# Referent Systems and Argument Structure

Kracht Marcus

Magyar Tudományos Akadémia, Nyelvtudományi Intézet,
Benczur u. 33, 1068 Budapest, e-mail: kracht@nytud.hu
and Department of Linguistics, UCLA, 3125 Campbell Hall, PO Box 951543,
Los Angeles, CA 90095-1543, email: kracht@humnet.ucla.edu

**Abstract.** The assignment of variables plays a pivotal role in the construction of semantics of complex expressions. In this paper we discuss the theory and implementation of an algorithm to identify variable names. It is based on referent systems, introduced in [5]. The theory is exposed in [3] and discusses in-depth the properties of referent systems. [1]

**Keywords:** argument structure, referent systems

## 1 Introduction

A crucial problem in the composition of meanings is the problem of *variable names*. Montague originally devised a semantics that assigned closed expressions to each word, thus relegating the problem to the $\lambda$-calculus. However, [1] has pointed out that this is problematic in view of transsentential binding. He proposed an alternative that led to the development of discourse representation theory (DRT), which uses—at least in its original form—no $\lambda$-binding mechanism at all. All unquantified variables where free. What needed to be solved, then, was the assignment of variable names.

Kamp and Reyle describe in [2] an algorithm to derive the semantics of a sentence. This algorithm needs a parser that not only produces a structure but also distributes indices to the constituents. Thus, the input to the semantical translation of (1.a) is (1.b) rather than (1.c).

(1) (*a*) `Egy fekete macska látja az egeret.`
(*b*) [[$\texttt{egy}_1$ [$\texttt{fekete}_1$ $\texttt{macska}_1$]] [$\texttt{látja}_{1,2}$ [$\texttt{az}_2$ $\texttt{egeret}_2$]]]
(*c*) [[`egy` [`fekete` `macska`]] [`látja` [`az` `egeret`]]]

The reason for this is that in DRT every variable is global. The variable $x$ points to the same object independently of the DRS in which it occurs. To see this,

[1] The author wishes to thank Váradi Tamás and Kenesei István for their generous support.

look at the way two DRSs are merged. The phrase `egy fekete macska` consists of three DRRs, each of which uses at least one variable.

(2)

| /egy/ | /fekete/ | /macska/ |
|---|---|---|
| x | ∅ | ∅ |
| ∅ | $\mathsf{black}'(x)$ | $\mathsf{cat}(x)$ |

Merge is associative and consists in taking the set union of the upper and the low box, respectively. Merging the first two, for example, results in accidental capture of the variables of the second DRS by the quantifier of the first, like this:

(3)

| /egy macska/ |
|---|
| x |
| $\mathsf{black}'(x).$ |

Merging the upper three we get

(4)

| /egy fekete macska/ |
|---|
| x |
| $\mathsf{black}'(x); \mathsf{cat}'(x).$ |

The problem with this approach is that if, for example, the middle DRS uses $y$ in place of $x$ we get an incorrect result:

(5)

| /egy fekete macska/ |
|---|
| x |
| $\mathsf{black}'(y); \mathsf{cat}'(x).$ |

This is because the merge operation cannot know whether two variables in different DRSs are meant to be 'the same' or not. To solve this problem, [2] simply relegated the problem to the parser; it was the responsibility of the parser to distribute the correct indexation to each lexical entry. The indices, in addition to being useful for syntax, provided the essential information to insert the correct names for the variables. The index $i$ is simply translated by the variable $\mathtt{x}_i$. (Notice, by the way, that entries with several free variables need several indices, and the order matters.)

## 2 Referent Systems

[5] has shown that from a logical point of view there is no need to do indexing if variables are instead considered local. Instead of considering two variables of two DRSs identical if they carry the same string, we assume by default that the variables of distinct DRSs are different *unless stated otherwise*. To say that two variable are to be identified, we associate a so-called 'name' with a variable. (Names are optional; if a variable has no name, it simply cannot be identified.) In OCaML, such names have effectively the same mechanics (and are used for

similar purposes) as *labels*. Names can be everything, but the idea is that in natural languages names are morpho-sytactic properties, like cases and grammatical roles. When merging two structures two occurrences of the same variable (or of two different variables) are made the same in the output DRS if (and only if) they carry the same name. This type of variable is called a **referent**. With the help of referent systems the argument structure can be enriched in such a way that the indexation proceeds automatically. If an entry, say a verb, needs several arguments, we want to allow it to take each argument in turn. Most syntactic theories postulate a canonical deep order in which the arguments are consumed in the same way as programming languages insist on the arguments being fed to a

function in the order specified. In OCaML, for example, we may declare a function in the following way:

(6) `let f x y = 2 * x + y;;`

In this case the variables `x` and `y` are plain variables, and bound inside the function declaration. Order matters. Evaluating `f 3 2` gives 8, evaluating `f 2 3` given 7.

Freedom from this order regime comes in the form of *labels*. Consider this slightly different definitions using the tilde convention:

(7) `let f ~x:a ~y:b = 2 * a + b;;`

Here, `~x` and `~y` are labels; `a` and `b` are the associated variables. The advantage is that order is now irrelevant: `f ~x:2 ~y:3` and `f ~y:3 ~x:2` both yield 7, and `f ~x:3 ~y:2` and `f ~y:2 ~x:3` both yield 8. Basically, it is the *order independence* of this mechanism that we exploit.

## 3 Argument Structure

We aspire for a surface oriented approach, that is, we want to interpret every constituent where it actually occurs. Moreover, we require arguments to be *adjacent* to each other. Given these requirements we must accommodate free word order not by distinguishing two different syntactic representations, but by allowing arguments to be identified by other means than their surface position. This leads to the idea of using inherent properties of the arguments as a way to identify them with variables of the head. These properties are, in Hungarian, foremost case, but also person, number, and definiteness. A verb decides not only the basis of position but on the basis of case which constituent is its subject and which one is object etc. The properties thus constitute the *name* of the argument.

The theory by Vermeulen is insufficient in certain respects. In its original form it distinguishes a left incoming name from a right outgoing name; however, the left-right distinction is relegated here to the morphology and does not figure at all in the semantics. However, the notion of incoming and outgoing names is

important. Consider merging two constituents $A$ and $B$. Then one of them, say $A$, will assume the role of the functor, taking the other, $B$, as argument. The variable $x$ of $A$ and $y$ of $B$ are identified if the outgoing name of $y$ in $B$ matches the incoming name of $x$ in $A$. In tandem with names, each variable is associated with a diacritic that states whether or not the variable actually has an incoming and/or outgoing name. For names can be dropped, in which case a variable loses its ability to be identified with other variables in further computation. By default, incoming names and outgoing names are the same; but they need not be. For syntactic purposes we need to distinguish arguments from adjuncts; also, we need to distinguish an ordinary variable from a parameter. All these characteristics are unified into a so-called argument identification statement (AIS). For each syntactic argument such an AIS must be issued. An **argument structure** (AS) is a sequence of AISs. (To stress: it is not necessary to have an AIS for every variable; an AIS is only needed if the variable needs to be manipulated.) It contains the name of the variable, it contains a so-called *diacritic*, specifying whether the variable is imported ($\triangledown$) or exported ($\triangle$), or both ($\Diamond$). Furthermore, if a variable is imported, a name is given under which it is imported; if it is exported, a name is given under which it is exported. Names have the form of an attribute value structures, with usual notion of unification (thus allowing for certain types of abstractness).

A constituent $A$ can be merged with a constituent $B$ only if the semantic merge identifies at least one variable. (Two variables is also possible, for example in control structures.) We note here that the merge of a single variable has consequences on a different set of referents, called *parameters*. Unlike ordinary variables, parameters are identified through their role (eg *reference time*, *event time*, *worlds* and so on). An AIS associates with a variable two sets of parameter statements. These have the form [*role* : *ref*], with *role* a role and *ref* a referent. The first set describes the incoming parameters, the second the outgoing parameters. When $A$ takes $B$ as argument, and $x$ is identified with $y$, then an *incoming* parameter for a role $\rho$ is identified with the *outgoing* parameter for the role $\rho$ of $B$. Parameters not mentioned in the lists are simply passed unchanged. It is possible to reevaluate parameters but also to make them change roles. For example, in Russian the reference time in the subordinate clause of an indirect speech act equals the event time of the main clause, while in English it equals the reference time of the main clause. Thus this mechanism can be used for sequencing context parameters, such as time, person, world and location (sequence-of-world, sequence-of-time, sequence-of-person and so on, as described in [4]).

Thus, a complete entry has three components:

1. an exponent, for example a string (but more complex exponents are implemented, see below);
2. an argument structure (AS). This is a sequence of argument identification statements.
3. a semantics, for example a DRS.

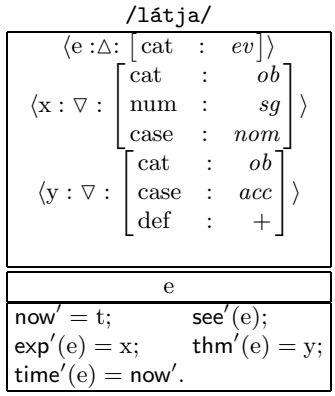
The structure (1) shows all three components, the string on top, the argument structure in the middle, and a DRS at the bottom.

The AS replaces not only the indexation but in fact a lot of the structure building itself. The syntactic categories are encoded mainly in the outgoing names. The complexity of merge is very low. Unification proceeds in $O(m + n)$ time. Thus to compute the semantics of a sentence is very fast ($O(n^3)$ for context free grammars). This is *not* true of our implementation, since the implementation also returns *all* parse terms and evaluates them into meanings. Since structural ambiguities can in worst cases be exponential in the length of the string the implementation runs in exponential time as worst case. (This is mainly due to the fact that the implementation is meant to reflect the theory as accurately as possible.)

## 4 Agreement

Fig. 1 gives an example of an entry. The referent $x$ belongs to an argument ($\triangledown$) which has to be a thing (cat : *ob*), in the nominative (case : *nom*) and singular (num : *sg*). The $e$ referent however has a $\triangle$, which means that it does not belong to an argument. Thus it is indicated that the denotation of the word `látja` is an event (cat : *ev*). The surface orientation of our approach

**Fig. 1.** The argument structure and semantics of the word 'látja'




has the following consequence. Names are needed to identify referents across structures; they must therefore be computable properties of the argument itself. The appearance of incoming names for the arguments is therefore correlated with surface differences in the arguments themselves. We see definiteness figure in the identification statement for the object for the reason that there is a different set of endings for definite objects. The verb flags for its object to be definite. It also flags for it to have accusative case. Here is an example from German.

(8) `Dir schärfsten Kritiker hat die Präsidentin in ihrer Heimat.`
The harshest critics has the president in her home [country].

Both arguments can be both nominative and accusative. However, when the verb shows singular agreement, excluding the first from being subject, since it unequivocally plural.

## 5 The Implementation: Description

The implementation is written in OCaML, a functional programming language. It has both a command line interface and a Tk-interface to allow for interactive sessions. The software is designed to support Unicode and multiple languages. At present, it can be both installed and run in English and German. Documentation is also available in both languages.

The output is sent to a .tex-file, which is translated using LaTeX, and is then shown to the user, but can also be stored independently for later use.

The algorithm proceeds via so-called *entries*, which are records consisting of four fields corresponding to

- the morphology: this is a set of *morphs*;
- the argument structure, which is an array of *argument identification statements*;
- the semantics, which is a DRS;
- the set of parse terms, which show the analysis terms of the entry.

Morphs consist of

1. an exponent, which is an array of string;
2. an array of subcategorisation statements. These consist in turn in a specification for each argument that the entry takes of
   (a) the required morphological class of the argument (possibly also its form)
   (b) the class (and shape) of the element produced when the argument is consumed,
   (c) the way the functor and argument morphology need to be combined (concatenation, reduplication and so on).

Argument identification statements consist of

1. a variable name;
2. a diacritic displaying the way in which the variable must be handled during merge;
3. a syntactic class for the argument to be consumed;
4. a syntactic class of the element constructed if the argument is consumed;
5. a parameter statement, showing the way in which parameters are consumed and passed up.

Notice that morphs need not consist of a single string, they can consist of several strings (thus we can accommodate circumfixes, but also the fact that in Hungarian the verbal prefix can be split from its verbal root). There are however no functions that allow to change any letter, and there may not be any empty morphs.

There is no inbuilt distinction between words and morphemes. The blank is considered a symbol of its own, like punctuation. Given a string as input, the system will match the morphs of the dictionary against any combination of substrings. If morphs consist of at most $k$ units (typically, $k = 2$ is sufficient), then this gives $O(n^{2k})$ many occurrences, where $n$ is the length of the string. The algorithm is a chart, which is implemented as a hash-table over pairs $(\ell, k)$, where $\ell$ is the overall length of the occurrence (the sum over all lengths of the parts), and $k$ is the index of the leftmost occurrence. The chart is constructed by induction over the length and is finished when that length equals the string length. The output is then returned.

The chart parser operates on *occurrences*. These are quadruples, consisting of an argument structure, an occurrence of a morph (an array of pairs of positions), some morphological components (stating how the element may be further combined) and a term. The semantics is absent. It is inserted only when the successful terms are finally evaluated. This allows to keep the burden on the parser small. Currently, it is not very efficient, but it can be made much faster if need be.

The most flexible session is the standalone-session. After compilation, dictionaries can be loaded and unloaded dynamically. A useful tool is the command `diagnose`. Given two entries it documents the calculations in a step-by-step fashion. All outputs can be saved in a file and used for different purposes.

## 6 Documentation and Source

The software was originally designed to allow for the evaluation of the theory of argument structure. The software and the theory are now under simultaneous development. At the time of writing, some developments of the software are not yet reflected in the documentation. The current version of the software is 5.0. Installation currently is possible for Unix platforms only and has been tested on several of them, including MacOS X. Both the software and the manuscript can be freely obtained from

`http://kracht.humnet.ucla.edu/marcus/referent`

subject only to usual open license conditions.

## References

1. Kamp, Hans: A theory of truth and semantic representation, in: Groenendijk, Jeroen (ed.): Formal methods in the study of language, Mathematisch Centrum, (1981).
2. Kamp, Hans, Reyle, Uwe: From Discourse to Logic, Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory, Kluwer, Dordrecht,
3. Kracht, Marcus: Agreement Mophology, Argument Structure and Syntax, Manuscript, UCLA, 2006.

4. Schlenker, Philippe: A Plea for Monsters, Linguistics and Philosophy 26, 29–120, (2003).
5. Vermeulen, Kees F. M., Merging without Mystery or: Variables in Dynamic Semantics, Journal of Philosophical Logic (24), 405–450, (1995).