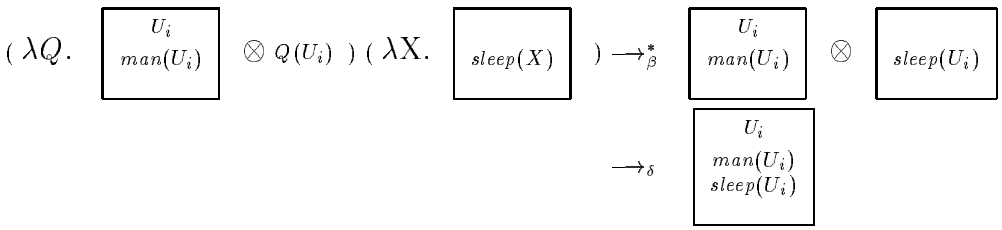
Dynamic Lambda Calculus

Michael Kohlhase, Susanna Kuschert
Universität des Saarlandes

The goal of this paper is to lay a logical foundation for discourse theories by providing an algebraic foundation of compositional formalisms for discourse semantics as an analogon to the simply typed λ-calculus. Just as that can be specialized to type theory by simply providing a special type for truth values and postulating the quantifiers and connectives as constants with fixed semantics, the proposed dynamic λ-calculus $\mathcal{DLC}$ can be specialized to λ-DRT by essentially the same measures, yielding a much more principled and modular treatment of λ-DRT than before; $\mathcal{DLC}$ is also expected to eventually provide a conceptually simple basis for studying higher-order unification for compositional discourse theories.

Over the past few years, there have been a series of attempts [Zee89, GS90, EK95, Mus96, KKP96, Kus96] to combine the Montagovian type theoretic framework [Mon74] with dynamic approaches, such as DRT [Kam81]. The motivation for these developments is to obtain a general logical framework for discourse semantics that combines compositionality and dynamic binding.

Let us look at an example of compositional semantics construction in λ-DRT which is one of the above formalisms [KKP96, Kus96]. By the use of β-reduction we arrive at a first-order DRT representation of the sentence $\texttt{A}^i$ `man sleeps.` (i denoting an index for anaphoric binding.)



where $\otimes$ is the λ-DRT conjunction operator that intuitively merges two DRSes by unifying the sets of discourse referents and that of conditions.

Unfortunately, the above mentioned unified formalisms have failed so far to duplicate a key aspect of type theory that has lead to interesting linguistic analyses in computational linguistics. Type theory (or higher-order logic) is a two-layered formalism, where the algebraic content (the behaviour of higher-order functions) is neatly packaged into a formalism of its own, namely simply typed λ-calculus; while the logical content (the specific semantics of connectives and quantifiers) is built on top of it. Thus the use of type theory allows us to deal with the complexities of natural language semantics on two distinct levels: the simply typed λ-calculus provides the theory of β-reduction (which is the motor of compositionality) whereas the logical side of semantics is dealt with by a system that is rather like predicate logic. By focussing on known mechanisms for dealing with each of the two subsystems, it has proved possible to use type theoretic techniques for natural language processing systems.

- Higher-order unification [Hue75] solves equations in the simply typed λ-calculus and leads to analyses of ellipsis [DSP91], and focus [GK96, Pul94]. Note that these accounts are inadequate for the treatment of the logical structure, so they make insufficient predictions about quantifiers and connectives.

- First-order automated theorem proving [Fit90] is used to reason about the logical structure of natural language, for presuppositions, and to integrate world knowledge into natural language semantics. Note that these approaches normally cannot capture higher-order aspects of the semantics like compositionality or underspecified (e.g. elliptic-) semantic elements.

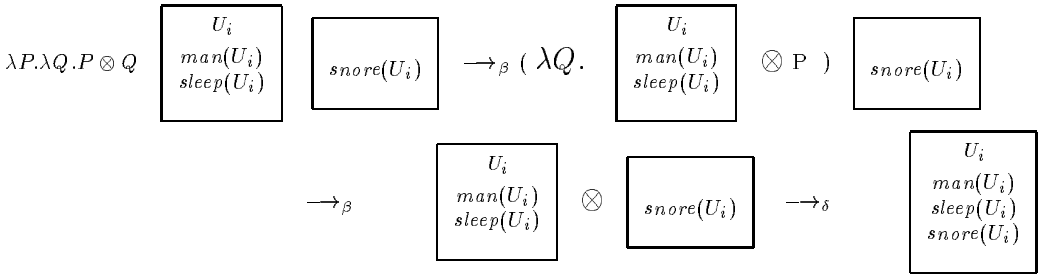
- Only recently, logic formalisms for higher-order theorem proving [Koh95] have appeared that are generalizations of both higher-order unification and automated theorem proving. These can be used to integrate world knowledge into the unification-based approaches [GKvL96].

The goal of this paper is to lay the foundation for analyses like the above by providing an algebraic foundation of compositional formalisms for discourse semantics as an analogon to the

simply typed λ-calculus. Just as that can be specialized to type theory by simply providing a special type o for truth values and postulating the quantifiers and connectives as constants with fixed semantics, the proposed *Dynamic Lambda-Calculus* ($\mathcal{DLC}$) can be specialized to λ-DRT [KKP96] by essentially the same measures, yielding a much more principled and modular treatment of λ-DRT than before.

However, we expect the benefits from a clean separation of the structural and logic parts of compositional discourse semantics will not be restricted to this. In particular, $\mathcal{DLC}$ is expected to serve as the formal basis for the development of higher-order unification algorithms for compositional formalisms for discourse semantics (the topic of a future talk, though), which in turn can be expected to lead to dynamic analyses of ellipses, focus, corrections, ..., corresponding to those discussed above. First experiments with the formal system have shown that these will be more intuitive than those for the static case.

The central theme of DRT is the establishment of anaphoric binding in discourse; λ-DRT, similar to the other above mentioned formalisms, establishes such bindings in a Montagovian-like construction process given coindexation of the pronoun with its antecedent through the syntactical analysis. As an example, consider continuing the above sentence by $\texttt{He}_i$ `snores.` The representation of the discourse may be constructed from the representation of the two sentences by applying them to $\lambda P.\lambda Q.P \otimes Q$ and reducing, arriving at



Note that in this process the free variable in the representation of $\texttt{He}_i$ `snores`, standing for the pronoun $\texttt{he}_i$, has in the course of β-reduction been captured by the declaration of the discourse referent U_i in the representation of the first sentence, which is part of the representation of $\texttt{a}^i$ `man`. Indeed, the capturing of free variables, *the* thing impossible in pure λ-calculus, is the driving force of the establishment of anaphoric binding in λ-DRT.

The proposed formalism $\mathcal{DLC}$ focuses on the interaction of dynamic binding (declaration of discourse referents), function abstraction and function application. It is spelt out by the interaction of two distinct abstraction operators, the well-known λ-, and the dynamic δ-operator. The *capturing of free variables*, motivated above, will be one of the central themes. This leads us to a thorough study of the variables known from λ-calculus and the variables that can thus be captured.

Most of the burden of the interaction of the two kinds of variables, and in particular the respective abstraction mechanisms that go with them, is carried by an elaborate type system that takes into account structural properties of dynamic systems (the *mode*), such as binding power and the accessibility relation. In first experiments, the use of these binding properties have proven very useful in different linguistic applications.

This abstract only gives a first idea about the details of $\mathcal{DLC}$; the full paper can be found via http://coli.uni-sb.de/~kuschert/academic.html.

1 The Syntax of $\mathcal{DLC}$

The key to understanding the character of compositional dynamic formalisms is the identification of functional and dynamic properties and the interaction of these. [KKP96, Kus96] already observe that we can locate two different kinds of variables together with two kinds of abstractions of very different nature: the well-known standard λ-abstraction, used for the compositional construction of representations, and a dynamic abstraction to be called δ-abstraction. Most prominent of their differences is the fact that δ-abstraction may capture free variables on β-reduction which breaks a taboo in standard λ-calculus. In essence this means that the two abstraction operators have quite different notions of scope: Whereas δ-abstraction is boundless with respect to function application,

it is bounded by the notion of accessibility, motivated and defined by some linguistic theory (in particular DRT).

Observing that both these properties are of a structural nature, the central idea of $\mathcal{DLC}$ is to encode the necessary information for variable capture and the accessibility relation in an elaborate type system. Such information must include knowledge about δ-bound variables, which have the power to capture variables, and knowledge about free variables, which are liable to being captured.

As a consequence, $\mathcal{DLC}$'s types include *mode* information for the variables that are visible from the outside of an expression: In addition to the standard λ-calculus types (formed from basic types and functional application) there are types of the form $\Gamma\#\alpha$ for any α, where the mode Γ marks a variable with a '+' if the variable has capturing power, and with a '−' if it is prone to being captured (i.e. if it is a free variable)[1]. As a first example, if the expression $\mathbf{A}$ has type $X^-, U^+, \Gamma\#\alpha$, we know that $\mathbf{A}$ is of standard (type theoretic) type α (e.g. if α is the base type o, then $\mathbf{A}$ is a proposition) and $\mathbf{A}$ contains at least the free variable X and the δ-bound variable U. λ-bound variables in $\mathbf{A}$ do not appear in the mode of its type, since they are not visible to the outside of $\mathbf{A}$.

The set $\mathcal{V}$ of (typed) variables is partitioned into two distinct sets of variables of inherently different character: the set of *functional variables* $\mathcal{V}^{fun}$ and the set of *dynamic variables* $\mathcal{V}^{dyn}$. In a mode, a positive declaration of a variable U will always overwrite a negative declaration, i.e. $U^+, U^- = U^+ = U^-, U^+$. We use Γ^+ (Γ^-) for the *positive* (*negative*) submode of Γ. As an example, if $\Gamma = U^+, Y^-, P^+$, then $\Gamma^+ = U^+, P^+$, and $\Gamma^- = Y^-$.

Members of the set $\mathcal{T}$ of *$\mathcal{DLC}$-types*, or simply *types*, are constructions of the form $\beta = \Gamma\#\alpha$, and *function types* of the form $\alpha \to \beta \in \mathcal{T}$ (# shall bind stronger than $\to$), where $\alpha, \beta \in \mathcal{T}$, built up from a fixed set $\mathcal{BT}$ of *base types*. We call a type *simple*, iff it is a base type or a functional type. We identify a simple type α with type $\emptyset\#\alpha$. Note that simple types may contain modes on functional level. Furthermore, we identify $\Gamma\#(\Delta\#\alpha)$ with $(\Gamma, \Delta)\#\alpha$. Thus we can always rewrite a type β such that α is simple in $\beta = \Gamma\#\alpha$. In this case we call α the *characteristic type* and Γ the *characteristic mode* of β.

We shall need to allow for the arguments of functional expressions to carry free variables, and therefore define extensions $\overline{\alpha}$ and $\overline{\overline{\alpha}}$ of a type α, given some negative modes Γ_i (i.e. $\Gamma_i^+ = \emptyset$): if α is of base type, then both $\overline{\alpha}$ and $\overline{\overline{\alpha}}$ equal α. If $\alpha = \Delta\#(\alpha_1 \to (\ldots \to \alpha_n))$, then $\overline{\alpha} = \Delta\#(\Gamma_1\#\alpha_1 \to (\Gamma_2\#\alpha_2(\ldots \to [\bigcup_{i=1\ldots(n-1)} \Gamma_i]\#\alpha_n)))$ and $\overline{\overline{\alpha}} = \Delta\#(\Gamma_1\#\alpha_1 \to (\Gamma_2\#\alpha_2 \to (\ldots \to \Gamma_n\#\alpha_n))$, where $\Delta \cap \Gamma_i = \emptyset$. Unless stated otherwise, when we speak of α in the context of a given $\overline{\alpha}$ or $\overline{\overline{\alpha}}$, we assume α to be the minimal type, that is, taking away all negative variables on all levels.

We extract the top-level positive (negative) variables of a type α by α^+ (α^-), defined as $\Gamma^+ \cup \beta^+$ ($\Gamma^- \cup \beta^-$) if $\alpha = \Gamma\#\beta$, and $\alpha^+ = \emptyset$ ($\alpha^- = \emptyset$) if α is a simple type. Further, we shall need a function which collects the variables of all levels of a type, the *binding potential of* α, defined as $\mathcal{BP}(\alpha) = \alpha^+ \cup \alpha^-$, if α is not a functional type and $\mathcal{BP}(\alpha) = \mathcal{BP}(\beta) \cup \mathcal{BP}(\gamma)$, if $\alpha = \beta \to \gamma$. We write the positive (negative) parts of α's binding potential as $\mathcal{BP}^+(\alpha)$ ($\mathcal{BP}^-(\alpha)$). The name *binding potential* reflects that $\mathcal{BP}(\alpha)$ gives full information on which variables contribute to the binding behaviour of an expression which is of type α. The binding behaviour needs to contain both positive and negative variables. Further, we need to consider all functional levels of the type, since they give information on what happens upon function application.

Lastly, we need to define a *substitution of modes in a type* $[\Delta/X]\alpha$ by

$$
\begin{array}{lll}
[\Delta/X]\alpha & = [\Delta/X]\Gamma\#[\Delta/X]\alpha' & \text{if } \alpha = \Gamma\#\alpha' \\
& = [\Delta/X]\beta \to [\Delta/X]\gamma' & \text{if } \alpha = \beta \to \gamma \\
& = \alpha & \text{if } \alpha \text{ base and simple} \\
[\Delta/X]\Gamma & = \Delta, (\Gamma - X^-) & \text{if } X^- \in \Gamma \\
& = [\Delta/X]n, [\Delta/X](\Gamma - n) & \text{for some } n \in N \\
& = \Gamma & \text{else} \\
[\Delta/X]n & = [\Delta/X]n & \text{(i.e. substitutions for placeholders are not resolved)}
\end{array}
$$

Defining the well-formed formulae of $\mathcal{DLC}$, we start from a signature Σ of typed constants, consisting of two disjunct subsets Σ^l and Σ^p (for logical constants and parameters, i.e. the non-

[1] Modes will also include natural numbers acting like de-Bruijn indices; however, their motivation is too involved for the extend of this abstract and may safely be ignored for a first understanding of $\mathcal{DLC}$. Still, we will mention them in the definitions for completeness.

logical constants, respectively). For reasons of minimality, we assume that Σ does not contain any constant functions — these may be constructed from atomic expressions by functional application.

We assume that Σ^l contains at least the operators ∂, ∂^s and ∂^o which will be called *dynamification operators*:

$$\begin{array}{ll} \partial & :(\alpha \to \beta \to \gamma) \to (\Gamma \# \alpha) \to (\Delta \# \beta) \to (\Gamma \cup \Delta) \# \gamma \\ \partial^s & :(\alpha \to \beta \to \gamma) \to (\Gamma \# \alpha) \to (\Delta \# \beta) \to (\Gamma \cup \Delta) \# \gamma \text{ where } \Delta^+ \cup \Gamma^- = \emptyset \\ \partial^o & :(\alpha \to \beta) \to (\Gamma \# \alpha) \to \Gamma \# \beta \end{array}$$

Well-formed formulae will be defined by means of an inference system, where the judgment $\mathbf{A}\colon \alpha$ holds, iff $\mathbf{A}$ is a formula of type α.

Definition 1.1 (Well-formed Formulae of $\mathcal{DLC}$). The syntactic category of *well-formed formulae* consists of constants, variables, *applications* ($\mathbf{AB}$), and *λ-abstractions* ($\lambda X_\alpha.\mathbf{A}$), *dynamic abstractions* ($\delta U_\alpha.\mathbf{A}$). The inference system for the judgment schema of *well-typedness with respect to some variable context* $\mathcal{A}$, denoted by $\mathcal{A} \vdash \mathbf{A}\colon \Gamma \# \alpha$, is given by the following schemata[2]:

$$\frac{\Sigma^p(c) = \alpha}{\mathcal{A} \vdash c\colon \overline{\alpha}}\ \mathit{wff{:}par} \qquad \frac{\Sigma^l(c) = \alpha}{\mathcal{A} \vdash c\colon \alpha}\ \mathit{wff{:}lconst} \qquad \frac{A \notin \mathcal{BP}(\overline{\overline{\alpha}})}{\mathcal{A}, [A : \overline{\overline{\alpha}}] \vdash A\colon A^- \# \overline{\overline{\alpha}}}\ \mathit{wff{:}var}$$

$$\frac{\mathcal{A}, [U : \overline{\overline{\beta}}] \vdash \mathbf{A}\colon \alpha}{\mathcal{A}, [U : \overline{\overline{\beta}}] \vdash \delta U_\beta.\mathbf{A}\colon U^+ \# \alpha}\ \mathit{wff{:}dyn}$$

$$\frac{\mathcal{A}, [X : \gamma] \vdash \mathbf{A}\colon \alpha \quad X \notin \mathbf{Dom}(\Gamma^-) \quad X \notin \mathcal{BP}(\gamma) \quad \gamma = \overline{\overline{\beta}}}{[0/X]\mathcal{I}(\mathcal{A}) \vdash (\lambda X_\beta.\mathbf{A})\colon \Gamma^- \# \mathcal{I}(\gamma) \to [\Gamma^-/X]\mathcal{I}(\alpha)}\ \mathit{wff{:}abs}$$

$$\frac{\mathcal{A} \vdash \mathbf{A}\colon \Gamma^- \#(\Delta^- \# \alpha \to \beta) \quad \mathcal{A} \vdash \mathbf{B}\colon \alpha}{\mathcal{D}([\alpha^-/0]\mathcal{A}) \vdash \mathbf{AB}\colon \Gamma^- \# \mathcal{D}([\alpha^-/0]\beta)}\ \mathit{wff{:}app}$$

If $\mathcal{A} \vdash \mathbf{A}\colon \Gamma \# \alpha$ for some $\mathcal{A}$, where α is simple, we shall also write $\mathbf{A} \in \mathit{wff}_\alpha(\Sigma; \Gamma)$. Note that α depends on the choice of $\mathcal{A}$.

Note that the definition of $\mathcal{DLC}$ well-formedness is a simple extension of the well-formedness in λ-calculus, being extended only by the definition of δ-abstraction and the management of modes. Note in particular that a variable is a member of its own mode, and that free variables of a function argument are allowed in through the Γ^- on top level and through the $\overline{\overline{\beta}}$ on functional level. Observe that the top level free variables are free in the result type also exactly if the λ-abstraction is not empty. Apologies for this rather brief exposition come with an example for a $\mathcal{DLC}$ type derivation. $\mathcal{B}, \mathcal{B}'$ and $\mathcal{B}''$ are variable contexts such that $\mathcal{B} = \mathcal{B}' - [U : \ldots] = \mathcal{B}'' - [F : \ldots]$.

$$\frac{\dfrac{\mathcal{A}, [U : \overline{\overline{\theta}}], [P : \overline{\overline{\gamma}}] \vdash P : P^- \# \overline{\overline{\gamma}}}{\mathcal{A}, [U : \overline{\overline{\theta}}], [P : \overline{\overline{\gamma}}] \vdash \delta U.P : U^+, P^- \# \overline{\overline{\gamma}}}}{\mathcal{A}, [U : \overline{\overline{\theta}}] \vdash \lambda P.\delta U.P : (\Gamma^- \# \overline{\overline{\gamma}}) \to [\Gamma^-/P]U^+, P^- \# \overline{\overline{\gamma}}}$$

$$\frac{\mathcal{B}', [F : \Delta'^- \# \alpha \to \Delta''^- \# \beta] \vdash F\colon F^- \#(\Delta'^- \# \alpha \to \Delta''^- \# \beta) \quad \mathcal{B}'', [U : \overline{\overline{\theta}}] \vdash U : U^- \# \overline{\overline{\theta}}}{\mathcal{B}, [U : \alpha], [F : \Delta'^- \# \alpha \to \Delta''^- \# \beta] \vdash F(U) : F^-, \Delta''^- \# \beta}\ \boxed{\begin{array}{l} \Delta'^- \# \alpha = \\ U^- \# \overline{\overline{\theta}} \end{array}}$$

[2] $\mathcal{D}$ and $\mathcal{I}$ are defined as the decrement and increment on natural numbers respectively, i.e. $\mathcal{I}(U^+, Y^-, 1, P^+ \#(\alpha \to \beta) = U^+, Y^-, 2, P^+(\mathcal{I}(\alpha) \to \mathcal{I}(\beta))$, and likewise for $\mathcal{D}()$. For purpose of this abstract and a first understanding of $\mathcal{DLC}$, ignore these as well as the introduction and substitution of numbers in modes.

The latter sub-construction illustrates how both, the type extension (introducing Δ'^- which is matched with U^-) and the negative mode Γ^- of the application rule (being matched with F^-) are used to derive $F(U)$'s type. Now, we derive the type of $\mathbf{A} = \lambda F.(\lambda P.\delta U.P)F(U)$, calling the above subderivations (1) and (2) respectively.

$$\dfrac{\dfrac{(1)\quad(2)}{\mathcal{B}, [U:\alpha], [F:\ldots] \vdash (\lambda P.\delta U.P)F(U)\colon U^+, F^-, \Delta''^-\#\beta}\ \boxed{\begin{array}{l}\Gamma^-\#\overline{\overline{\gamma}} \\ F^-, \Delta''^-\#\beta\end{array} =}}{\mathcal{B}, [U:\alpha] \vdash \lambda F.(\lambda P.\delta U.P)F(U)\colon (\Xi^-\#(U^-\#\alpha \to \Delta''^-\#\beta)) \to U^+, \Xi^-, \Delta''^-\#\beta}$$

Observe that Δ''^- is not yet specified in $\mathbf{A}$'s type. This depends on what kind of function will be substituted for F. E.g. if we apply $\mathbf{A}$ to $\lambda X.s(X) : \Theta^-\#\epsilon \to \Theta^-\#o$ given $[X:\epsilon] \in \mathcal{B}$ and some type o, we get $\Delta''^- = U^-$. Alternatively, if we apply $\mathbf{A}$ to $\lambda X.r : (\Theta^-\#\epsilon) \to o$, then $\Delta''^- = \emptyset$.

In the first of these alternatives, i.e. $\mathbf{A} = (\lambda P.\delta U.P)s(U)$, we may also observe how variable capture is mirrored in the types: here, the Δ''^- in the type of $F(U)$ contains a negative U^- by the specification of F. The positive U^+, which comes from $\lambda P.\delta U.P$, overwrites the U^-. Thus the key to the structural modeling of variable capturing is the overwriting effect of positive variables. Note that capturing works regardless of whether the positive variable occurs in the functor or in the argument.

If the mode of an expression contains no positive variables, we call that expression *static*, and *dynamic* otherwise. In the same spirit, we call $\lambda X.\mathbf{A}$ a *static* or *functional abstraction*, and $\delta U.\mathbf{A}$ a *dynamic abstraction*, since it adds to the degree of dynamicity. Further, for a variable A we distinguish between A being *functionally bound* (bound by a λ-operator) in an expression $\mathbf{A}$, defined as in standard λ-calculus, and A being *dynamically bound* (bound by a δ-operator) in $\mathbf{A}$, if A occurs positively in $\mathbf{A}$'s type. A is *free* in $\mathbf{A} : \alpha$, iff $A \in \alpha^-$, and we write $A \in \mathcal{FV}(\mathbf{A})$.

Substitution (for functional variables) in $\mathcal{DLC}$ is defined very much like its respective notion in λ-calculus, except that capturing of dynamic variables through substitution is perfectly allowed. With $\mathcal{DLC}$-types we may also express λ-calculus substitutability by means of types only: An expression $\mathbf{B}\colon\beta$ is *substitutable* for a functional variable Y in an expression $\mathbf{A}\colon\alpha$, iff $\mathcal{BP}(\alpha) \cap \beta^- = \emptyset$. The definition of substitution must be extended by a clause for dynamic expressions, which is trivial though: since U is not a functional variable, we have $[\mathbf{B}/Y]\mathbf{A} = \delta U.([\mathbf{B}/Y]\mathbf{C})$. We note that $\mathcal{DLC}$ substitution is type-preserving.

In $\mathcal{DLC}$, we may use the well-studied β-reduction of type theory for the semantic construction process, in the way suggested by Montague. In a similar way, we also need a so-called δ-reduction, by which means dynamic expressions are constructed from their dynamic constituents. A particular feature of $\mathcal{DLC}$ is, however, that α-renaming known from type theory is not be confined to λ-bound variables alone but extended to δ-bound variables. In fact, α-renaming of dynamically bound variables is a problem for the existing systems for compositional discourse semantics mentioned in the introduction and $\mathcal{DLC}$ evolved from an attempt to understand full α-conversion. Facilitated by the richer type system, we are able to define a joint α-conversion rule for functionally and dynamically abstracted variables (let A and B be either both functional or dynamic variables); the operator $\mathcal{C}_A^B$ changes B for A everywhere in $\mathbf{A}$ itself, including the abstractions, *and* also in its type.

$$\dfrac{\mathcal{A} \vdash \mathbf{D}\colon\beta \quad A_\gamma \notin \mathcal{BP}(\beta) \quad B_\gamma \notin \mathcal{A}}{(\mathcal{A} \vdash \mathbf{D}) =_\alpha ([B/A]\mathcal{A} \vdash \mathcal{C}_A^B(\mathbf{D}))}$$

The α-rule may be applied if the variable A to be changed in $\mathbf{A}$ does not occur in the binding potential of $\mathbf{A}$, and B is a new variable (not in $\mathcal{A}$). This means that a variable, whether functional or dynamic, can be renamed if its scope is closed off (note that the scope of free variables can be considered boundless), even over functional application, i.e. it does not have an effect on any of $\mathbf{A}$'s arguments. Note that the substitution $[B/A]\mathcal{A}$ is non-empty, if A is a dynamic variable. As usual, we will assume a built-in α-equality, i.e. that expressions are syntactically equal, if they are α-equal.

In addition to the above α-rule and the standard (λ-calculus) β- and η-rules, $\mathcal{DLC}$ also has a δ-rule which defines that $\partial\mathbf{R}(\delta U.\mathbf{A})\mathbf{B} \longrightarrow_\delta \delta U.(\partial\mathbf{RAB})$ and $\partial\mathbf{R}.\mathbf{A}(\delta U.\mathbf{B}) \longrightarrow_\delta \delta U.(\partial\mathbf{RAB})$, and if both $\mathbf{A}$ and $\mathbf{B}$'s types are static, $\partial\mathbf{RAB} \longrightarrow_\delta \mathbf{RAB}$. Similarly for ∂^s and ∂^o.

The induced equality shows that ∂ and ∂^s are dynamification operators for binary relations $\mathbf{R}$, a symmetric and an asymmetric one respectively, and that ∂^o is a dynamification operator for unary relations $\mathbf{R}$. We have chosen ∂, ∂^s and ∂^o as primitives for $\mathcal{DLC}$ instead of λ-DRT's $\otimes$-operator (and the asymmetric equivalents of other theories). Indeed, we are convinced that it will be possible to characterize the dynamic operators of other approaches in relation to known binary relations by means of these dynamification operators.

2 The Semantics of $\mathcal{DLC}$

The main focus on the definition of $\mathcal{DLC}$'s semantics will be the capturing of dynamic variables, as before. Here it means that the interpretation of the variable to be captured must be such that it can be mirrored onto the interpretation of the bound variable as a side-effect of function application. This effect shall be modelled by delaying the interpretation of the variable until after function application. A general technique to do this is the use of intensionalization. The underlying idea of $\mathcal{DLC}$'s semantics is to use this technique implicitely and let the interpretation process itself guard *all* dynamic variables (instead of using explicit $^\wedge$ and $^\vee$-operators). λ-bound variables will be interpreted as usual, by an assignment function φ that is added to the interpretation function.

The basis of interpretation of $\mathcal{DLC}$-expressions will be a dynamic pre-structure, a straightforward extension of the well-known pre-structure of Henkin models of standard λ-calculus by a domain for (potentially) dynamic structures.

Definition 2.1 (Dynamic Pre-Structures). Let $\mathcal{D}_\mathcal{T}$ be a typed collection of sets and $\mathcal{I}\colon \Sigma \longrightarrow \mathcal{D}$ be a typed total function, then we call the triple $\mathcal{A} := (\mathcal{D}, \underline{@}, \mathcal{I})$ a *dynamic pre-structure*, if

1. $\mathcal{D}_{\alpha\to\beta} \subseteq \mathcal{F}(\mathcal{D}_\alpha; \mathcal{D}_\beta)$
2. $\mathcal{D}_{\Gamma\#\alpha} = \mathcal{F}(\mathcal{B}_\Gamma; \mathcal{D}_\alpha)$

where $\mathcal{B}_\Gamma = \bigcup_{\Delta\supseteq\Gamma} \mathcal{F}(\Delta^{dyn}; \mathcal{D})$[3] is the set of *variable assignments* for the mode Γ, also called Γ-*states*.

The function $\mathcal{I}$ must be defined such that[4]

$$\begin{array}{ll} \mathcal{I}(\partial) & = \Lambda\mathcal{R}, \mathcal{A}, \mathcal{B}.\{a^1 \cup b^1 \mapsto \mathcal{R}(a^2, b^2) \mid a \in \mathcal{A}, b \in \mathcal{B}, a^1 \Big|\Big| b^1\} \\ \mathcal{I}(\partial^s) & = \Lambda\mathcal{R}, \mathcal{A}, \mathcal{B}.\{a^1 \cup b^1 \mapsto \mathcal{R}(a^2, b^2) \mid a \in \mathcal{A}, b \in \mathcal{B}, a^1 \Big|\Big| b^1\} \\ \mathcal{I}(\partial^o) & = \Lambda\mathcal{R}, \mathcal{A}.\{a^1 \mapsto \mathcal{R}(a^2) \mid a \in \mathcal{A}\} \end{array}$$

and the constant $\underline{@}\colon (\Gamma\#\alpha \to \beta) \to (\Delta\#\alpha) \to (\Gamma \cup \Delta\#\beta)$ is defined as $\partial@$ where $@$ is the static (standard) application operator.

The collection $\mathcal{D}$ is called the *carrier set* or the *frame of* $\mathcal{A}$, the set $\mathcal{D}_\alpha$ the *universe of type* α, $\underline{@}$ is the *dynamic application operator*, and the function $\mathcal{I}$ is the *interpretation of constants*.

Definition 2.2 (Denotation $\mathcal{I}_\varphi(\mathbf{A})$). Let $\mathcal{A} = (\mathcal{D}, \underline{@}, \mathcal{I})$ be a dynamic pre-structure and φ be a partial assignment function for λ-bound variables. The *denotation* $\mathcal{I}_\varphi(\mathbf{A})$ of a well-formed formula $\mathbf{A} \in \mathit{wff}_\alpha(\Sigma; \Gamma)$ is defined inductively as follows:

1. $\mathcal{I}_\varphi(c) = \mathcal{I}(c)$ for any $c \in \Sigma$
2. $\mathbf{A}$ is a variable: If $\mathbf{A} = X \in \mathcal{V}^{fun}$, then $\mathcal{I}_\varphi(X) = \varphi(X)$,
   if $\mathbf{A} = U \in \mathcal{V}^{dyn}$, then $\mathcal{I}_\varphi(U) = \{t \mapsto t(U) \mid t \in \mathcal{B}_{[U:\alpha]}\}$
3. $\mathcal{I}_\varphi(\delta U.\mathbf{D}) = \mathcal{I}_\varphi(\mathbf{D})\Big|\Big|_U$[5]
4. $\mathcal{I}_\varphi(\lambda X.\mathbf{D}) = \Lambda\mathcal{A}.\mathcal{I}_{\varphi,[\mathcal{A}/X]}(\mathbf{D})$[6]
5. $\mathcal{I}_\varphi(\mathbf{AB}) = \mathcal{I}_\varphi(\mathbf{A})\underline{@}\mathcal{I}_\varphi(\mathbf{B})$

[3] We write Δ^{dyn} to denote the set of dynamic variables in Δ.

[4] Two partial functions f_1 and f_2 agree, $f_1 \Big|\Big| f_2$, if for all $X \in \mathbf{Dom}(f_1) \cap \mathbf{Dom}(f_2)$ we have $f_1(X) = f_2(X)$.

For all $s \in \mathbf{Dom}(\mathcal{I}_\varphi(\mathbf{A}))$ we call $\mathcal{I}_\varphi(\mathbf{A})(s)$ the *static meaning* of $\mathbf{A}$ with respect to state s and assignment φ.

From the definition of the variables as described above, everything else is simple. Since the set of Γ-states has already been built up recursively in the course of the interpretation of some expression $\mathbf{A}$, the δ-abstraction has little to add in the interpretation of $\delta U.\mathbf{A}$. Note that U may not occur in $\mathbf{A}$ itself, thus the denotation of $\mathcal{A}$ may include Γ-states which are not defined on U. In this case, these states have to be eliminated since the mode of $\delta U.\mathbf{A}$ does contain U.

Note that the job of dynamic binding on the semantic side is mainly done by the dynamification operators: they dynamify a static operation by coordinating the states of two dynamic objects and applying the static operator on the static meanings belonging to the respective states. This coordination is facilitated by the agreement condition $a^1\big|\big|b^1$ and the set union $a^1 \cup b^1$.

3 Application to λ-DRT

We introduced $\mathcal{DLC}$ as an algebraic foundation for (existing) compositional formalisms for discourse semantics. Let us demonstrate this for one of such systems, the λ-DRT [KKP96, Kus96]. To arrive at λ-calculus, we only need to fix the set of base types to $\mathcal{BT} = \{e, o\}$ (individuals and truth values), and specify the set Σ^l, the set of logical constants, thus:

symbol	*type*
$\underline{\sqsupset}_\Gamma$	$(\Gamma\#o) \to (\Gamma^-\#o)$
$\underline{\wedge}_{\Gamma\Delta}$	$(\Gamma\#o) \to (\Delta\#o) \to (\Gamma, \Delta\#o)$
$\underline{\vee}_{\Gamma\Delta}$	$(\Gamma\#o) \to (\Delta\#o) \to (\Gamma^-, \Delta^-\#o)$
$\underline{\Rightarrow}_{\Gamma\Delta}$	$(\Gamma\#o) \to (\Delta\#o) \to ((\Gamma^-, (\Delta^-/\Gamma^+)))\#o$

The types of these constants fully reflect DRT's notion of accessibility which is, in effect, a specification of the dynamic behaviour of certain linguistic constructs.

Note that the formalization in $\mathcal{DLC}$ allows a finer analysis of dynamic objects than in [Kus96], since the mode Γ records the exact degree of dynamicity of a DRS — the λ-DRT type t for DRSes is now expressed by the collection of dynamic types of the form $\Gamma\#o$. This points to an important and useful property of $\mathcal{DLC}$, the merging of DRSes and conditions: these are not inherently different, but DRSes are merely dynamic conditions. For one, this means that the awkward distinction between the two has been dropped. Further, this means that the logical constants now are conglomerations of the respective static and dynamic operator, e.g. $\underline{\wedge}$ unites the static $\wedge$ and the $\otimes$-operator used in [Kus96].

As an example, let us have a look on the type of the representation of a sentence similar to the one quoted in the introduction. The representation of `every man` with its type is $\mathcal{B}, [U : e] \vdash \lambda Q.(\delta U.\mathsf{man}(U) \underline{\Rightarrow} Q(U)) : (\Delta^-\#(U^-\#e \to \Gamma''^-\#o)) \to \Delta^-, (\Gamma''^-/U)\#o$, and `sleeps` is represented by $\mathcal{A} \vdash \lambda X.\mathsf{sleep}(X) : \Gamma^-\#e \to \Gamma^-\#o$. Thus, on application of the two, Δ^- will be instantiated by $\emptyset$, since $\Gamma^-\#\alpha \to \Gamma^-\#o$ matches perfectly on $U^-\#e \to \Gamma''^-\#o$), giving $\Gamma^- = U^- = \Gamma''^-$. With these, the result type $\Delta^-, (\Gamma''^-/U)\#o$ gives $\emptyset\#o$. This is as one would expect since $(\delta U.\mathsf{man}(U)) \underline{\Rightarrow} \mathsf{sleep}(U))$ has no dynamic potential.

The step from $\mathcal{DLC}$'s semantics to a semantics for λ-DRT again is simple. We fix the carrier sets for the basic types to the standard carrier sets for expressions of types e and o, the universe of individuals and the set of truth values respectively. The semantics of the logical constants is summarized below; for $\underline{\wedge}$ we have a simple dynamification of the static $\wedge$, whereas the interpretation of the other constants needs to take into account some notion of quantification.

[5] We define $f\big|\big|_U$ to be the restriction of f to those elements that contain U in their domain, i.e. $f\big|\big|_U = \{s \mapsto f(s) \mid U \in \mathbf{Dom}(s)\}$.

[6] We use Λ for lambda-abstraction in the meta-language with the intuitive meaning.

$$
\begin{array}{lcl}
\mathcal{I}_\varphi(\underline{\neg}_\Gamma)\underline{@}\mathcal{A} & = & \{t \mapsto r \mid t \in \mathcal{B}_{\Gamma^-}, r = \mathrm{T}, \text{if for all } a^1 \mapsto a^2 \in \mathcal{A} \text{ such that} \\
 & & \quad a^1|_{-\Gamma^+} = t \text{ we have } a^2 = \mathrm{F}, \text{ else } r = \mathrm{F}\} \\
\mathcal{I}_\varphi(\underline{\wedge}_{\Gamma\Delta})\underline{@}\mathcal{A}\underline{@}\mathcal{B} & = & \partial\underline{@} \wedge \underline{@}\mathcal{A}\underline{@}\mathcal{B} \\
\mathcal{I}_\varphi(\underline{\vee}_{\Gamma\Delta})\underline{@}\mathcal{A}\underline{@}\mathcal{B} & = & \{a^1|_{\Gamma^-} \cup b^1|_{\Delta^-} \mapsto a^2 \vee b^2 \mid a^1 \mapsto a^2 \in \mathcal{A}, b^1 \mapsto b^2 \in \mathcal{B}, a^1 \big|\big| b^1\} \\
\mathcal{I}_\varphi(\underline{\Rightarrow}_{\Gamma\Delta})\underline{@}\mathcal{A}\underline{@}\mathcal{B} & = & \{t \mapsto r \mid t \in \mathcal{B}_{\Gamma^-,(\Delta^-/\Gamma^+)}, r = \mathrm{T} \text{ if for all } a^1 \mapsto \mathrm{T} \in \mathcal{A} \\
 & & \quad \text{such that } \mathbf{Dom}(a^1|_{-\Gamma^+}) \subseteq \mathbf{Dom}(t) \text{ there} \\
 & & \quad \text{exists a } b^1 \mapsto \mathrm{T} \in \mathcal{B} \text{ such that } a^1 \big|\big| b^1|_{-\Delta^+} \text{ and} \\
 & & \quad a^1|_{-\Gamma^+} \cup b^1|_{-\Delta^+} = t, \text{ else } r = \mathrm{F}\}
\end{array}
$$

4 Conclusion

We are convinced that $\mathcal{DLC}$ with its new typing system constitutes a powerful algebraic basis for a whole class of logical systems combining functional and dynamic logics and that its further study may reveal more of the properties of the interplay of their features. In particular, we hope that in the same way as the types guide the higher order unification of standard λ-calculus, this type system may be useful for dynamic higher order unification.

First experiments have been done to use $\mathcal{DLC}$ instead of the static λ-calculus for applications of natural language understanding. It turned out that the type information does indeed improve the processing power. Further work to be done abounds.

References

[DSP91] Mary Dalrymple, Stuart Shieber, and Fernando Pereira. Ellipsis and higher-order unification. *Linguistics and Philosophy*, 14:399–452, 1991.

[EK95] Jan van Eijck and Hans Kamp. Representing discourse in context. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*. Elsevier Science B.V., 1995.

[Fit90] Melvin Fitting. *First-Order Logic and Automated Theorem Proving*. Springer Verlag, 1990.

[GK96] Claire Gardent and Michael Kohlhase. Focus and higher–order unification. In *Proceedings of the 16th International Conference on Computational Linguistics*. Copenhagen, 1996.

[GKvL96] Claire Gardent, Michael Kohlhase, and Noor van Leusen. Corrections and Higher-Order Unification. In *Proceedings of KONVENS96*, pages 268–279. De Gruyter, Bielefeld, Germany, 1996.

[GS90] J. Groenendijk and M. Stokhof. Dynamic Montague Grammar. In L. Kálmán and L. Pólos, editors, *Papers from the Second Symposium on Logic and Language*, pages 3 – 48. Budapest, Akadémiai Kiadó, 1990.

[Hue75] Gérard P. Huet. An unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.

[Kam81] H. Kamp. A theory of truth and semantic representation. In J. Groenendijk, Th. Janssen, and M. Stokhof, editors, *Formal Methods in the Study of Language*, pages 277 – 322. Mathematisch Centrum Tracts, Amsterdam, 1981.

[KKP96] Michael Kohlhase, Susanna Kuschert, and Manfred Pinkal. A type-theoretic semantics for λ-DRT. In P. Dekker and M. Stokhof, editors, *Proceedings of the 10th Amsterdam Colloquium*, pages 479–498. ILLC, Amsterdam, 1996.

[Koh95] Michael Kohlhase. Higher-order tableaux. In *Proceedings of the Tableau Workshop*, pages 294–309, Koblenz, Germany, 1995.

[Kus96] Susanna Kuschert. *Higher Order Dynamics: Relating operational and denotational semantics for λ-DRT*. CLAUS-Report 84, Universität des Saarlandes, 1996.

[Mon74] R. Montague. The proper treatment of quantification in ordinary English. In R. Montague, editor, *Formal Philosophy. Selected Papers*. Yale University Press, New Haven, 1974.

[Mus96] R. Muskens. Combining Montague semantics and discourse representation. *Linguistics and Philosophy*, 14:143 – 186, 1996.

[Pul94] Stephen G. Pulman. Higher order unification and the interpretation of focus. Technical Report CRC-049, SRI Cambridge, UK, 1994.

[Zee89] H. Zeevat. A compositional approach to DRT. *Linguistics and Philosophy*, 12:95–131, 1989.