

Then we have

(7)
$$\begin{array}{|c|}\multicolumn{1}{c}{\texttt{/senator/}}\\\hline \langle x : 3.\textsc{nom.sg}\rangle\\\hline \multicolumn{1}{c}{}\\\hline x\\\hline \mathsf{senator}'(x).\\\hline\end{array}\ \bullet\ \begin{array}{|ll|}\multicolumn{2}{c}{\texttt{/videt/}}\\\hline \multicolumn{2}{|c|}{\langle x : 3.\textsc{nom.sg}\rangle, \langle y : \textsc{acc}\rangle}\\\hline \multicolumn{2}{c}{}\\\hline \multicolumn{2}{|c|}{e}\\\hline \mathsf{see}'(e); & \mathsf{exp}'(e, x);\\ \mathsf{thm}'(e, y). & \\\hline\end{array}$$

$$= \begin{array}{|ll|}\multicolumn{2}{c}{\texttt{/senator videt/}}\\\hline \multicolumn{2}{|c|}{\langle x0 : 3.\textsc{nom.sg}\rangle, \langle y1 : \textsc{acc}\rangle}\\\hline \multicolumn{2}{c}{}\\\hline \multicolumn{2}{|c|}{e1, x0}\\\hline \mathsf{see}'(e1); & \mathsf{exp}'(e1, x0);\\ \mathsf{thm}'(e1, y1); & \mathsf{senator}'(x0).\\\hline\end{array}$$

The sentence `senator videt rosam` can be built in two different ways; first by merging `senator` and `videt` (as shown above) and then merging the result with `rosam`, or by first forming the constituent `videt rosam`.

The result is slightly different:

(8)

$$\begin{array}{c}
\texttt{/((senator videt) rosam)/}\\
\begin{array}{|c|}\hline \langle \mathrm{x00} : 3.\textsc{nom.sg}\rangle, \langle \mathrm{y10} : \textsc{acc}\rangle \\ \hline \end{array}\\
\begin{array}{|ll|}\hline \multicolumn{2}{|c|}{\mathrm{e10}, \mathrm{x00}, \mathrm{y01}}\\ \hline
\text{see}'(\mathrm{e10}); & \text{exp}'(\mathrm{e10}, \mathrm{x00});\\
\text{thm}'(\mathrm{e10}, \mathrm{y10}); & \text{senator}'(\mathrm{x00});\\
\text{rose}'(\mathrm{y10}). & \\ \hline\end{array}\\
\\
\texttt{/(senator (videt rosam))/}\\
\begin{array}{|c|}\hline \langle \mathrm{x0} : 3.\textsc{nom.sg}\rangle, \langle \mathrm{y01} : \textsc{acc}\rangle \\ \hline \end{array}\\
\begin{array}{|ll|}\hline \multicolumn{2}{|c|}{\mathrm{e01}, \mathrm{y01}, \mathrm{x0}}\\ \hline
\text{see}'(\mathrm{e01}); & \text{exp}'(\mathrm{e01}, \mathrm{x0});\\
\text{thm}'(\mathrm{e01}, \mathrm{y01}); & \text{senator}'(\mathrm{x0});\\
\text{rose}'(\mathrm{y01}). & \\ \hline\end{array}
\end{array}$$

Since the identifier string (what we have called the variable) is unimportant, it is allowed to perform a substitution as long as it is bijective. Thus the two structures are equivalent. To be more precise, a representation $\langle S, \Delta\rangle$ is satisfied in a first-order structure $\langle M, \mathfrak{I}\rangle$ iff there is an assignment β from variables to objects in M such that $\langle M, \mathfrak{I}, \beta\rangle \vDash \Delta$. The triple $\langle M, \mathfrak{I}, \beta\rangle$ is also called a **model**. This is different from the standard DRT approach which assumes that DRSs are not evaluated in structures but in models; a DRS places restrictions on valuations, whence its dependence on the actual identifier string. (Actually, since that is so we can dispense with putting the variables into the head section of the box. This move requires care, however, since at some point it will become necessary to do so, for example when the DRS is embedded. But the added flexibility may actually be welcome. In this paper we shall continue the standard practice, though.)

However, look what happens if the names of the variables do not match.

(9)

$$\begin{array}{c}
\texttt{/consuli/}\\
\begin{array}{|c|}\hline \langle \mathrm{x} : \textsc{dat}\rangle \\ \hline \end{array}\\
\begin{array}{|c|}\hline \mathrm{x}\\ \hline \text{consul}'(\mathrm{x}). \\ \hline\end{array}
\end{array}
\bullet
\begin{array}{c}
\texttt{/videt/}\\
\begin{array}{|c|}\hline \langle \mathrm{x} : 3.\textsc{nom.sg}\rangle, \langle \mathrm{y} : \textsc{acc}\rangle \\ \hline \end{array}\\
\begin{array}{|ll|}\hline \multicolumn{2}{|c|}{\mathrm{e}}\\ \hline
\text{see}'(\mathrm{e}); & \text{exp}'(\mathrm{e}, \mathrm{x});\\
\text{thm}'(\mathrm{e}, \mathrm{y}). & \\ \hline\end{array}
\end{array}$$

$$= \begin{array}{c}
\texttt{/consuli videt/}\\
\begin{array}{|c|}\hline \langle \mathrm{x1} : 3.\textsc{nom.sg}\rangle, \langle \mathrm{y1} : \textsc{acc}\rangle \\ \langle \mathrm{x0} : \textsc{dat}\rangle \\ \hline \end{array}\\
\begin{array}{|ll|}\hline \multicolumn{2}{|c|}{\mathrm{e1}, \mathrm{x0}}\\ \hline
\text{see}'(\mathrm{e1}); & \text{exp}'(\mathrm{e1}, \mathrm{x1});\\
\text{thm}'(\mathrm{e1}, \mathrm{y1}); & \text{consul}'(\mathrm{x0}). \\ \hline\end{array}
\end{array}$$

This is a representation that is true in a structure iff there is an event of seeing (with an experiencer and a theme) and if there is a consul.

4. Adding Fine Structure

The previous section has shown that the merger of two structures does not need any external source for choosing the identifier strings (= variables); they can be negotiated between the referent systems. However, this needs to be fine tuned. The first problem is that names cannot be changed or eliminated one assigned. To remedy this, I have introduced a system of so-called **vertical diacritics**. Each variable is associated with one of the following diacritics: $-$, $\vartriangle$, $\triangledown$ and $\Diamond$. If the diacritic is $\triangledown$ we say that the variable is **imported**, and if it is $\vartriangle$ we say that it is exported. $\Diamond$ is actually an abbreviation of $\{\vartriangle, \triangledown\}$, and so the variable is both imported and exported; $-$ likewise is the abbreviation of the empty set of vertical diacritics. The variable is neither imported nor exported (and therefore what we call **nameless**). Syntactic arguments belong to variables that are just imported, adjuncts to variables that are both imported and exported. At most one variable may be exported. You may think of it as specifying the result.

If two referent systems meet, a variable must be argument in one of them and result in the other. So, in the merge a $\vartriangle$ of one structure is cancelled together with a $\triangledown$ of the other. If no such pair exist, the merge is undefined. Here is a 'multiplication table' for these diacritics ($*$ means: not defined):

(10)

	$-$	$\triangledown$	$\vartriangle$	$\Diamond$
$-$	$*$	$*$	$*$	$*$
$\triangledown$	$*$	$*$	$-$	$\triangledown$
$\vartriangle$	$*$	$-$	$*$	$\vartriangle$
$\Diamond$	$*$	$\triangledown$	$\vartriangle$	$\Diamond$

We say the variable is a **head** if the diacritic is $\triangledown$, it is an **argument** if the diacritic is $\vartriangle$; the variables is an **adjunct** if the diacritic is $\Diamond$. The remaining case of $-$ is factually unimportant and will be omitted below.

Further, if a variable is argument or adjunct, a **horizontal diacritic** may be added. The horizontal diacritic is one of ⧁, ⧀ and ⦶. ⧁ means that the other referent system must be to the right; ⧀ means that the referent system must be to the left; ⦶ means that it may be both to the right or to the left.

The table above is now extended as follows.

(11)

	▽⧁	▽⧀	△	◊⧁	◊⧀
▽⧁	∗	∗	–	▽⧁	▽⦶
▽⧀	∗	∗	∗	∗	∗
△	∗	–	∗	∗	△
◊⧁	∗	▽⦶	△	◊⧁	◊⦶
◊⧀	∗	▽⧀	∗	∗	◊⧀

For ⦶ take the union of the results with ⧁ and ⧀. Some of the combinations with ◊ are actually tricky; we shall not go into the complications here.

We allow names to be drawn from an attribute value matrix, with underspecified values. This allows names to match despite being nonidentical; all that is required is that they can be unified. For example, `videt` expects its object under the name ACC. The entry `senatorem` will list it as 3.SG.ACC, which means it is accusative, 3rd, singular. The two unify, and so the variables can be identified.

Names can be lost or changed. If the diacritic is ▽ this means that the name associated with the variable is lost after merge. This option is very useful. It means that one does not accumulate names ad infinitum. However, if the diacritic is ◊ there is an option of changing the name. The notation is $A \mapsto B$, which means that the name of the variable is A before merge and B thereafter. For example, a nominative case suffix will change the name of the variable from one that has no case to one that has nominative case. In this way, morphology connects with referent systems in adding detail to the argument structure.

There is a general requirement that argument structures must be complete when they are merged. This means that they do not need an argument themselves. Say that a an argument structure is a **phrase** if it contains only one argument identification statement. Notice that this treats adjectives and adverbs differently from verbs. Adjectives, adverbs and adjuncts in general can be phrases despite the fact that they still need an argument. However, there is a distinction between `aware` and `blue` in that the latter is not a phrase because it needs an argument, a PP headed by `of`. This mirrors observations in GB that heads takes as arguments and specifiers. This rule does have exceptions, though. We allow heads to have one of the diacritics ▼, ♦; they signal that the head may take another head H' as its argument regardless of whether H' head needs further arguments. However, there is a proviso that H' has not combined yet with anything. (This requires tracking the derivation; ideally, I would like to put some diacritic here that says whether the object counts as complex but I am unsure as to what the exact conditions are to be placed on being 'simple'.) Merge with this diacritic is called **fusion**. Fusion allows for clause union. It allows for example raising infinitives to combine with each other to form a complex verb that takes an unlimited number of arguments. Their order is controlled by the argument structure, so no confusion arises. At the moment of merge it needs to be decided where the argument of H' should be placed within the argument structure of the selecting head. They could be placed at the beginning or at the end, and this will result in basically either nesting or crossing dependencies. German and Dutch raising infinitives are built using fusion. (This is the standard analysis which creates a verb cluster.)

The last piece to be added is that the referent systems are not sets but lists and that merge typically proceeds by identifying one and only one variable in each of the referent systems. Here is how an entry looks like. The example is the verb form `lets`.

(12)
$$\begin{array}{c}
\texttt{/lets/} \\
\boxed{\begin{array}{c}
\left\langle \mathrm{e0} :\triangle: \begin{bmatrix} \mathrm{cat} & : & \mathit{ev} \\ \mathrm{agr} & : & \mathit{fin} \end{bmatrix} \right\rangle \\
\left\langle \mathrm{x} : \triangledown\olessthan : \begin{bmatrix} \mathrm{num} & : & \mathit{sg} \\ \mathrm{cat} & : & \mathit{ob} \end{bmatrix} \right\rangle \\
\left\langle \mathrm{e} : \triangledown\ogreaterthan : \begin{bmatrix} \mathrm{cat} & : & \mathit{ev} \\ \mathrm{agr} & : & \mathit{inf} \end{bmatrix} \right\rangle \\
\left\langle \mathrm{y} : \triangledown\ogreaterthan : \begin{bmatrix} \mathrm{cat} & : & \mathit{ob} \end{bmatrix} \right\rangle
\end{array}} \\
\boxed{\begin{array}{c} \mathrm{e} \end{array}} \\
\boxed{\begin{array}{ll}
\mathsf{thm}'(\mathrm{e},\mathrm{f}); & \mathsf{let}'(\mathrm{e0}); \\
\mathsf{ben}'(\mathrm{e},\mathrm{y}); & \mathsf{agt}'(\mathrm{f},\mathrm{y}); \\
\mathsf{agt}'(\mathrm{e},\mathrm{x}). &
\end{array}}
\end{array}$$

It takes two kinds of arguments: objects (x, y) and events (f), and it itself passes up an event (e). The first argument is y, which is the object of `lets`; the next argument is the event f. Notice that by having the clause $\mathsf{agt}'(\mathrm{f},\mathrm{y})$ the semantics sees to it that y is automatically the subject of the embedded clause. The last argument is the subject, x. It is the one that is found to the left (whence the diacritic $\olessthan$). No directionality is specific for e because it is not argument, so the directionality is specified only in the head that selects it as argument.

The part above the DRS is from now on called **argument structure**. It is a list of argument identification statements (AISs). The first part of such an AIS is called its **main** variable; the other variables are parameters. There are two kinds of access rules for argument structures. The **configurational access** determines that merge can use only the last member of the argument structure. If access is configurational, the order in which the elements appear in the argument structure projects into the phrase in perfect mirror: the least element is the closest in the tree. If access is **nonconfigurational** then merge can jump over an identification statement (counting from the end) provided merge of the AISs would not succeed. The first succeeding match is taken. Thus the way in which the order projects into a tree may well be different from the way they appear in the list. English uses configurational access, German nonconfigurational access. Thus, English verbs are strict in their order requirements, while German verbs may take their arguments in any order they please. However, notice that a sentence in which two arguments are ambiguous between nominative and accusative turn out to be ambiguous.

(13) `..., dass die Mutter eine Maus sieht.`
... that the mother a mouse sees

This is predicted as follows. The words `die`, `eine`, `Mutter` and `Maus` all have two argument structures: one where the argument is nominative, and one where it is accusative. The verb is looking for two arguments, the first nominative and the second accusative. The accusative, being second in the list, is to be consumed first. The next item to the left may be accusative or nominative. If the first, it merges with the last entry; if the second, it merges with the first entry. The remaining complex is now looking for a nominative in the first case and an accusative in the second case. If the next argument satisfies this requirement, it is taken in. In the present case, this is the case. There is another type of sentence which is unambiguous:

(14) `..., dass den Kater die Mutter sieht.`
..., that the.ACC tomcat the mother sees

Notice that this requires nonconfigurational merge, since both arguments are on the left hand side. English is a rather tricky case. In a normal declarative sentence the arguments are on opposite sides of the verb so this offers no decisive evidence for or against configurational merge. (In fact, quantifier raising would be evidence against it.) Yet, a topicalised structure will not allow the nominative to precede the accusative, indication the English might after all use configurational merge. The decision must be based on looking at further arguments of the verb.

5. Implementation

The referent systems have been implemented in OCaML. At present, only the basic system has been installed, with no recursive structure on the DRSs, and no variables for them. All the phenomena talked about in this paper are however fully covered. In particular, it is possible to implement the behaviour of parameters. The system can be downloaded from `http://kracht.humnet.ucla.edu/marcus/referent`. Rudimentary dictionaries of German and English can be downloaded as well, which illustrate the basic features of the system. Instructions (as well as an online demo) can be found on the site. I plan a next phase where full DRSs can be included and thus logical connectives may be modeled. At present it is built mainly for testing the theory, so the main design criterion is faithfulness to the theory. At present it uses a chart parser that computes a forest of representations, which are alphabetically compressed and reduced modulo equivalence at each stage. The compression is perhaps a rather time consuming part, but considering the small size of sentences this does affect performance very much; it greatly enhances readability, though. To gain efficiency, one may consider to just compute the derivation terms before evaluating the DRSs, but for testing purposes this will not significantly reduce the computations.

6. Parameters

One advantage of the system comes with the introduction of parameters. While during merge there is exactly one variable that is officially being shared via handshake, each of the variables may drag along an unlimited number of parameters with them that are shared during merge as well. The way it works is as follows. There is a set of so-called **parameter roles**. Any variable can fill in any parameter role as long as it has the correct sort. For time, for example, we assume three roles: **reference time**, **story time** and **predication time**. In the present tense, all three are equal (and may therefore be filled by the same variable). In the past, only story time and predication time coincide, and they precede reference time. In the pluperfect, all three are different. A variable that has diacritic $\Diamond$ may associate two variables with each parameter role; one for the value it imports and one which it exports to its head. This allows for sequence of tense phenomena to be accounted for. We attribute different characteristics of the complementizer in English and Russian. In Russian, the complementizer passes on the story time variable of the superordinate clause as the reference time of the subordinate clause. In this way, subordinate clause tenses are as if seen from the perspective of the main event. In English, subordinate reference time equals main reference time, so past tense is mandatory even when the two events happen at the same time.

(15) `I said that I read the book.`

The mechanics of tenses is quite subtle, but the parameters can accommodate quite a number of different schemata. For example, in the subordinate clause the use of past tense signals ‘predication time same as main clause reference time’ or even, more simply, ‘predication time is now’. For subsequent events, the subjunctive is mandatory and for anterior events the pluperfect.

(16) `I said that I would read the book.`

(17) `I said that I had read the book.`

A semantic solution to this is to make the choice of tenses conditional on the relative position of reference time and story time of the superordinate structure.

The way parameter handling is implemented is as follows. For every parameter role one may add a statement that declares which variable assumes that role. If the diacritic is $\vartriangle$ this statement takes the form $[R : x]$, where R is a role and x a variable identifier. If the diacritic is $\triangledown$ it also takes this form. For example, assume we merge the following two referent systems:

(18) $\langle u : \triangledown\ogreaterthan : [\textsc{cat} : ev] :: [\textsc{ptime} : t]\rangle ⓢ \langle v :\vartriangle: [\textsc{cat} : ev] :: [\textsc{ptime} : t1]\rangle$

Then u and v will both be mapped to u0 because they match; additionally, both have an entry for the role ‘PTIME’, and thus additionally the following substitution is made: t is mapped to t0, and t1 is also mapped to t0. Thus, upon succesful merge, the substitution mechanism is extended to the parameters that have the same role. If for example the argument had a parameter RTIME with value t01, this variable would be mapped t011, if the head does not contain any statement of the form $[\textsc{rtime} : w]$ for any w. If it did, then w would be sent to $w0$, as would be t01. Thus, in addition to the main variable any number of parameters can be mapped to each other. Finally, note that if the diacritic is $\Diamond$ the parameter statements take the form $[R : x \mapsto y]$, which means that x has role R when imported from an argument, but it is y which is R when exported (upstairs). For example, look at the entry of `damalig`.

(19)
$$\begin{array}{|c|}
\multicolumn{1}{c}{\texttt{/damaliger/}} \\ \hline
\left\langle \mathrm{x} : \Diamond\ogreaterthan : \begin{bmatrix} \mathrm{num} & : & sg \\ \mathrm{gen} & : & m \\ \mathrm{cat} & : & ob \\ \mathrm{case} & : & nom \end{bmatrix} :: \begin{bmatrix} \mathrm{stm} & : & \mapsto \mathrm{t} \\ \mathrm{ptm} & : & \mathrm{t}\mapsto\mathrm{t0} \end{bmatrix} \right\rangle \\ \hline
\\ \hline
\\ \hline
\end{array}$$

This says the following: the adjective takes its argument to the right and imports t as the predication time. It passes up that value as the story time. The verb has a story time, but it may be different from the reference time. So, the net effect is that whatever is predicated of the individual denoted by x is valid at the time of the story. Actually, I am not certain whether `damalig` connects only to the story time. I suspect it can also connect to the predication time. This creates subtle ambiguities, as in

(20) `Der Senator hatte dem damaligen Minister Geld geliehen.`
the senator had lent money to the then minister

The sentence may be saying that the person the senator had given money to was a minister at the time of giving the money or at the time the story is evolving. If that is so, then an additional entry must be entered into the lexicon:

(21)
$$\begin{array}{c} \texttt{/damaliger/} \\ \begin{array}{|c|} \hline \left\langle \mathrm{x} : \Diamond\ogreaterthan : \begin{bmatrix} \mathrm{num} & : & \mathit{sg} \\ \mathrm{gen} & : & \mathit{m} \\ \mathrm{cat} & : & \mathit{ob} \\ \mathrm{case} & : & \mathit{nom} \end{bmatrix} :: \begin{bmatrix} \mathrm{ptm} & : & \mathrm{t}\mapsto\mathrm{t} \end{bmatrix} \right\rangle \\ \hline \\ \hline \\ \hline \end{array} \end{array}$$

Without the word `damalig` it would be unclear which time point we are actually speaking about. This means that the noun phrase does not export its predication time in general; it only does so when a specific temporal modifier is present that relates it to time points set by the verb. Others set the time relative to external clocks (`heutig` 'today's', `gestrig` 'yesterday's', `ehemalig` 'ex-'). These can be dealt with assuming that we have anchoring devices. This is not the point to discuss them, however.

7. Morphology

The other important trait of referent systems is that they allow to capture the contribution that morphology makes. Morphology sometimes makes clear semantic contributions (plural, causatives, 'semantic' cases) but more often than not it simply provides clues for the structure of a sentence. We have seen above how cases can help in working out the meaning of a sentence in German. But there is more. In many languages we find *agreement*, and its nature from a semantic and syntactic point of view is not always clear. Negation markers, for example, may either signal negation or they may signal agreement with an already present negation marker. The difference is that an agreement marker for negation does not invert the polarity of the sentence, whereas a negation marker does. Agreement is treated here on the line of GPSG/HPSG as sharing of certain features. Basically, the default is that all features must be shared, but that need not always be the case. We shall not discuss the ramifications. Basically, an adjunct that appears obligatorily to the left contains the following argument identification statement:

(22)
$$\left\langle x : \lozenge\ogreaterthan : \begin{bmatrix} \textsc{att}1 : \mathit{val1} \\ \textsc{att}2 : \mathit{val2} \\ \cdots \\ \textsc{attn} : \mathit{valn} \end{bmatrix} \right\rangle$$

(We ignore parameters for this discussion.) This says that what it modifies has to have certain features, and after modification they remain the same. Now, the stem of an adjective, say Latin `magn` 'big' will be drawn from the lexicon like this:

(23)
$$\begin{array}{|c|} \multicolumn{1}{c}{/\mathtt{magn}/} \\ \hline \left\langle x : \lozenge\obar : \begin{bmatrix} \textsc{cat} & : \mathit{ob} \\ \textsc{per} & : \ \mathit{3} \\ \textsc{gen} & : \ * \\ \textsc{num} & : \ * \\ \textsc{case} & : \ * \end{bmatrix} \right\rangle \\ \hline \varnothing \\ \hline \mathsf{big}'(x). \\ \hline \end{array}$$

$*$ is a special value. It is not to be confused with underspecified. $*$ does not match anything but $*$. Gender is installed with a particular morpheme, like this one.

(24)
$$\begin{array}{|c|} \multicolumn{1}{c}{/\textsc{fem}/} \\ \hline \left\langle x : \blacklozenge\olessthan : \begin{bmatrix} \textsc{cat} & : & \mathit{ob} \\ \textsc{per} & : & \mathit{3} \\ \textsc{gen} & : & * \mapsto \mathit{fem} \\ \textsc{num} & : & * \\ \textsc{case} & : & * \end{bmatrix} \right\rangle \\ \hline \varnothing \\ \hline \varnothing \\ \hline \end{array}$$

Apply this to `magn` and one gets

(25)
$$\begin{array}{|c|} \multicolumn{1}{c}{/\mathtt{magna}/} \\ \hline \left\langle x : \lozenge\obar : \begin{bmatrix} \textsc{cat} & : \ \mathit{ob} \\ \textsc{per} & : \ \mathit{3} \\ \textsc{gen} & : \mathit{fem} \\ \textsc{num} & : \ * \\ \textsc{case} & : \ * \end{bmatrix} \right\rangle \\ \hline \varnothing \\ \hline \mathsf{big}'(x). \\ \hline \end{array}$$

(The fact that `magn` plus FEM yields `magna` is no analytical statement, only illustrative. We are not concerned with the actual form of these entries.) The next morpheme that is added is PL:

(26)
$$\begin{array}{|c|}
\multicolumn{1}{c}{/\text{PL}/} \\ \hline
\left\langle x : \blacklozenge\olessthan : \begin{bmatrix} \text{CAT} & : & ob \\ \text{PER} & : & 3 \\ \text{GEN} & : & \top \\ \text{NUM} & : & * \mapsto pl \\ \text{CASE} & : & * \end{bmatrix} \right\rangle \\ \hline
\varnothing \\ \hline
\varnothing \\ \hline
\end{array}$$

Notice that this morpheme has gender value $\top$, which means that it can be added only after gender was installed.

This system therefore allows each individual morpheme to contribute to the semantic representation. Each adjective in Latin shows agreement, and this is reflected in the fact that the argument structures are as given above. Notice however that the morphemes all operate via fusion, so they require the argument to be a simple word. This forces the repetition of agreement at every word. If we had placed $\lozenge$ instead, a different behaviour would emerge. In that case the morpheme would have to attach to the entire phrase rather than the word. This is the case in Hungarian. In Hungarian, prenominal adjectives do not agree with the noun. Rather, agreement (in number and case) is signaled only once at the end.

(27) `a nagy fehér ház-ok-ban`
det big white house-PL-INESS

So, the plural suffix in Hungarian is rather as follows (there is no morphological gender in Hungarian).

(28)
$$\begin{array}{|c|}
\multicolumn{1}{c}{/\mathtt{k}/} \\ \hline
\left\langle x : \lozenge\olessthan : \begin{bmatrix} \text{CAT} & : & ob \\ \text{PER} & : & 3 \\ \text{NUM} & : & * \mapsto pl \\ \text{CASE} & : & * \end{bmatrix} \right\rangle \\ \hline
\varnothing \\ \hline
\varnothing \\ \hline
\end{array}$$

This suffix can be added to `ház` alone, but after having been added the adjective is no longer in a position to modify it. Here I assume basically that Hungarian adjectives do not inflect, and that their entries are like Latin roots, see `magn` above. But there is an alternative structure, namely adding the plural and the case suffix to the entire phrase:

(29) ((`a nagy fehér ház`)`ok`)`ban`

The present analysis can extended to many more phenomena of agreement. Notice that it is typical of agreement that the semantics proper is actually void. The agreement maker contributes no further information beyond the fact that the element is considered to be of a particular category. This ensures that a particular feature, say plural, does get to contribute its meaning over and over again. This is also dangerous. There are plenty of pluralia tanta whose meaning is not necessarily plural (Latin `litterae` 'letter' is a case in point). Adjectives will show plural agreement but the plural morpheme must be barred from contributing its original meaning.

This is a pervasive phenomenon. A morpheme may either function as a content word or as a grammatical word. The distinction is that in the latter case it ceases to contribute any meaning. I have shown for example in [3] that the same noun phrase, here the ablative of Finnish `laiva` 'ship' can have three different meanings depending on whether the morpheme that is added is considered a grammatical or a contentful element. The grammatical use of an element is that use which manipulates the feature structure. The negation marker, for example, if used grammatically, simply changes the polarity.

8. Complexity

Some words on the overall performance of the system. Though I have not done any large scale implementation of it, it is not hard to show that the complexity of these structures is very low. This is in sharp contrast with λ-calculus, where normalisation is at least exponential unless the type structure flat, that is, of the form $b_1 \to b_2 \to \cdots \to b_n \to a$ for basic types a and $b_1, \ldots, b_n$ (see [6] for upper bounds on the lengths of normalisation sequences). In the latter case, functions are directly applied to their arguments. Still, there are substitutions to be made. With referent systems, the result of a merge is computed as follows.

(1) It needs to be seen which of the variables in the two referent systems can be identified. In configurational merge this is a matter of looking up the last element in the two systems and see if they match the first one of the other. This is a matter of unifying two AVSs. Given that AVSs are of bounded size, this takes constant time. The substitution is then computed, taking time linear in the length of the argument identification statements. Execution of the substitution is linear in length of the DRSs. Thus, a structure (or parse) term is evaluated in quadratic time.

(2) When doing a parse, each merge can increase the length of the involved DRSs. Basically, an upper bound for the merge is the sum of the

individual DRSs (we ignore the lengths of the variable identifiers, which is legitimate). Thus, each merge step can take at most the time linear in the sum of all involved structures.

Thus, let L be a lexicon, with q the maximal length of an entry in L. Let $\vec{x}$ be a string of length n. Each parse will take $n - 1$ steps, with overall cost $O(q(n-1)n/2) = O(n^2)$. This is for the computation of a single parse. For a forest, matters are slightly differently.

Two cases have to be considered. (Case A) There is no fusion. Then the grammar is basically context free. Then parsing takes $O(n^3)$ steps, and it turns out that the entire forest is computed in $O(n^4)$ time (not $O(n^5)$, as one might expect). (Case B) There is fusion. Then matters are more difficult. The fusion itself is straightforward. The only possibility is to take the head that is available for fusion right away. There is a theoretical possibility that arguments appear at alternating sides of the head and that can create an exponential search space. Similarly, with noncofigurational access the complex head as exponentially many possibilities for creating constituents. This explosion is not only rare, it actually can be eliminated. We shall not go into detail here, though.

Finally, complexity of parsing can be further reduced by noting that it is not necessary to actually compute the value of the parse term. All that is needed is to record the argument structure without the actual variable identifiers or parameter statements. There is only a finite array of these structures in the nonfusional case, which essentially reduced to a context free grammar. In the case with fusion we get a linear indexed grammar. Once it is established that there is a term t for the entire string, it can be evaluated in quadratic time. Thus parsing is $O(n^3)$ without fusion and $O(n^6)$ with fusion.

9. Conclusion

Referent systems allow to do the administration of variables completely locally. This eliminates the need to have parsers come up with indexing; also, they do not use λ-abstraction, which generally introduces a lot of costly computations. The administration is flexible, and we have shown above some ways of handling it. There certainly are more; it is beyond the scope of this paper to discuss them all. Instead I refer to [2]. A very important addition to the referent systems designed by Kees Vermeulen is the notion of a parameter. This powerful device allows to mimic the handling of indices and context parameters.

The present approach has various shortcomings, though. One is that it is basically centered around the idea of continuous constituents. It therefore has a hard time coping with phenomena such as verb second and split

DP not to mention the massive discontinuity found in Australian languages. Some of this can be made up for by assuming somewhat stronger string handling devices. Suppose, for example, constituents can consist of two parts. Then it is possible to generate verb second simply by listing verbs as constituents consisting of a part that moves plus a (possibly empty) remnant. The directionality is then set from the point of view of the righthand position of the remnant. This solution approaches syntax more in the style of morphology: the verb in German is considered a transfix, which inserts one part into second place and another at the end.

References

[1] Rens Bod, Remko Scha, and Khalil Sima'an, editors. *Data-Oriented Parsing*. Studies in Computational Linguistics. CSLI, 2003.
[2] Marcus Kracht. Agreement Morphology, Argument Structure and Syntax. Manuscript, 1999.
[3] Marcus Kracht. Against the Feature Bundle Theory of Case. In Ellen Brandner and Heike Zinsmeister, editors, *New Perspectives on Case Theory*, pages 165 – 190. CSLI, 2003.
[4] Marcus Kracht. The Emergence of Syntactic Structure. Manuscript, UCLA, 2005.
[5] D. Osherson, D. de Jongh, E. Martin, and S. Weinstein. Formal learning theory. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*, pages 737 – 775. North-Holland, Amsterdam, 1997.
[6] Helmut Schwichtenberg. An upper bound for reduction sequences in the typed λ-calculus. *Archive for Mathematical Logic*, 30:405–408, 1991.
[7] Menno van Zaanen. Alignment Based Learning versus Data Oriented Parsing. In Rens Bod, Remko Scha, and Khalil Sima'an, editors, *Data-Oriented Parsing*, Studies in Computational Linguistics. CSLI, 2003.
[8] Kees F. M. Vermeulen. Merging without mystery or: Variables in dynamic semantics. *Journal of Philosophical Logic*, 24:405 – 450, 1995.

Department of Linguistics, UCLA, 3125 Campbell Hall, PO Box 951543, Los Angeles, CA 90095-1543, `kracht@humnet.ucla.edu`