Type Reconstruction for λ-DRT Applied to Pronoun Resolution

Hans Leiß(✉) and Shuqian Wu

Centrum Für Informations- und Sprachverarbeitung,
Ludwig-Maximilians-Universität München, München, Germany
leiss@cis.uni-muenchen.de

Abstract. λ-DRT is a typed theory combining simply typed λ-calculus with discourse representation theory, used for modelling the semantics of natural language. With the aim of type-checking natural language texts in the same vein as is familiar from type-checking programs, we propose *untyped* λ-DRT with automatic type reconstruction. We show a principal types theorem for λ-DRT and how type reconstruction can be used to make pronoun resolution type-correct, i.e. the inferred types of a pronoun occurrence and its antecedent noun phrase have to be compatible, thereby reducing the number of possible antecedents.

Keywords: Pronoun resolution · Discourse representation theory · λ-DRT · Type reconstruction · Principal types

1 Introduction

In order to give a compositional semantics for discourse, [2] have extended the non-compositional and first-order approach of Discourse Representation Theory (DRT, [10]) by adding λ-abstraction and functional application. As is familiar from Montague-semantics, the meaning of an expression can then be defined bottom-up, by abstracting from the meaning contribution of the context; function application is then used to combine this meaning with those of expressions from the context.

While DRT uses *discourse representation structures*, i.e. pairs of variables and quantifier-free formulas, and avoids higher-order logic on its way to translate natural language to first-order logic, Montague-grammar and λ-DRT make heavy use of higher order types and are commonly expressed in a simply typed language.

Our first goal is to have a type-free notation of λ-DRS-terms, such that meanings can be written without types, but checked for typeability by "reconstructing" suitable types from types of built-in constants (polymorphic function words and monomorphic content words in the lexicon) and the context of occurrence. For this, we will show that most general types exist and can be inferred automatically. The second goal is to integrate the type reconstruction into a program for pronoun resolution. We want to be able to type-check when a pronoun resolution (i.e. the unification of the discourse variable of a pronoun with the discourse referent of an antecedent) is type-correct, and moreover, we want to use the type reconstruction for unresolved pronouns to filter possible antecedents by their types and the type of the unresolved pronoun.

2 λ-DRT

Where [18] uses meanings like *a* $\mapsto \lambda P \lambda Q \exists x (P\,x \wedge Q\,x)$, *man* $\mapsto \lambda x.man(x)$ and *walks* $\mapsto \lambda x.walk(x)$ and combines these by application to *a man walks* $\mapsto \exists x(man(x) \wedge walk(x))$, in λ-DRT of [2], one uses somewhat different lexical entries

$$\lambda P \lambda Q (\begin{array}{|c|}\hline x \\ \hline \\ \hline \end{array} \otimes P\,x \otimes Q\,x),\ \ \lambda x \begin{array}{|c|}\hline \\ \hline man(x) \\ \hline \end{array},\ \ \lambda x \begin{array}{|c|}\hline \\ \hline walk(x) \\ \hline \end{array}$$

and an operation $\otimes$ of *merging* discourse representation structures as in

$$\begin{array}{|c|}\hline x \\ \hline \\ \hline \end{array} \otimes \begin{array}{|c|}\hline \\ \hline man(x) \\ \hline \end{array} \otimes \begin{array}{|c|}\hline \\ \hline walk(x) \\ \hline \end{array} = \begin{array}{|l|}\hline x \\ \hline man(x),\ walk(x) \\ \hline \end{array}.$$

In general, two discourse structures are merged by appending their (disjoint) lists of discourse referents (variables) and formulas, respectively:

$$\begin{array}{|c|}\hline x_1, \dots, x_m \\ \hline \varphi_1, \dots, \varphi_k \\ \hline \end{array} \otimes \begin{array}{|c|}\hline y_1, \dots, y_n \\ \hline \psi_1, \dots, \psi_p \\ \hline \end{array} = \begin{array}{|c|}\hline x_1, \dots, y_1, \dots \\ \hline \varphi_1, \dots, \psi_1, \dots \\ \hline \end{array}$$

Since a variable in the referent list is seen as a binding, a binder of each merge-factor can bind free variable occurrences in the formulas of *both* merge-factors. In a discourse *A man walks. He talks.*, the meanings of the sentences have to be combined. The pronoun *he* in the second sentence introduces a new discourse referent y with the appropriate property. The combination of the meanings of the sentences is the merging

$$\begin{array}{|l|}\hline x \\ \hline man(x),\ walk(x) \\ \hline \end{array} \otimes \begin{array}{|l|}\hline y \\ \hline talk(y) \\ \hline \end{array}$$

of their discourse structures, followed by pronoun resolution: the referent y of the anaphoric pronoun is *resolved* against some previously introduced discourse referent, here x. This can be implemented by adding an equational constraint $x = y$ to the merged DRS, or by unifying the variables.

If one assumes some co-indexing of pronouns and antecedent noun phrases as a result of syntactic analysis, one can use the referent of the antecedent noun phrase as referent of the anaphoric pronoun. Then, the binding is *dynamic*, i.e. the scope extends beyond sentence boundaries as the discourse goes on:

$$\begin{array}{|l|}\hline x \\ \hline man(x),\ walk(x) \\ \hline \end{array} \otimes \begin{array}{|l|}\hline \\ \hline talk(x) \\ \hline \end{array}.$$

With type reconstruction for λ-DRT, one could just check the type-soundness of pronoun resolution, i.e. that the semantic type of the pronoun occurrence fits the semantic type of the referent of its antecedent. However, we want to use type reconstruction to help pronoun resolution. To do so, we mark discourse referents as anaphors or possible antecedents, use type reconstruction for λ-DRT to infer types for the discourse referents, and then do pronoun resolution with typed referents. Our typing rules for DRSs and DRT's accessibility relation are closely related.

2.1 Untyped λ-DRS-Terms

We use four kinds of raw expressions: terms, formulas, discourse representation structures, and discourses:

$$\begin{array}{lll} \textbf{Term:} & s,t := x & (x \in Var) \\ & \quad\;\; \mid\; c & (c \in Const) \\ \lambda\textbf{-DRS:} & D := x & (x \in Var) \\ & \quad\;\; \mid\; \lambda x D & \\ & \quad\;\; \mid\; (D_1 \cdot D_2) & \\ & \quad\;\; \mid\; \langle [], \varphi \rangle & \\ & \quad\;\; \mid\; \langle x, D \rangle & \\ & \quad\;\; \mid\; (D_1 \otimes D_2) & \end{array}$$

$$\begin{array}{ll} \textbf{Formula:} & \varphi, \psi := \top \\ & \qquad\;\; \mid\; R(t_1, \ldots, t_n) \\ & \qquad\;\; \mid\; t_1 \doteq t_2 \\ & \qquad\;\; \mid\; (\varphi \wedge \psi) \\ & \qquad\;\; \mid\; \neg D \\ & \qquad\;\; \mid\; (D_1 \Rightarrow D_2) \\ & \qquad\;\; \mid\; (D_1 \vee D_2) \\ \textbf{Discourse:} & \mathcal{D} := \epsilon \\ & \qquad\;\; \mid\; \mathcal{D} ; D \end{array}$$

All terms are atomic. Formulas are built from atomic formulas by conjunction of formulas and (non-conjunctive) Boolean combinations of λ-DRSs.

A *box* or *value*-DRS D is a pair $\langle [x_1, \ldots, x_n], \varphi \rangle$ of a list $[x_1, \ldots, x_n]$ of variables and a formula φ, recursively defined by

$$\langle [x_1, x_2, \ldots, x_n], \varphi \rangle := \begin{cases} \langle [], \varphi \rangle, & n = 0, \\ \langle x_1, \langle [x_2, \ldots, x_n], \varphi \rangle \rangle, & \text{else.} \end{cases}$$

Two DRSs D_1 and D_2 may be *merged* to a DRS $(D_1 \otimes D_2)$. So far, the merge-operator $\otimes$ is just a constructor. We will later add reduction rules which provide the intended meaning of the merge of two value-DRSs (with disjoint variable lists) as

$$\langle [x_1, \ldots, x_n], \varphi \rangle \otimes \langle [y_1, \ldots, y_m], \psi \rangle \;\; \rightarrow^* \;\; \langle [x_1, \ldots, x_n, y_1, \ldots, y_m], (\varphi \wedge \psi) \rangle.$$

Finally, we want to have abstraction and application of λ-DRSs. *Note*: We use the pair notation $\langle s, t \rangle$ not for arbitrary terms s, t. Likewise for the types $\sigma \times \tau$: the intention is that σ is an individual type, τ a DRS-type.

The *toplevel referents* and the *free variables* of D are defined by

$$\begin{aligned} top(x) &= \emptyset \\ top(\lambda x D) &= \emptyset \\ top(D_1 \cdot D_2) &= \emptyset \\ top(\langle [], \varphi\rangle) &= \emptyset \\ top(\langle x, D\rangle) &= \{x\} \cup top(D) \\ top(D_1 \otimes D_2) &= top(D_1) \cup top(D_2) \end{aligned} \qquad \begin{aligned} free(x) &= \{x\} \\ free(\lambda x D) &= free(D) \setminus \{x\} \\ free(D_1 \cdot D_2) &= free(D_1) \cup free(D_2) \\ free(\langle [], \varphi\rangle) &= free(\varphi) \\ free(\langle x, D\rangle) &= free(D) \setminus \{x\} \\ free(D_1 \otimes D_2) &= (free(D_1) \cup free(D_2)) \\ &\quad \setminus top(D_1 \otimes D_2) \end{aligned}$$

For formulas built from DRSs, we put

$$\begin{aligned} free(\neg D) &= free(D) \\ free((D_1 \Rightarrow D_2)) &= free((D_1 \vee D_2)) \\ &= free(D_1) \cup (free(D_2) \setminus top(D_1)) \end{aligned}$$

This is motivated by considering free variables of D_2 (representing pronouns) as bound by toplevel referents of D_1 (their antecedents). However, these notions are not stable under β-reduction $\rightarrow$: for example, for $D_1 = \lambda y \langle [x], \varphi\rangle \cdot y$ and $D_1' = \langle [x], \varphi\rangle$ we have $D_1 \rightarrow D_1'$, but $top(D_1) = \emptyset \neq top(D_1')$, and so $(D_1 \Rightarrow D_2)$ may bind less variables of D_2 than $(D_1' \Rightarrow D_2)$. Hence these definitions make sense for expressions in β-normal form only.[1]

In Sect. 5 we define the meaning of application $\cdot$ by β-reduction, i.e. by reducing an application $(t \cdot s)$ to the substitution $t[x/s]$ of free occurrences of x in t by s. Some care is needed to avoid variable capture.

We treat toplevel referents of a merge-factor as binders with scope over all factors. Hence, when substituting a free occurrence of x in $(D_1 \otimes D_2)$ by s, we have to α-rename the top-level referents of D_1 and D_2 to avoid capturing free variables of s. But we also have to rename toplevel referents of s when applying $[x/s]$ to $(D_1 \otimes D_2)$, since s might become a merge-factor, as for $D_1 = x$, and then its toplevel referents would capture free variables of D_2. Since D_1, D_2, s might have toplevel referents after some reductions, we define $t[x/s]$ in such a way that *all* bound variables and referents of t and s are renamed to fresh ones before the free occurrences of x are replaced.[2]

[1] In Sect. 6.2, the DRSs are computed bottom-up along the syntax tree, and at each syntactic construction, the DRS resulting from a combination of the constituents' DRSs is reduced.

[2] Our implementation actually does the renaming only when applications are involved, so $\lambda P((P \cdot x) \otimes (P \cdot y)) \cdot \lambda z D$ copies $\lambda z D$ to get $(\lambda z D \cdot x) \otimes (\lambda z D \cdot y)$ and then renames referents in D when treating the applications as $(D[z/x] \otimes D[z/x])$. Thus, merge-factors have disjoint reference lists, provided the lexical entries have.

An essential clause in the definition of $D[x/s]$ is:

$$(D_1 \otimes D_2)[x/s] = (D'_1[x/s'] \otimes D'_2[x/s'']),$$

where D'_i is D_i with $\mathit{top}(D_1 \otimes D_2) \cap \mathit{free}(s)$ renamed, and s', s'' are s with $\mathit{bound}(s)$ renamed. Similar clauses are needed to treat $(D_1 \Rightarrow D_2)[x/s]$ and $(D_1 \vee D_2)[x/s]$. For example, if P is not in φ, then in

$$\begin{aligned}((\langle [x], \varphi\rangle \otimes (P \cdot x))[P/\lambda x D] &= (\langle [x'], \varphi[x/x']\rangle \otimes (P \cdot x')[P/\lambda x D']) \\ &= (\langle [x'], \varphi[x/x']\rangle \otimes D'[x/x']),\end{aligned}$$

D' is D with toplevel referents renamed and hence does not bind free variables of φ.

3 Typing Rules

Montague-semantics and λ-DRT usually come with base types e for entities and t for truth values. As boxes are pairs $\langle \boldsymbol{x}, \varphi\rangle$ of a list of individual variables and a formula, it seems natural to give them the pair type $e^* \times \mathbf{t}$, where e^* is the type of lists of entities. Instead, all boxes have another base type in [2], and the type $s \to (s \to t)$ of binary relations between situations s (resp. assignments of entities to discourse referents) in [19]).

For the kind of semantic checking of texts we want to do, a more fine-grained typing of DRSs is needed. One should distinguish between entities of different kinds, i.e. replace the base type e by a family $\langle e_i\rangle_{i\in I}$ of base types or *sorts*. The type of a box $\langle \boldsymbol{x}, \varphi\rangle$ then becomes a pair $\boldsymbol{e} \times \mathbf{t}$, so that, essentially, a typed DRS $\langle \boldsymbol{x}, \varphi\rangle : \boldsymbol{e} \times \mathbf{t}$ is a pair of a *type environment* $\boldsymbol{x} : \boldsymbol{e}$ and a formula $\varphi : \mathbf{t}$.

The type $\boldsymbol{e} \times \mathbf{t}$ of a merge-DRS $D_1 \otimes D_2$ then ought to be related to the types $\boldsymbol{e}_1 \times \mathbf{t}$ and $\boldsymbol{e}_2 \times \mathbf{t}$ of the constituents D_1 and D_2 in that $\boldsymbol{e}$ is obtained by *appending* $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$, so $\boldsymbol{e} = \mathit{append}(\boldsymbol{e}_1, \boldsymbol{e}_2)$. However, since $\otimes$ is just a DRS-constructor, we will likewise introduce a type constructor $\otimes$ and use a constraint $\boldsymbol{e} = \boldsymbol{e}_1 \otimes \boldsymbol{e}_2$ in the type reconstruction process. Since the length of referent- and type-lists have to match –even if we had only a single sort of entities–, we cannot use the list type constructor *, but build type lists by *cons*ing a type e_i to a list $\boldsymbol{e}$ of types, $e_i \times \boldsymbol{e}$, beginning with the type $\mathbb{1}$ for the empty list paired with a truth value.

Types:

$$\begin{array}{llll} \sigma, \tau := \alpha & \text{(type variables)} & & \\ \quad\mid\ e_i & \text{(atomic types of individuals)} & \mid (\sigma \times \tau) & \text{(DRSs with non-empty ref-list)} \\ \quad\mid\ \mathbf{t} & \text{(truth values)} & \mid (\sigma \otimes \tau) & \text{(merge-DRSs)} \\ \quad\mid\ \mathbb{1} & \text{(DRSs with empty ref-list)} & \mid (\sigma \to \tau) & \text{(functions)} \end{array}$$

We call a type a *drs-type*, if it is of the forms α, $\mathbb{1}$, $e_i \times \tau$, or $\sigma \otimes \tau$ with drs-types σ and τ. We write $\sigma \times \tau \times \mathbb{1}$ for $(\sigma \times (\tau \times \mathbb{1}))$ and $[\sigma_1, \dots, \sigma_n]$ for $\sigma_1 \times \dots \times \sigma_n \times \mathbb{1}$.

Typing rules:

Typing variables (and constants), abstractions and applications

$$\frac{}{x : \sigma, \Gamma \vdash x : \sigma} \; (var_1)$$

$$\frac{x \not\equiv y \qquad \Gamma \vdash x : \sigma}{y : \tau, \Gamma \vdash x : \sigma} \; (var_2)$$

$$\frac{x : \rho, \Gamma \vdash t : \tau}{\Gamma \vdash \lambda x t : (\rho \to \tau)} \; (abs)$$

$$\frac{\Gamma \vdash t : \sigma \to \tau \qquad \Gamma \vdash s : \sigma}{\Gamma \vdash (t \cdot s) : \tau} \; (app)$$

Using a typed DRS as a type context

$$\frac{\Gamma \vdash x : \sigma}{\langle [], \varphi \rangle : \mathbb{1}, \Gamma \vdash x : \sigma} \; (var_3)$$

$$\frac{y : \rho, \; D : \tau, \; \Gamma \vdash x : \sigma}{\langle y, D \rangle : \rho \times \tau, \; \Gamma \vdash x : \sigma} \; (var_4)$$

$$\frac{D : \rho, E : \sigma, \Gamma \vdash x : \tau}{(D \otimes E) : (\rho \otimes \sigma), \Gamma \vdash x : \tau} \; (var_5)$$

$$\frac{\Gamma \vdash x : \tau}{(D_1 \cdot D_2) : \sigma, \Gamma \vdash x : \tau} \; (var_6)$$

$$\frac{\Gamma \vdash x : \tau}{\lambda y D : \sigma, \Gamma \vdash x : \tau} \; (var_7)$$

An assumption $D : \sigma$ can only be used when D is a variable, a value-DRS, or a merged DRS. The rules (var_3) and (var_4) amount to a typing rule

$$\frac{x_1 : \sigma_1, \ldots, x_n : \sigma_n, \Gamma \vdash x : \sigma}{\langle [x_1, \ldots, x_n], \varphi \rangle : [\sigma_1, \ldots, \sigma_n], \Gamma \vdash x : \sigma} \; (var^+)$$

which says that a typed DRS as assumption is used as a list of typing assumptions of its top-level discourse referents. By (var_5), assuming a typed merged DRS amounts to assuming suitably typed merge-factors. By (var_6) and (var_7), assumptions for typed applications and abstractions can be ignored.

We need typed DRSs as assumptions to type merge-DRSs, disjunctions, implications and discourses (rules $(\otimes)$, $(impl)$, $(disj)$, $(\,;\,)$), where part of the DRS to be typed contains top-level referents whose types have to be assumed to type the rest of the DRS.

Typing value DRSs and merged DRSs

$$\frac{\Gamma \vdash \varphi : \mathbf{t}}{\Gamma \vdash \langle [], \varphi \rangle : \mathbb{1}} \; (drs_1^+) \qquad \frac{x : \sigma, \Gamma \vdash D : \tau}{\Gamma \vdash \langle x, D \rangle : (\sigma \times \tau)} \; (drs_2^+)$$

$$\frac{D_2 : \tau_2, \Gamma \vdash D_1 : \tau_1 \qquad D_1 : \tau_1, \Gamma \vdash D_2 : \tau_2}{\Gamma \vdash (D_1 \otimes D_2) : (\tau_1 \otimes \tau_2)} \; (\otimes)$$

Notice that in (drs_2^+) a variable is removed from the context and built into a DRS. Hence, $\langle x, D \rangle$ corresponds to a binding operator, written $\delta x.D$ in Kohlhase e.a. [13] But in $(\otimes)$ a typed DRS is used like a type context to type another DRS, whereby the scope of $\langle x, D \rangle : \sigma$ is extended to terms outside of D. This is what Kohlhase e.a. [13] call "dynamic" binding of variables in D_2 by binding operators of D_1.

Typing formulas

$$\frac{\Gamma \vdash t_1 : \tau_1 \quad \dots \quad \Gamma \vdash t_n : \tau_n \qquad \Gamma \vdash R : \tau_1 \to (\dots \to (\tau_n \to \mathbf{t}) \dots)}{\Gamma \vdash R(t_1, \dots, t_n) : \mathbf{t}}\,(rel) \qquad \frac{\Gamma \vdash D : \sigma}{\Gamma \vdash \neg D : \mathbf{t}}\,(neg)$$

$$\frac{\Gamma \vdash t_1 : e \qquad \Gamma \vdash t_2 : e}{\Gamma \vdash t_1 \doteq t_2 : \mathbf{t}}\,(eqn) \qquad \frac{\Gamma \vdash D_1 : \sigma_1 \qquad D_1 : \sigma_1, \Gamma \vdash D_2 : \sigma_2}{\Gamma \vdash (D_1 \vee D_2) : \mathbf{t}}\,(disj)$$

$$\frac{\Gamma \vdash \varphi : \mathbf{t} \qquad \Gamma \vdash \psi : \mathbf{t}}{\Gamma \vdash (\varphi \wedge \psi) : \mathbf{t}}\,(conj) \qquad \frac{\Gamma \vdash D_1 : \sigma_1 \qquad D_1 : \sigma_1, \Gamma \vdash D_2 : \sigma_2}{\Gamma \vdash (D_1 \Rightarrow D_2) : \mathbf{t}}\,(impl)$$

Discourses are sequences of sentences; to type the sequence of their DRSs, each DRS is typed in the context extended by the previous DRSs. (Thereby we can resolve pronouns anaphoric ally, to referents in the left textual context.)

Typing discourses

$$\frac{\Gamma \vdash D_1 : \tau_1 \qquad D_1 : \tau_1, \Gamma \vdash D_2 : \tau_2 \qquad \cdots \quad D_n : \tau_n, \dots, D_1 : \tau_1, \Gamma \vdash D_{n+1} : \tau_{n+1}}{\Gamma \vdash (D_1 \,;\, D_2 \,;\, \dots \,;\, D_{n+1}) : ((\dots(\tau_1 \otimes \tau_2)\dots) \otimes \tau_{n+1})}\,(\,;\,)$$

In typing a term, a typed assumption $D : \sigma$ can only be used by decomposing it to the typed top-level discourse referents of D, using (var_3) to (var_5). This cannot be done if D is a variable, application, or abstraction. We ignore assumed typed abstractions by (var_7), which is harmless since they cannot evaluate to boxes, but (var_6), ignoring assumed typed applications, is not: they may reduce to a box containing x as a top-level discourse referent and thus block an assumption $x : \tau$ in Γ. We need to restrict (var_6) to have a form of subject-reduction, see Sect. 5.

By induction on the structure of terms, formulas and λ-DRSs t, we obtain:

Lemma 1. *Suppose for all $x \in free(t)$ and all types σ, $\Gamma \vdash x : \sigma$ iff $\Delta \vdash x : \sigma$. Then $\Gamma \vdash t : \tau$ iff $\Delta \vdash t : \tau$.*

Corollary 1. *1. If $\Gamma, \langle [], \varphi \rangle : \mathbb{1}, \Delta \vdash s : \sigma$, then $\Gamma, \Delta \vdash s : \sigma$.*
2. If $x : \rho, E : \tau, \Gamma \vdash s : \sigma$ and x is not a top-level referent of E, then $E : \tau, x : \rho, \Gamma \vdash s : \sigma$.

4 Type Reconstruction

We want to extend Hindley's well-known "principal types"-theorem from (simply typed) λ-calculus to λ-DRT. The theorem says that the set of typings $\Gamma \vdash t : \tau$ of a term t is the set of instances $S\Gamma_0 \vdash t : S\tau_0$ of a single typing $\Gamma_0 \vdash t : \tau_0$, where $S : \mathit{TyVar} \to \mathit{Ty}$ are the assignments of types to type variables. Then $\Gamma_0 \vdash t : \tau_0$ is a *most general* or *principal typing* of t. A (principal) typing of t *modulo* Γ is a (principal) typing $S\Gamma \vdash t : \sigma$ for some type substitution S and type σ.

It is not hard to see that instances of a DRS-typing are also typings of the DRS.

Lemma 2. *If $\Gamma \vdash D : \sigma$ and $S : \mathit{TyVar} \to \mathit{Ty}$, then $S\Gamma \vdash D : S\sigma$.*

More work is needed to show the existence of principal typings.

Theorem 1. *There is an algorithm W that, given a type context Γ and a term t, either returns a pair (U, τ) of a type substitution $U : \mathit{TyVar} \to \mathit{Ty}$ and a type τ such that $U\Gamma \vdash t : \tau$ is a most general typing of t modulo Γ, or returns 'fail', if there is no (U, τ) such that $U\Gamma \vdash t : \tau$.*

The algorithm W has an easy modification which, on input (Γ, e) where e has a type in some instance of Γ, not only delivers (U, τ) such that $U\Gamma \vdash e : \tau$, but also a variant e' of e where variable bindings are annotated with types.

Proof. The proof is an extension of the proof of [6, 9]. We only consider the cases of variables and terms that are new in λ-DRT over the λ-calculus. Define W as follows:

$$W(\Gamma, x) = \begin{cases} (Id, \tau), & \Gamma = x : \tau, \Gamma' \text{ for some } \Gamma', \\ W((D : \sigma, E : \tau, \Gamma'), x), & \Gamma = (D \otimes E) : (\sigma \otimes \tau), \Gamma', \\ W((z : \sigma, D : \tau, \Gamma'), x) & \Gamma = \langle z, D \rangle : \sigma \times \tau, \Gamma', \\ W(\Gamma', x), & \Gamma = s : \sigma, \Gamma', \text{else}, \\ \mathit{fail}, & \text{else} \end{cases}$$

$$W(\Gamma, \langle x, D \rangle) = S\alpha \times S\tau, \quad \text{if } W((x : \alpha, \Gamma), D) = (S, \tau) \text{ for fresh } \mathit{TyVar}\ \alpha$$

$$W(\Gamma, (D_1 \otimes D_2)) = \begin{cases} (US_2S_1, (US_2\tau_1 \otimes U\tau_2)), & \\ & \text{if for some } \tau_1, \tau_2 \text{ and fresh } \alpha_2 \\ & W((D_2 : \alpha_2, \Gamma), D_1) = (S_1, \tau_1), \\ & W((D_1 : \tau_1, S_1\Gamma), D_2) = (S_2, \tau_2), \\ & \text{and } U = mgu(\tau_2, S_2S_1\alpha_2) \neq \mathit{fail} \\ \mathit{fail}, & \text{else} \end{cases}$$

By induction on t, we want to show that for all Γ, S, τ:

(i) $W(\Gamma, t)$ terminates.
(ii) If $W(\Gamma, t) = \mathit{fail}$, then there is no typing of t modulo Γ.
(iii) If $W(\Gamma, t) = (S, \tau)$, then $S\Gamma \vdash t : \tau$ is a principal typing of t modulo Γ.

Case $t = x$: (i) $W(\mathit{\Gamma}, x)$ searches the type context from left to right, unpacking boxes and merge-DRSs to lists of typed referents, and applies (var_1) to the first assumption $x : \tau$ found. Clearly, this terminates. (ii) If $W(\mathit{\Gamma}, x) = \mathit{fail}$, then no assumption $x : \tau$ is found in the (unpacked) context, so x is untypeable, since (var_1) cannot be applied to x. (iii) If $W(\mathit{\Gamma}, x) = (S, \tau)$, then $S = Id$ and $\mathit{\Gamma} = x : \tau, \mathit{\Gamma}'$ for some $\mathit{\Gamma}'$. Suppose $R\mathit{\Gamma} \vdash x : \rho$ is a typing of x modulo $\mathit{\Gamma}$. Then $R\mathit{\Gamma} = x : R\tau, R\mathit{\Gamma}'$, and hence $\rho = R\tau$ by (var_1). So $R\mathit{\Gamma} \vdash x : \rho$ is obtained from $S\mathit{\Gamma} \vdash x : \tau$ by instantiating with R.

Case $t = (D_1 \otimes D_2)$:

(i) $W(\mathit{\Gamma}, (D_1 \otimes D_2))$ terminates, since by induction, $W((D_2 : \alpha, \mathit{\Gamma}), D_1)$ terminates, for each (S_1, τ_1), $W((D_1 : \tau_1, S_1\mathit{\Gamma}), D_2)$ terminates, and for each (S_2, τ_2), $mgu(\tau_2, S_2S_1\alpha)$ terminates.

(ii) Suppose there is a typing of $(D_1 \otimes D_2)$ modulo $\mathit{\Gamma}$. For some S, τ_1, τ_2, the typing derivation ends in

$$\frac{D_2 : \tau_2, S\mathit{\Gamma} \vdash D_1 : \tau_1 \qquad D_1 : \tau_1, S\mathit{\Gamma} \vdash D_2 : \tau_2}{S\mathit{\Gamma} \vdash (D_1 \otimes D_2) : (\tau_1 \otimes \tau_2)}\,(\otimes).$$

Thus there is a typing of D_1 modulo $D_2 : \alpha_2, \mathit{\Gamma}$, whence, by induction, $W((D_2 : \alpha_2, \mathit{\Gamma}), D_1) \neq \mathit{fail}$, and there is a most general typing $D_2 : S_1\alpha_2, S_1\mathit{\Gamma} \vdash D_1 : \sigma_1$ of D_1 modulo $(D_2 : \alpha_2, \mathit{\Gamma})$. Since it is most general, there is a type substitution T_1 such that

$$D_2 : \tau_2, S\mathit{\Gamma} \vdash D_1 : \tau_1 \quad \equiv \quad D_2 : T_1S_1\alpha_2, T_1S_1\mathit{\Gamma} \vdash D_1 : T_1\sigma_1.$$

There is also a typing of D_2 modulo

$$(D_1 : \tau_1, S\mathit{\Gamma}) \equiv (D_1 : T_1\sigma_1, T_1S_1\mathit{\Gamma}),$$

hence a typing of D_2 modulo $(D_1 : \sigma_1, S_1\mathit{\Gamma})$. Therefore, by induction, $W((D_1 : \sigma_1, S_1\mathit{\Gamma}), D_2) \neq \mathit{fail}$, and there is a most general typing $D_1 : S_2\sigma_1, S_2S_1\mathit{\Gamma} \vdash D_2 : \sigma_2$ of D_2 modulo $(D_1 : \sigma_1, S_1\mathit{\Gamma})$. Since it is most general, there is a type substitution T_2 such that

$$\begin{aligned} D_1 : \tau_1, S\mathit{\Gamma} \vdash D_2 : \tau_2 \quad &\equiv \quad D_1 : T_1\sigma_1, T_1S_1\mathit{\Gamma} \vdash D_2 : \tau_2 \\ &\equiv \quad D_1 : T_2S_2\sigma_1, T_2S_2S_1\mathit{\Gamma} \vdash D_2 : T_2\sigma_2. \end{aligned}$$

So we have $T_2\sigma_2 = \tau_2 = T_1S_1\alpha_2$, and on the type variables of $S_1\mathit{\Gamma}$ and σ_1, $T_1 = T_2S_2$. On type variables β of $S_1\alpha_2$ which are not in $S_1\mathit{\Gamma}$ or σ_1, we have $S_2\beta = \beta$ as S_2 is idempotent. We can assume that β is not in the support of T_2 and put $T_2\beta := T_1\beta$, obtaining $T_1\beta = T_2S_2\beta$. Then from $T_2\sigma_2 = \tau_2 = T_1S_1\alpha_2 = T_2S_2S_1\alpha_2$, we know that σ_2 and $S_2S_1\alpha_2$ unify, so $mgu(\sigma_2, S_2S_1\alpha_2) \neq \mathit{fail}$. By the definition of W, it then follows that $W(\mathit{\Gamma}, (D_1 \otimes D_2)) \neq \mathit{fail}$.

(iii) Suppose $W(\mathit{\Gamma}, (D_1 \otimes D_2)) = (US_2S_1, (US_2\sigma_1 \otimes U\sigma_2))$ with $U, S_1, S_2, \sigma_1, \sigma_2$ as in the definition of W. Then with fresh α_2, $W((D_2 : \alpha_2, \mathit{\Gamma}), D_1) = (S_1, \sigma_1)$, $W((D_1 : \sigma_1, S_1\mathit{\Gamma}), D_2) = (S_2, \sigma_2)$, and $U = mgu(\sigma_2, S_2S_1\alpha_2) \neq \mathit{fail}$. By induction, we know that

(a) $D_2 : S_1\alpha_2, S_1\Gamma \ \vdash \ D_1 : \sigma_1$ is a principal typing of D_1 modulo $(D_2 : \alpha_2, \Gamma)$,
(b) $D_1 : S_2\sigma_1, S_2S_1\Gamma \ \vdash \ D_2 : \sigma_2$ is a principal typing of D_2 modulo $(D_1 : \sigma_1, S_1\Gamma)$.

By specializing the typing in (a) with US_2 and the one in (b) with U, one obtains

$$D_2 : US_2S_1\alpha_2, US_2S_1\Gamma \ \vdash \ D_1 : US_2\sigma_1, \quad \text{and} \quad D_1 : US_2\sigma_1, US_2S_1\Gamma \ \vdash \ D_2 : U\sigma_2.$$

Since $US_2S_1\alpha_2 = U\sigma_2$, we can apply the rule $(\otimes)$ and obtain a typing

$$US_2S_1\Gamma \ \vdash \ (D_1 \otimes D_2) : (US_2\sigma_1 \otimes U\sigma_2)$$

of $(D_1 \otimes D_2)$ modulo Γ. It remains to be shown that this is a most general typing.

So suppose $(D_1 \otimes D_2)$ has a typing modulo Γ. The last step in the typing derivation is

$$\frac{D_2 : \tau_2, S\Gamma \ \vdash \ D_1 : \tau_1, \qquad D_1 : \tau_1, S\Gamma \ \vdash \ D_2 : \tau_2}{S\Gamma \ \vdash \ (D_1 \otimes D_2) : (\tau_1 \otimes \tau_2)} \ (\otimes).$$

For the left subderivation of $D_2 : \tau_2, S\Gamma \ \vdash \ D_1 : \tau_1$ we may assume $\tau_2 = S\alpha_2$ for some fresh type variable α_2. So D_1 has a typing (S, τ_1) modulo $D_2 : \alpha_2, \Gamma$. By a) there is a type substitution T_1 such that $(S, \tau_1) = (T_1S_1, T_1\sigma_1)$, whence

$$D_2 : \tau_2, S\Gamma \ \vdash \ D_1 : \tau_1 \quad \equiv \quad D_2 : T_1S_1\alpha_2, T_1S_1\Gamma \ \vdash \ D_1 : T_1\sigma_1.$$

Now the right subderivation $D_1 : \tau_1, S\Gamma \ \vdash \ D_2 : \tau_2$ is a derivation of

$$D_1 : T_1\sigma_1, T_1S_1\Gamma \ \vdash \ D_2 : T_1S_1\alpha_2,$$

which is a typing of D_2 modulo $(D_1 : \sigma_1, S_1\Gamma)$. By b), there is a type substitution T_2 with

$$D_1 : \tau_1, S\Gamma \ \vdash \ D_2 : \tau_2 \quad \equiv \quad D_1 : T_2S_2\sigma_1, T_2S_2S_1\Gamma \ \vdash \ D_2 : T_2\sigma_2.$$

It follows that

$$S\Gamma \ \vdash \ (D_1 \otimes D_2) : (\tau_1 \otimes \tau_2) \ \equiv \& T_2S_2S_1\Gamma \ \vdash \ (D_1 \otimes D_2) : (T_2S_2\sigma_1 \otimes T_2\sigma_2).$$

To show that this is an instance of the typing

$$US_2S_1\Gamma \ \vdash \ (D_1 \otimes D_2) : (US_2\sigma_1 \otimes U\sigma_2),$$

we need a type substitution R such that $T_2 = RU$ on the type variables of $S_2S_1\Gamma$, $S_2\sigma_1$ and σ_2. We have $T_2\sigma_2 = T_1S_1\alpha_2$. As in (ii), $T_1 = T_2S_2$ on the type variables of $S_1\alpha_2$, so $T_2\tau_2 = T_2S_2S_1\alpha$, and since $U = mgu(\tau_2, S_2S_1\alpha_2)$, $T_2 = RU$ on the type variables of τ_2 and $S_2S_1\alpha_2$. On other type variables β, we have $U\beta = \beta = R\beta$ and can redefine $R\beta := T_2\beta$, to obtain $T_2 = RU$ on all type variables of $S_2S_1\Gamma$, $S_2\sigma_1$ and σ_2.

The remaining cases of t can be treated similarly.

Example 1. The lexicon entry for the indefinite article *a* in λ-DRT of [13] is

$$\lambda P \lambda Q(\delta x_i \top \otimes P(\hat{}x_i) \otimes Q(\hat{}x_i)) : (d,t),((d,t),t)$$

where d is the type of individual concepts and t the type of DRSs. Simplifying this to the extensional case and using the DRS-notation from above, type reconstruction yields the principal typing

$$\vdash\ \lambda P \lambda Q(\langle [x], \top \rangle \otimes Px \otimes Qx) : (\alpha \to \gamma) \to (\alpha \to \delta) \to [\alpha] \otimes \gamma \otimes \delta.$$

Instead of the basic type t for DRSs in [13], we have infinitely many types $[\sigma_1, \ldots, \sigma_n]$. Moreover, we have the principal typing

$$man' : e \to t\ \vdash\ \lambda x \langle [], man'\, x \rangle : e \to \mathbb{1}.$$

The unreduced meaning term for *a man* therefore is

$$\lambda P \lambda Q(\langle [x], \top \rangle \otimes Px \otimes Qx) \cdot \lambda x \langle [], man'\, x \rangle$$

and has the principal type $(e \to \delta) \to [e] \otimes \mathbb{1} \otimes \delta$.

For the kind of semantic checking of natural language text that we are interested in, we need to distinguish between different sorts of individuals. Lexical entries should assign different base types to the arguments of content words, in particular verbs and nouns. It is then useful, if not imperative, to have a lexicon with polymorphic types for the functional words like the indefinite article above, rather than be forced to put into the lexicon all the instance types needed for a specific application.

The type-checking in texts is slightly different from the one in programs: in programs, we need to check that in applications $f(a_1, \ldots, a_n)$, the type of the arguments equal (or are subtypes of) the argument types of the function, while in texts, in predications $v(np_1, \ldots, np_k)$ the types of the (generally quantified) argument noun phrases have to be related by type-raising to the argument types of the verb.

But in principle, we want to have the same *phase distinction* between type checking and evaluation: we want to build meaning terms according to the syntactic structure, then check if the meaning is typable, and only then perform semantic evaluation. Thus, evaluation only needs to be defined on typed expressions, and type checking would be pointless if evaluation would not preserve the type of expressions.

5 Reduction

We assume the familiar β-reduction and congruence rules of λ-calculus,

$$\frac{}{(\lambda x D \cdot s) \to D[x/s]}\,(\beta) \qquad \frac{D \to D'}{\lambda x D \to \lambda x D'},$$

$$\frac{D \to D'}{(D \cdot E) \to (D' \cdot E)}, \qquad \frac{E \to E'}{(D \cdot E) \to (D \cdot E')}.$$

The intended meaning of the merge $(D_1 \otimes D_2)$ of two value-DRSs with disjoint referent lists, $D_1 = \langle [x_1, \dots, x_n], \varphi\rangle$ and $D_2 = \langle [y_1, \dots, y_m], \psi\rangle$, is the value-DRS

$$\langle [x_1, \dots, x_n, y_1, \dots, y_m], (\varphi \wedge \psi)\rangle.$$

We therefore define the reduction (resp. evaluation) of DRS-expressions by the following *δ-reduction* rules:

$$\frac{}{\langle [], \varphi\rangle \otimes \langle [], \psi\rangle \to \langle [], (\varphi \wedge \psi)\rangle}\ (\delta_1), \qquad \frac{}{\langle [], \varphi\rangle \otimes \langle y, E\rangle \to \langle y, \langle [], \varphi\rangle \otimes E\rangle}\ (\delta_2),$$

$$\frac{}{\langle x, D\rangle \otimes E \to \langle x, D \otimes E\rangle}\ (\delta_3).$$

From these, the intended meaning for the merge of value-DRSs follows:

$$\langle [x_1, \dots, x_n], \varphi\rangle \otimes \langle [y_1, \dots, y_m], \psi\rangle \to^* \langle [x_1, \dots, x_n, y_1, \dots, y_m], (\varphi \wedge \psi)\rangle.$$

In order to use (δ_1) - (δ_3), by reductions we must achieve that arguments of $\otimes$ are value-DRSs. Hence we also need congruence rules for $\delta = \langle \cdot, \cdot \rangle$ and $\otimes$:

$$\frac{D \to E}{\langle x, D\rangle \to \langle x, E\rangle}\ (\delta_4), \quad \frac{D \to D'}{(D \otimes E) \to (D' \otimes E)}\ (\delta_5), \quad \frac{E \to E'}{(D \otimes E) \to (D \otimes E')}\ (\delta_6),$$

so that reductions can be performed in subterms of $\langle x, D\rangle$, $(D \otimes E)$ as well as $\lambda x D$ and $(D_1 \cdot D_2)$. Then the following reduction rules are derivable:

$$\frac{E \to^* E'}{\langle [], \varphi\rangle \otimes \langle y, E\rangle \to^* \langle y, \langle [], \varphi\rangle \otimes E'\rangle}\ (\delta_2^+), \qquad \frac{D \to^* D', \quad E \to^* E'}{\langle x, D\rangle \otimes E \to^* \langle x, D' \otimes E'\rangle}\ (\delta_3^+).$$

Normalization

It is obvious that applications of the δ-reduction rules do not lead to new occurrences of β-redexes. Therefore, expressions can be reduced by first performing β-reductions as long as possible, and only then apply δ-reduction rules. If we start with a typed expression, then from the strong normalization property for simply typed λ-calculus the first will terminate. It is also clear that the δ-reduction rules cannot lead to infinite reduction sequences.

Notice that on value-DRSs with disjoint top-level referents, $\otimes$ is associative, if we consider formula conjunction to be associative, i.e. use list $[\varphi_1, \dots, \varphi_n]$ of formulas, as we do in Sect. 6.2.

We would like to show that in a derivable typing statement $\Gamma \vdash s : \sigma$, where the "predicate" σ applies to the "subject" s, we may reduce the subject and still the predicate σ applies. However, this is not quite true: when we reduce a merge-DRS, the type constructor $\times$ is interpreted as a *cons* of a referent and a referent list, and $\otimes$ is interpreted as an *append* of referent lists, and since the type of a DRS mirrors its construction, we need to *cons* resp. *append* the lists of types of the referents.

We use the following type reductions, which amount to a recursive definition of *append* ($\otimes$) in terms of the empty list ($\mathbb{1}$) and *cons* ($\times$):

$$\frac{}{\mathbb{1} \otimes \mathbb{1} \rightharpoonup \mathbb{1}}\,(\delta'_1) \qquad \frac{}{\mathbb{1} \otimes (\sigma \times \rho) \rightharpoonup \sigma \times (\mathbb{1} \otimes \rho)}\,(\delta'_2)$$

$$\frac{}{(\sigma \times \rho) \otimes \tau \rightharpoonup \sigma \times (\rho \otimes \tau)}\,(\delta'_3)$$

Moreover, type reduction may operate on embedded type expressions:

$$\frac{\sigma \rightharpoonup \sigma'}{\sigma \times \tau \rightharpoonup \sigma' \times \tau}\,(\times') \qquad \frac{\tau \rightharpoonup \tau'}{\sigma \times \tau \rightharpoonup \sigma \times \tau'}\,(\times'')$$

$$\frac{\sigma \rightharpoonup \sigma'}{\sigma \otimes \tau \rightharpoonup \sigma' \otimes \tau}\,(\otimes') \qquad \frac{\tau \rightharpoonup \tau'}{\sigma \otimes \tau \rightharpoonup \sigma \otimes \tau'}\,(\otimes'')$$

$$\frac{\sigma \rightharpoonup \sigma'}{(\sigma \to \tau) \rightharpoonup (\sigma' \to \tau)}\,(\to') \qquad \frac{\tau \rightharpoonup \tau'}{(\sigma \to \tau) \rightharpoonup (\sigma \to \tau')}\,(\to'')$$

Example 1. (continued) Reducing the above term

$$\lambda P \lambda Q(\langle [x], \top \rangle \otimes Px \otimes Qx) \cdot \lambda x \langle [], man'\,x \rangle$$

by β-reductions gives $\lambda Q(\langle [x], \top \rangle \otimes \langle [], man'\,x \rangle \otimes Qx)$ and reducing further by δ-reductions leads to

$$\lambda Q(\langle [x], \top \wedge man'\,x \rangle \otimes Qx).$$

Its principal type $(e \to \delta) \to [e] \otimes \delta$ is obtained from the one of the unreduced term by applications of $(\otimes')$, (δ'_3), and (δ'_1) that simplify $[e] \otimes \mathbb{1} \otimes \delta$ to $[e] \otimes \delta$.

Since our types of DRSs closely reflect the construction of their top-level referent lists, in order to have a subject reduction property we need to consider types equivalent when they get equal by interpreting $\otimes$ as *append*, $\times$ as *cons*, and $\mathbb{1}$ as the empty list.

A more serious obstacle to subject-reduction is the typing rule (var_6) which permits us to ignore assumptions $(D_1 \cdot D_2) : \sigma$. In fact, the subject-reduction property does not hold in general.

Example 2. Consider the application of

$$\frac{D_2 : \tau_2, \Gamma \;\vdash\; D_1 : \tau_1 \qquad D_1 : \tau_1, \Gamma \;\vdash\; D_2 : \tau_2}{\Gamma \;\vdash\; (D_1 \otimes D_2) : (\tau_1 \otimes \tau_2)}\,(\otimes)$$

Suppose $(D_1 \otimes D_2) \to (D'_1 \otimes D_2)$ via $D_1 \to D'_1$. As we have seen above, we may have $x \in top(D'_1) \setminus top(D_1)$. In the left subderivation $D_1 : \tau_1, \Gamma \;\vdash\; D_2 : \tau_2$, a free occurrence of x in D_2 gets its type from Γ, while in the context $D'_1 : \tau_1, \Gamma$, it gets its type from $D'_1 : \tau_1$. Hence, it may be impossible to obtain $D'_1 : \tau_1, \Gamma \;\vdash\; D_2 : \tau_2$. (For example, take $D_1 : \tau_1 = \lambda y \langle x, E \rangle \cdot a : (\sigma \times \tau)$, $D_2 : \tau_2 = \langle [], Px \rangle : \mathbb{1}$.) Thus, $\Gamma \;\vdash\; (D_1 \otimes D_2) : (\tau_1 \otimes \tau_2)$ does not imply $\Gamma \;\vdash\; (D'_1 \otimes D_2) : (\tau_1 \otimes \tau_2)$.

The problem similarly arises for $(D_1 \Rightarrow D_2)$, $(D_1 \vee D_2)$, or $(D_1 \,;\, D_2)$, where D_1 may β-reduce to a DRS with a new top-level referent occurring free in D_2. This is a defect of λ-DRT terms which admit the binding part D_1 of such expressions to arise from a β-redex like $(\lambda zz \cdot D_1)$.

We will sidestep this problem for the application to pronoun resolution below by assuming

1. all λ-DRS-expressions used as meanings of lexical entries are closed and in normal form,
2. in substitution $t[x/s]$, bound variables (including referents) in t are renamed to make them distinct from free variables of s,
3. in $t[x/s]$, s is in normal form, and referents of s are renamed at each occurrence of x in t (in merge-factors, so that their scope does not extend).[3]
4. all bound variables are pairwise distinct; in particular, no referent is used twice as a binding variable.

In particular, we will use a call-by-value strategy when computing the meaning of phrases: if the meaning of a phrase is an application $\lambda xt \cdot s$, we will have λxt and s in normal form, and deliver a normal form $nf(t[x/s])$ of $t[x/s]$ as value, see the computation rules in Sect. 6.2. We think that the following weak form of the subject reduction property holds under the above assumptions:

Conjecture 1. If t and s are in normal form, and $\Gamma \vdash (\lambda xt \cdot s) : \tau$, then there is τ' with $\tau \rightharpoonup^* \tau'$ and $\Gamma \vdash nf(t[x/s]) : \tau'$.

However, we do not make use of that in the following; termination of reduction suffices.

6 Application to Pronoun Resolution

There are two possible ways to combine type reconstruction and pronoun resolution. Either one applies a pronoun resolution algorithm and then uses type reconstruction to check if the resolution is type-correct, or one first applies type reconstruction and then does pronoun resolution by exploiting the type information.

6.1 Type Informed Pronoun Resolution

The second way has been implemented [22]. It roughly proceeds as follows:

– Step 1: for each pronoun occurrence, introduce a fresh discourse referent x and extend the DRS by an anaphor-declaration like $anp(x, fem, sg)$. For the discourse referent y of each noun phrase that is not a pronoun, add an antecedent-declaration like $ant(y, masc, sg)$ to the DRS.

[3] Notice that $\lambda P(P \otimes P) \cdot \langle [z], [\varphi] \rangle$ then reduces to $(\langle [z_1], [\varphi(z/z_1)] \rangle \otimes \langle [z_2], [\varphi(z/z_2)] \rangle$, and further to $\langle [z_1, z_2], [\varphi(z/z_1), \varphi(z/z_2)] \rangle$, like turning $(\exists z\varphi \wedge \exists z\varphi)$ into prenex form $\exists z_1 \exists z_2 (\varphi_1(z/z_1) \wedge \varphi(z/z_2))$.

– Step 2: apply type reconstruction to get a most general typing for the discourse, including individual types e_i for discourse referents x as inferred from the occurrence context of the pronoun.
– Step 3: "resolve" an anaphoric (or cataphoric) pronoun by unifying its typed discourse referent $x : \alpha$ with some discourse referent $y : \beta$ of a possible antecedent *of the same type*, observing the grammatical properties of gender and number in the corresponding declarations $anp(x, g_x, n_x)$ and $ant(y, g_y, n_y)$.

A more detailed description is best obtained by explaining the relevant parts of the Prolog-program of [22].

A parse tree is represented as a list `[Root|Subtrees]` where the root is the syntactic category of the parsed expression. A discourse is either empty, with tree `[d]`, or the extension of a discourse by a sentence, and then has tree `[d,S,D]` where `S` is the parse tree of the final sentence and `D` the parse tree of the initial discourse.[4] For each parse tree, `sem(+Tree,-DRS)` computes a number of meanings. If the tree is a discourse, each meaning is a typed λ-DRS, otherwise an untyped λ-DRS in normal form.

```
% sem(+ParseTree,-DRS); for a discourse, DRS is typed
...
sem([d], drs([],[])) :- !.
sem([d,S,D], Sem) :-
    !, sem(S,SemS), sem(D,SemD), resolve(SemS,SemD,Sem).
```

Having computed a typed meaning `SemD` for the initial discourse and an untyped meaning `SemS` for the final sentence, we try to resolve anaphors of `SemS`, using `SemD` as accessible DRS for possible antecedents.

```
% resolve(+SemS,+SemD,-Sem)
resolve(SemS,SemD,Sem) :-
    type([],SemS,SemSTy,_TypS),
    resolve_drs([SemSTy,SemD],[DrsS,DrsD]),
    mergeTerm(DrsD + DrsS, Sem).
```

First, type reconstruction `type/4` is applied to `SemS`; as pronouns get fresh discourse referents in `SemS`, we can use the empty type context to find a principal type `TypS` for the DRS `SemS`. Actually, we use a modification of the type reconstruction algorithm that also returns a typed version `SemSTy` of `SemS`, which has type annotations at variable bindings (including referents in referent lists). This typed DRS `SemSTy` is resolved with `SemD` as accessible DRS, using `resolve_drs/2`; the modifications `DrsS` and `DrsD` are finally merged by appending the referents and formulas of `DrsS` to those of `DrsD`.

To resolve a DRS `drs(Refs,Fmls)` with respect to a stack `Ds1` of partially resolved accessible DRSs, we go through the formulas, which may contain unresolved DRSs, resolve these, and construct a resolved form of `drs(Refs,Fmls)` on top of the stack:

[4] To prevent Prolog's top-down parsing strategy from diverging for left-recursive grammar rules `d -> d, s.`, we use a right-recursive rule `d --> s, d.` for discourses and reverse the sequence of input sentences before parsing.

```
% resolve_drs(+DRSs, -resolvedDRSs)
resolve_drs([drs(Refs,Fmls)|Ds1],RDs):-
    resolve_fml(Fmls,[drs(Refs,[])|Ds1],RDs).
```

If a formula is built from DRSs, like $(D_1 \Rightarrow D_2)$, $(D_1 \vee D_2)$, or $\neg D$, the component DRSs are resolved in term, respecting the accessibility conditions of DRT, and the formula built form the resolved component DRSs is added to the result-DRS under construction, before the remaining formulas are processed:

```
% resolve_fml(+Fmls,[?resultDRS|+accessDRSs],-resolvedDRSs)
resolve_fml([(D1 => D2)|Fmls],Ds,RDs):-
    !, resolve_drs([D1|Ds],[D1r|Dsr]),
    resolve_drs([D2,D1r|Dsr],[D2R,D1R,drs(R,F)|Ds3]),
    resolve_fml(Fmls,[drs(R,[(D1R => D2R)|F])|Ds3],RDs).

...
```

If the formula is an anaphor `anp(Ref,Gen,Num)` with typed(!) referent `Ref` and gender and number information, one tries to find a suitable antecedent in the result-DRS under construction (i.e. in the pronoun's left textual context in the current sentence) or the accessible DRSs, or in the remaining formulas of the DRS currently under process:

```
resolve_fml([anp(Ref,Gen,Num)|Fmls], [drs(R,F)|Ds1], RDs) :-
    !, ( ( % in sentence prefix or previous sentences
           resolve_anp(Ref,Gen,Num,[drs(R,F)|Ds1])
         ; % in sentence suffix
           resolve_anp(Ref,Gen,Num,[drs(R,Fmls)])
         ),
         delete_ref(Ref,R,NewR),  % omit duplicates of Ref
         NewD = drs(NewR,F) % omit anp(Ref,..) in the result DRS
       ; NewD = drs(R,[anp(Ref,Gen,Num)|F]) % or: fail, to
       ),                                    % exclude unre-
    resolve_fml(Fmls,[NewD|Ds1],RDs).       % solved anaphors
```

Possessive pronouns are handled by looking for antecedents in their left context only.

To find a suitable antecedent, simply choose some of the accessible DRSs and some antecedent among its formulas that can be unified with the referent:

```
% resolve_anp(+Ref,+Gen,+Num,+DRSs)
resolve_anp(Ref,Gen,Num,Ds):-
    member(drs(_Refs,Fs),Ds),
    member(ant(Ref,Gen,Num),Fs).
```

By using the same variables `Ref`, `Gen`, `Num`, Prolog unifies a typed anaphor `R:Ty` with a typed antecedent `R':Ty'` of the same number and gender features.

Atomic formulas can just be transferred to the result-DRS under construction, and when all formulas of the DRS are processed, the sequence of resolved formulas is reversed to its expected order:

```
resolve_fml([Fml|Fmls],[drs(R,F)|Ds1],RDs):-
    !, resolve_fml(Fmls,[drs(R,[Fml|F])|Ds1],RDs).

resolve_fml([],[drs(R,F)|Ds],[drs(R,Frev)|Ds]):-
    !, reverse(F,Frev).
```

The stack of resolved DRSs with a resolved form of the DRS `drs(Refs,Fmls)` on top is returned.

6.2 Example

We assume that nouns N and relational nouns RN are classified according to gender $g \in \{m, f, n\}$ (masculine, feminine, neuter), and implicitly inflect for number $n \in \{sg, pl\}$ and case c. (We use gender m as in the corresponding German nouns and pronouns to get more possible antecedents below.)

1. Content words are assigned a meaning and a type in the lexicon, for example:

expression	*meaning*	*type*
$Galilei : PN$	$galilei$	h
$Jupiter : PN$	$jupiter$	s
$astronomer : N$	$\lambda x\langle[], [ant(x, m, sg), astronomer(x)]\rangle$	$h \to \mathbb{1}$
$star : N$	$\lambda x\langle[], [ant(x, m, sg), star(x)]\rangle$	$s \to \mathbb{1}$
$moon : RN$	$\lambda x\lambda y\langle[], [ant(x, m, sg), moon(x, y)]\rangle$	$s \to (s \to \mathbb{1})$
$shine : V$	$\lambda x\langle[], [shine(x)]\rangle$	$s \to \mathbb{1}$
$observe : TV$	$\lambda x\lambda y\,\langle[], [observe(x, y)]\rangle$	$h \to (s \to \mathbb{1})$
$discover : TV$	$\lambda x\lambda y\,\langle[], [discover(x, y)]\rangle$	$h \to (s \to \mathbb{1})$

Pronouns inflect for number, gender, and case, if we consider person fixed to 3rd person. Like determiners, pronouns have polymorphic type; i.e. from their untyped λ-DRS-meaning we reconstruct their most general (schematic) type.

expression	*meaning*	*principal type*
$he : Pron$	$\lambda P(\langle[x], [anp(x, m, sg)]\rangle \otimes P\,x)$	$(\alpha \to \beta) \to [\alpha] \otimes \beta$
$she : Pron$	$\lambda P(\langle[x], [anp(x, f, sg)]\rangle \otimes P\,x)$	$(\alpha \to \beta) \to [\alpha] \otimes \beta$
$his : PossPron$	$\lambda R\lambda P(\,\langle[x, y], [anposs(y, m, sg)]\rangle$	$(\alpha \to \beta \to \gamma)$
	$\otimes(R\,x\,y \otimes P\,x))$	$\to (\alpha \to \delta)$
		$\to [\alpha, \beta] \otimes \gamma \otimes \delta$
$who : RelPron$	$\lambda P\lambda x\,(P\,x)$	$(\alpha \to \beta) \to (\alpha \to \beta)$
$a : Det$	$\lambda N\lambda P(\langle[x], []\rangle \otimes (N\,x \otimes P\,x))$	$(\alpha \to \beta) \to (\alpha \to \gamma)$
		$\to [\alpha] \otimes \beta \otimes \gamma$
$every : Det$	$\lambda N\lambda P\langle[], [(\langle[x], []\rangle \otimes N\,x) \Rightarrow P\,x]\rangle$	$(\alpha \to \beta) \to (\alpha \to \gamma)$
		$\to \mathbb{1}$
eq	$\lambda x\lambda y.eq(x, y)$	$\alpha \to (\alpha \to t)$

Each use of a personal, relative, or possessive pronoun uses a new referent x. Moreover, *eq*, *anp*, *anposs*, *ant* have polymorphic lexical (not reconstructed) type.

2. Compound expressions are built according to grammar rules; each grammar rule is accompanied by one or several meaning computation rules. Some examples are:

$$\frac{p : PN_g}{p : NP}\,(S\ 1) \qquad \frac{p'}{\lambda P(\langle [x], [\,ant(x, g, sg),\ eq(x, p')]\rangle \otimes P \cdot x)}\,(C\ 1)$$

$$\frac{p : Pron_{g,n}}{p : NP}\,(S\ 2) \qquad \frac{p'}{p'}\,(C\ 2)$$

$$\frac{d : Det \quad n : N}{d\ n : NP}\,(S\ 3) \qquad \frac{d' \quad n'}{nf(d' \cdot n')}\,(C\ 3)$$

$$\frac{p : PossPron \quad r : RN}{p\ r : NP}\,(S\ 4) \qquad \frac{p' \quad r'}{nf(p' \cdot r')}\,(C\ 4)$$

$$\frac{np_1 : NP \quad v : TV \quad np_2 : NP}{np_1\ v\ np_2\ : S}\,(S\ 5) \qquad \frac{np_1' \quad v' \quad np_2'}{nf(np_1' \cdot \lambda x(np_2' \cdot \lambda y(v' \cdot x \cdot y)))}\,(C\ 5)$$

$$\frac{}{\epsilon : \mathcal{D}}\,(S\ 6) \qquad \frac{}{\langle [], []\rangle}\,(C\ 6)$$

$$\frac{d : \mathcal{D} \quad s : S}{d\ ;\ s : \mathcal{D}}\,(S\ 7) \qquad \frac{d' \quad s'}{(d'' \otimes s'')}\,(C\ 7)$$

An additional computation rule ($C\ 5'$) for sentences np_1 v np$_2$ *: S* might give np_2 wide scope. In (C 7), d'' and s'' are obtained by pronoun-resolution from most general typings of d' and s' in the empty type context, i.e. $resolve(s', d', d'' \otimes s'')$ by the resolution algorithm explained above.

3. Let us consider the sample discourse *Galilei observed a star. He discovered his moon.* The first sentence is constructed with (S 1), (S 3), and (S 5). We compute the meaning of the subject as

$$np_1' = \lambda P(\langle [x], [ant(x, m, sg), eq(x, galilei)]\rangle \otimes P \cdot x),$$

the meaning of the object as

$$\begin{aligned} np_2' &= nf(\lambda N \lambda P(\langle [x], []\rangle \otimes (N\, x \otimes P\, x)) \cdot \lambda x \langle [], [ant(x, m, sg), star(x)]\rangle) \\ &= nf(\lambda P(\langle [x], []\rangle \otimes (\langle [], [ant(x, m, sg), star(x)]\rangle \otimes P\, x))) \\ &= \lambda P(\langle [x], [ant(x, m, sg), star(x)]\rangle \otimes P\, x) \end{aligned}$$

and from those obtain the sentence meaning by the computation rule for (S 5) as

$$\begin{aligned}
s'_1 &= \mathit{nf}(np'_1 \cdot \lambda x(np'_2 \cdot \lambda y(v' \cdot x \cdot y)))\\
&= \mathit{nf}(np'_1 \cdot \lambda x(np'_2 \cdot \lambda y(\langle[], [\mathit{observe}(x,y)]\rangle)))\\
&= \mathit{nf}(np'_1 \cdot \lambda x.(\langle[x], [\mathit{ant}(x,m,sg), \mathit{star}(x)]\rangle \otimes P\, x)[P/\lambda y\langle[], [\mathit{observe}(x,y)]\rangle])\\
&= \mathit{nf}(np'_1 \cdot \lambda x(\langle[\tilde{x}], [\mathit{ant}(\tilde{x},m,sg), \mathit{star}(\tilde{x})]\rangle \otimes \langle[], [\mathit{observe}(x,\tilde{x})]\rangle))\\
&= \mathit{nf}(np'_1 \cdot \lambda x\langle[y], [\mathit{ant}(y,m,sg), \mathit{star}(y), \mathit{observe}(x,y)]\rangle)\\
&= \mathit{nf}((\langle[x], [\mathit{ant}(x,m,sg), \mathit{eq}(x,\mathit{galilei})]\rangle \otimes P\, x)[P/\lambda x\langle[y], [\mathit{ant}(\ldots), \ldots]\rangle])\\
&= \mathit{nf}((\langle[x], [\mathit{ant}(x,m,sg), \mathit{eq}(x,\mathit{galilei})]\rangle \otimes \langle[y], [\mathit{ant}(y,m,sg), \ldots]\rangle))\\
&= \langle[x,y], [\mathit{ant}(x,m,sg), \mathit{eq}(x,\mathit{galilei}), \mathit{ant}(y,m,sg), \mathit{star}(y), \mathit{observe}(x,y)]\rangle
\end{aligned}$$

From the type assumptions for nouns and verbs (and *eq*), type reconstruction can annotate the bound variables of s'_1 as

$$\langle[x:h, y:s], [\mathit{ant}(x:h,m,sg), \mathit{eq}(x,\mathit{galilei}), \mathit{ant}(y:s,m,sg), \ldots]\rangle$$

and return a most general type $\langle[h,s],t\rangle$. In the second sentence, the subject *he* has meaning

$$np'_1 = \lambda P(\langle[x], [\mathit{anp}(x,m,sg)]\rangle \otimes P\, x),$$

which receives the following annotation and principal type:

$$\lambda P:(\alpha \to \beta)(\langle[x:\alpha], [\mathit{anp}(x:\alpha,m,sg)]\rangle \otimes P\, x) : (\alpha \to \beta) \to [\alpha] \otimes \beta.$$

The object *his moon* gets the meaning[5]

$$\begin{aligned}
np'_2 = \mathit{nf}(&\lambda R\lambda P(\langle[x,y], [\mathit{anposs}(y,m,sg)]\rangle \otimes (R\, x\, y \otimes P\, x))\\
&\cdot\lambda x\lambda y\langle[], [\mathit{ant}(x,m,sg), \mathit{moon}(x,y)]\rangle)\\
= &\lambda P(\langle[x,y], [\mathit{anposs}(y,m,sg), \mathit{ant}(x,m,sg), \mathit{moon}(x,y)]\rangle \otimes P\, x),
\end{aligned}$$

which type reconstruction annotates to

$$\begin{aligned}
\lambda P:(s \to \alpha)(\langle[x:s, y:s], [\,&\mathit{anposs}(y:s,m,sg),\\
&\mathit{ant}(x:s,m,sg), \mathit{moon}(x,y)]\rangle \otimes P\, x)
\end{aligned}$$

and to which it assigns a most general type $(s \to \alpha) \to [s,s] \otimes \alpha$. By the computation rule for (S 5), the meaning of the second sentence is

$$\begin{aligned}
s'_2 &= \mathit{nf}(np'_1 \cdot \lambda x(np'_2 \cdot \lambda y(v' \cdot x \cdot y)))\\
&= \mathit{nf}(np'_1 \cdot \lambda x(np'_2 \cdot \lambda y\langle[], [\mathit{discover}(x,y)]\rangle))\\
&= \mathit{nf}(np'_1 \cdot \lambda x\langle[\tilde{x},y], [\,\mathit{anposs}(y,m,sg), \mathit{ant}(\tilde{x},m,sg), \mathit{moon}(\tilde{x},y),\\
&\qquad \mathit{discover}(x,\tilde{x})]\rangle)\\
&= \mathit{nf}((\langle[x], [\mathit{anp}(x,m,sg)]\rangle \otimes P\, x)[P/\lambda x\langle[\tilde{x},y], [\mathit{anposs}(y,m,sg), \ldots,]\rangle)\\
&= \langle[x,\tilde{x},y], [\,\mathit{anp}(x,m,sg), \mathit{anposs}(y,m,sg),\\
&\qquad \mathit{ant}(\tilde{x},m,sg), \mathit{moon}(\tilde{x},y), \mathit{discover}(x,\tilde{x})]\rangle.
\end{aligned}$$

[5] By an additional reduction $D_1 \otimes (D_2 \otimes D_3) \to (D_1 \otimes D_2) \otimes D_3$ when D_1, D_2 are value-DRSs.

If several computation rules can be applied, a sentence can get several untyped meanings this way. As normalisation has to return fresh bound variables, we write

$$\begin{aligned} s_2' = \langle [u, v, z], [\, &\mathit{anp}(u, m, sg), \mathit{anposs}(z, m, sg), \\ &\mathit{ant}(v, m, sg), \mathit{moon}(v, z), \mathit{discover}(u, v)] \rangle. \end{aligned}$$

4. Pronoun resolution for the discourse $\epsilon\,;\, s_1\,;\, s_2$ proceeds as follows.
   (a) The most general typing of the meaning $\langle [], [] \rangle$ of ϵ in the empty context is $\vdash \langle [], [] \rangle : \mathbb{1}$.
   (b) Type reconstruction is applied to the first sentence, followed by pronoun resolution with $\langle [], [] \rangle : \mathbb{1}$. As no pronoun occurred in s_1, the type-annotated version of s_1' is returned:

$$\begin{aligned} s_1'' &= \langle [x : h, y : s], [\, \mathit{ant}(x : h, m, sg), \mathit{eq}(x, \mathit{galilei}), \\ &\qquad \mathit{ant}(y : s, m, sg), \mathit{star}(y), \mathit{observe}(x, y)] \rangle \\ &= \langle [], [] \rangle \otimes s_1''. \end{aligned}$$

   (c) Type reconstruction is applied to (each of) the meaning(s) of the next sentence, followed by pronoun resolution with s_1''. Here type reconstructions just returns

$$\begin{aligned} s_2'' = \langle [u : h, v : s, z : s], [\, &\mathit{anp}(u : h, m, sg), \mathit{anposs}(z : s, m, sg), \\ &\mathit{ant}(v : s, m, sg), \mathit{moon}(v, z), \mathit{discover}(u, v)] \rangle, \end{aligned}$$

   where the types of u, v, z are derived from the argument types of nouns and verbs whose argument positions they occupy. The anaphor $u : h$ has no antecedent in the current sentence, as $v : s$ has different type. Assuming that possessives have to be resolved in their left context, the possessive anaphor $z : s$ also cannot be resolved against $v : s$.
   (d) Pronouns of s_2 may also be resolved against antecedents in the type-annotated left context, s_1''. For each typed anaphor, we search for a suitably typed antecedent, unify the referents and remove the anaphor referent in the DRS of the current sentence, s_2''. For the anaphor $\mathit{anp}(u : h, m, sg)$, the only type-compatible antecedent in s_1'' is $\mathit{ant}(x : h, m, sg)$, so we unify u with x (i.e. rename u by x in s_2''), remove $x : h$ from its referent list and $\mathit{anp}(x : h, m, sg)$ from its formulas, getting a partially resolved DRS

$$\begin{aligned} \langle [v : s, z : s], [\, &\mathit{anposs}(z : s, m, sg), \mathit{ant}(v : s, m, sg), \\ &\mathit{moon}(v, z), \mathit{discover}(x, v)] \rangle. \end{aligned}$$

   The next formula is a possessive anaphor $\mathit{anposs}(z : s, m, sg)$. As we want these to be resolved in their left context only, $z : s$ cannot be resolved against $v : s$. But it can be resolved against $\mathit{ant}(y : s, m, sg)$ in s_1'', which leads to

$$r(s_2'') = \langle [v : s], [\mathit{ant}(v : s, m, sg), \mathit{moon}(v, y), \mathit{discover}(x, v)] \rangle$$

   as the resolved"'result"'-DRS of s_2''.

(e) Finally, the resolved version of s''_2 is merged with s''_1, yielding

$$\begin{aligned} s''_1 \otimes r(s''_2) = \langle [x:h, y:s, v:s], & \\ & [ant(x:h,m,sg), eq(x, galilei), ant(y:s,m,sg), star(y), \\ & \; observe(x,y), ant(v:s,m,sg), moon(v,y), discover(x,v)]\rangle \end{aligned}$$

as the typed meaning of the discourse $d = \epsilon\,;\, s_1\,;\, s_2$.

6.3 Type Reconstruction for Bach-Peters-Sentences

One of the motivations for the *symmetric* merge-operator $\otimes$ was hinted at, but not elaborated in [13, p. 480]: the potential to treat Bach-Peters-sentences "in which two phrases are connected by both an anaphor and a cataphor", like $[$*The boy who deserved it*$_y]_x$ *got* $[$*the prize he*$_x$ *wanted*$]_y$. We use variants of (S 2), (S 5) and (C 2), (C 5) as syntax and computation rules for relative clauses

$$\frac{p : RelPron}{p : RelNP}\,(S\ 2') \qquad\qquad \frac{p'}{p'}\,(C\ 2')$$

$$\frac{np_1 : RelNP \quad np_2 : NP \quad v : TV}{np_1\ np_2\ v\ :\ RelS}\,(S\ 5') \qquad \frac{np'_1 \quad v' \quad np'_2}{nf(np'_1 \cdot \lambda x(np'_2 \cdot \lambda y(v' \cdot x \cdot y)))}\,(C\ 5')$$

$$\frac{d : Det \quad n : N \quad s : RelS}{d\ n\ s\ :\ NP}\,(S\ 8) \qquad \frac{d' \quad n' \quad s'}{nf(d' \cdot \lambda x(n'\,x \wedge s'\,x))}\,(C\ 8)$$

Omitting the grammatical features and the uniqueness conditions for the definite article, the untyped meaning of *a boy who deserves it gets the prize he wanted* is obtained via

$$\lambda P(\begin{array}{|l|}\hline x,y \\ \hline ant(x) \\ boy(x) \\ anp(y) \\ deserve(x,y) \\ \hline \end{array} \otimes P\,x) \cdot \lambda z \begin{array}{|l|}\hline x',y' \\ \hline ant(y') \\ prize(y') \\ anp(x') \\ want(x',y') \\ get(z,y') \\ \hline \end{array} \quad \to_\beta \quad \begin{array}{|l|}\hline x,y \\ \hline ant(x) \\ boy(x) \\ anp(y) \\ deserve(x,y) \\ \hline \end{array} \otimes \begin{array}{|l|}\hline x',y' \\ \hline ant(y') \\ prize(y') \\ anp(x') \\ want(x',y') \\ get(x,y') \\ \hline \end{array}\,.$$

From suitable type assumptions for nouns and verbs in the lexicon, with a type h of humans and e of objects, type reconstruction would infer types $x : h, y : e, x' : h, y' : e$, and hence type-respecting pronoun resolution could only resolve x' against x and y against y', as expected.

The typing rule for $\otimes$-DRSs was designed for merge-DRSs whose factors are linked through resolving cataphors and anaphors by type-independent "coindexing" or referent unification. Type-checking a DRS $\langle [x], \varphi(x,y)\rangle \otimes \langle [y], \psi(x,y)\rangle$ of this kind leads to a typing problem of the form

$$\dfrac{\dfrac{x:\alpha, y:\beta\ \vdash\ \varphi(x,y):t}{\vdots} \qquad \dfrac{y:\beta, x:\alpha\ \vdash\ \psi(x,y):t}{\vdots}}{\vdash\ \langle [x], \varphi(x,y)\rangle \otimes \langle [y], \psi(x,y)\rangle : [\alpha] \otimes [\beta]}$$

The type variables α, β get instantiated when the two typing problems in the premise are solved. As we perform merging of value-DRSs during normalization, we need the typing rule ($\otimes$) only when a merge-factor is not a value-DRS, not for Bach-Peters-sentences.

6.4 Supporting Pronoun Translation

To translate between natural languages, we need to resolve pronouns in order to translate them correctly: the gender of the translated pronoun is generally not the gender of the source language pronoun, but the gender of the antecedent noun phrase in the target language, which in turn depends on the antecedent of the pronoun in the source sentence. For example, Google translates the English text *The child opened the box. It contained a pen.* into the German *Das Kind öffnete die Schachtel. Es enthielt einen Stift.*, where neuter *es* should be feminine *sie*. A type difference between humans h and things e and the verb type *contain/enthalten* : $e \to e \to t$ shows that *it* at position of type e cannot refer to *the child* : $(h \to t) \to t$ at position of type h. But only if *it* is resolved to *the box* : $(e \to t) \to t$, the gender for the German pronoun *er/sie/es* can be inferred to be the gender of the translation *die Schachtel* of *the box*, i.e. feminine.

6.5 Related Work

On the practical side, discourse representation structures are used as intermediate representation of meaning when translating texts from natural language to first-order logic. This is done for large-scale processing of newspaper texts by the *C&C/Boxer* program[6] [5] and for mathematical texts by the *Naproche* system [4].

The Groningen Meaning Bank [3] (GMB) is a large collection of English texts for which *C&C* computes syntactic analyses in categorial grammar and *Boxer* turns them into DRSs and first-order formulas. By using referents for individuals, events and times and predicates for thematic roles, *Boxer* covers far more of discourse representation theory than we do. In the examples of the GMB, nouns are classified according to animacy (human, non-concrete, etc.), which can be seen as type assignments. But, apparently, these classifications are not related to the meaning of verbs and hence not used in the pronoun resolution process. For example, in *Ein Mann füttert einen Hund; wenn er ihn beißt, schlägt er ihn.*, our system correctly resolves the four pronouns in the only type-compatible way (the first *er* to *Hund*, the second to *Mann* etc.), if we provide types h for humans, a for animals and typings for nouns *Mann* : $h \to t$, *Hund* : $a \to t$ and verbs *füttern*, *schlagen* : $h \to a \to t$ and *beißen* : $a \to h \to t$. The *C&C/Boxer* program, when we use masculine pronouns in the English input *A man feeds a dog. If he bites him, he beats him.*, resolves both subject

[6] Since the link provided in [5] did not work, we were only able to access *C&C/Boxer* via its demo version `gmb.let.rug.nl/webdemo/demo.php` of the Groningen Meaning Bank.

pronouns *he* to the *man* and both object pronouns to the *dog* (as one can infer from the logical formula). Thus, if the argument slots of verbs of the GMB were annotated with animacy, too, its pronoun resolution and meaning translation could be improved by using our type-respecting resolution procedure. As type distinctions are easier to make in mathematics than for natural language, a similar improvement can be expected for the anaphora resolution in systems using DRS-like proof representations like [4,8].

On the theoretical side, there is a growing amount of work (cf. [1,14,17,20]) that uses constructive type theory to develop semantic representations for natural language. In this setting, the notion of type is extended (from simple types, i.e. intuitionistic propositional formulas) to first-order formulas, and proofs of the formulas are the objects of these types. In particular, proofs of existential statements $\exists x\varphi$ consist of pairs (t, p) where t is a term denoting an individual and p a proof of $\varphi[x/t]$. Such terms t may then be used to resolve anaphoric expressions. For example, Mineshima [17] uses constructive type theory enriched by ϵ-terms to treat definite descriptions; the use of an ϵ-term has to be justified by an existential sentence, whose proof object then contains a referent for the description. Instead of ϵ-terms, Bekki [1] has terms $(@ : \gamma \to e)(c)$ of unknown choice functions @ applied to contexts c to select suitable referents of type e; by instantiating γ and constructing an object of type $\gamma \to e$ from proof objects in the typing environment Γ, this amounts to "anaphora resolution by proof search and type checking". Clearly, the contexts Γ used in constructive type theory provide a more general domain to search for referents than the typed DRS of the textual left context in our system; for example, one can have background assumptions that do not arise from translation of the textual left context, which is useful to handle bridging anaphora [14]. However, the formulation of background knowledge may often be unfeasible, and proof search in constructive type theory seems more complex that type reconstruction by unification from simple type annotations in the lexicon.

7 Open Problems

Extension to generalized quantifiers and plural pronouns. In [16], we have shown that type reconstruction for Montague grammar with plural noun phrases can be used to resolve some plural ambiguities. The idea is that plural noun phrases in general have several types, for distributive, reciprocal and collective readings, but argument types of predicates only unify with one of those. The type reconstruction program of [16] has been changed in [22] to type reconstruction for λ-DRT and extended to type-respecting pronoun resolution for singular pronouns. So far, type reconstruction for plurals is not adapted to λ-DRT yet. To interpret *She introduced the guests to each other*, for example, we would need discourse referents X for sets of individuals and apply the symmetric predicate distributively to any 2-element subset of X. As our system admits second-order discourse referents X, it seems possible to add type-respecting pronoun resolution for plural pronouns. For this, one should consider if the treatment of plurals and generalised quantifiers via "duplex conditions" [10] can be given a formulation that allows for principal types and type reconstruction.

First-order λ-DRT. In contrast to typed versions of λ-DRT, our untyped version is a kind of "higher-order" DRT: there is no demand that discourse referents have individual type. So we can type some expressions which, from a traditional point of view, should be untypable. For example,

$$P : \sigma \to t \;\vdash\; (\boxed{\begin{array}{c} x \\ \hline Px \end{array}} \otimes x) : [\sigma] \otimes \sigma$$

is a most general typing, using σ both as a referent-type and as a drs-type. To avoid such defects, we could introduce different kinds of types, notice when a type variable must be instantiated by an individual resp. by a drs-type, and forbid to equate types of different kinds. But in realistic cases, conditions of a DRS express properties of referents using predicates with individual argument type, which makes a formal restriction to first-order referents unnecessary.

Principal typings for pronoun resolved discourses. Does type-respecting pronoun resolution as suggested above "preserve principal types"? More precisely, in a merge-DRS $D_1 \otimes D_2$ of two typed DRSs with disjoint toplevel referent lists and principal types, we unify referents $x : \sigma$ of D_1 and $x' : \sigma'$ of D_2 by substituting x for y in D_2 and removing $x' : \sigma'$ from its referent list. Applying the most general unifier U of $x : \sigma$ and $x' : \sigma'$ gives a typed DRS $UD_1 \otimes UD_2$. Can one prove that $UD_1 \otimes UD_2'$ corresponds to the principal typing of $\tilde{D}_1 \otimes \tilde{D}_2'$, where D_2' is the modification of D_2 by the pronoun resolution, and $\tilde{D}_1$ resp. $\tilde{D}_2'$ are the untyped versions of D_1 and D_2'?

Semantics. A semantics for typed λ-DRT is given in [13,15], with a compositional meaning for the *symmetric* ⊗. The relational interpretation of [19] for the unsymmetric merge $(;)$ is not sufficient for our purposes. The *Dynamic lambda calculus* DLC of [11,12] claims to give a typed semantics for a system subsuming typed λ-DRT, but we found their types involving individual variables fairly incomprehensible. In order to show that the typing and reduction rules given here are correct, we ought to interprete typings $\Gamma \vdash t : \tau$ in a suitable domain-model of the *untyped* λ-calculus, like the one in [21], and handle free type variables as universally quantified. We have not yet tried to do so.

8 Conclusion

Our aim was to use semantic type information from the lexicon to reduce the number of possible antecedents of an anaphor to type-compatible ones. For this, a single type e of entities is too crude. Many verbs and nouns in natural language can only be applied to facts/propositions, inanimate physical objects, animals, or humans, respectively. Candidates for pronoun resolution can be reduced with these types quite reasonably in many situations. Of course, in a discourse about humans only, the reduction in candidates may be minimal.

The basic idea is simple: a pronoun gets a type from its occurrence as an argument of a verb, and a noun phrase gets a type from its head noun and the verb argument type of its occurrence; hence, one can filter the set of possible antecedents of a pronoun by comparing their types. To do this efficiently, we prefer a system of simple types with schematic types for function words like determiners, in which complex expressions have principal types that can easily be reconstructed from type assumptions for content words. (A complex expression can have a principal type for each choice of types of its words.)

Using DRSs provides us with DRTs [10] notion of possible "accessible" antecedent noun phrases. Our typing rules for λ-DRT expressions closely reflect the accessibility conditions of DRT; this is to be expected, as the antecedent noun phrase provides a type assumption for its discourse referent, which in turn corresponds to the pronoun occurrences referring to the antecedent. However, the peculiarities of λ-DRT concerning the subject-reduction property might be a good reason to consider a mathematically "cleaner" language for expressing the dynamics of discourse, such as simply typed λ-calculus with continuation semantics [7]. But in contrast to [7], we are not *assuming* pronoun resolution via some oracles, but rather integrate a type reconstruction algorithm into a pronoun resolution algorithm – in a particularly simple way.

Acknowledgement. We thank the referees for a number of critical remarks and questions that helped to improve the presentation.

References

1. Bekki, D.: Representing anaphora with dependent types. In: Asher, N., Soloviev, S. (eds.) LACL 2014. LNCS, vol. 8535, pp. 14–29. Springer, Heidelberg (2014)
2. Bos, J., Mastenbroek, E., McGlashan, S., Millies, S., Pinkal, M.: A compositional DRS-based formalism for NLP-applications. In: Proceedings of International Workshop on Computational Semantics, Tilburg, pp. 21–31 (1994)
3. Bos, J., Basile, V., Evang, K., Venhuizen, N., Bjerva, J.: The Groningen Meaning Bank. In: Ide, N., Pustejovsky, J. (eds.) Handbook of Linguistic Annotation. Springer, Berlin (2017, to appear). http://gmb.let.rug.nl
4. Cramer, M., Fisseni, B., Koepke, P., Kühlwein, D., Schröder, B., Veldman, J.: The Naproche project controlled natural language proof checking of mathematical texts. In: Fuchs, N.E. (ed.) CNL 2009. LNCS (LNAI), vol. 5972, pp. 170–186. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14418-9_11
5. Curran, J.R., Clark, S., Bos, J.: Linguistically motivated large-scale NLP with C&C and boxer. In: Proceedings of ACL, Prague, June 2007, pp. 33–36. Association for Computational Linguistics (2007)
6. Damas, L., Milner, R.: Principal type-schemes for functional programs. In: Proceedings of 9th ACM Symposium on Principles of Programming Languages, pp. 207–212 (1982)
7. de Groote, P.: Towards a Montagovian account of dynamics. In: Gibson, M., Howell, J. (eds.) Proceedings of SALT XVI, vol. 16 (2006)
8. Ganesalingam, M.: The Language of Mathematics. LNCS, vol. 7805. Springer, Heidelberg (2013)
9. Hindley, R.: The principal type-scheme of an object in combinatory logic. Trans. Am. Math. Soc. **146**, 29–60 (1969)

10. Kamp, H., Reyle, U.: From Discourse to Logic. Kluwer, Dordrecht (1993)
11. Kohlhase, M., Kuschert, S.: Towards a dynamic type theory. Technical report, Universität des Saarlands (1996)
12. Kohlhase, M., Kuschert, S.: Dynamic lambda calculus. In: Proceedings of 5th Conference on the Mathematics of Language, Schloß Dagstuhl (1997)
13. Kohlhase, M., Kuschert, S., Pinkal, M.: A type-theoretic semantics for λ-DRT. In: Dekker, P., Strokhof, M. (eds.) Proceedings of 10th Amsterdam Colloquium, pp. 479–498 (1996)
14. Krahmer, E., Piwek, P.: Presupposition projection as proof construction. In: Bunt, H., Muskens, R. (eds.) Computing Meanings: Current Issues in Computational Semantics. Kluwer, Dordrecht (1999)
15. Kuschert, S.: Eine Erweiterung des λ-Kalküls um Diskursrepräsentationsstrukturen. Master's thesis, Universität Saarbrücken (1995)
16. Leiß, H.: Resolving plural ambiguities by type reconstruction. In: Groote, P., Nederhof, M.-J. (eds.) FG 2010-2011. LNCS, vol. 7395, pp. 267–286. Springer, Heidelberg (2012). doi:10.1007/978-3-642-32024-8_18
17. Mineshima, K.: A presuppositional analysis of definite descriptions in proof theory. In: Satoh, K., Inokuchi, A., Nagao, K., Kawamura, T. (eds.) JSAI 2007. LNCS (LNAI), vol. 4914, pp. 214–227. Springer, Heidelberg (2008). doi:10.1007/978-3-540-78197-4_20
18. Montague, R.: The proper treatment of quantification in ordinary English (chapter 8). In: Thomason, R. (ed.) Formal Philosophy, pp. 247–270. Yale University Press, New Haven (1974)
19. Muskens, R.: Combining montague semantics and discourse representation. Linguist. Philos. **19**, 143–186 (1996)
20. Ranta, A.: Type-Theoretical Grammar. Clarendon Press, Oxford (1994)
21. Ruhrberg, P.: Simultaneous abstraction and semantic theories. Ph.D. thesis, University of Edinburgh (1996)
22. Wu, S.: Getypte Lambda-Diskursrepräsentationsstrukturen - Typrekonstruktion für die λ-Diskursrepräsentationstheorie. Master's thesis, Centrum für Informations- und Sprachverarbeitung, Universität München (2012)