Chapter 21

Underspecified dependent type theory

In natural language, the meaning of a constituent may depend on the meaning of other constituents, typical examples of which are anaphora and presupposition. The behaviour of these phenomena is considered a *dynamics* of natural language and has been the subject of long-standing research in formal semantics since Karttunen (1976). However, it has gradually become clear in subsequent research that such a dynamic phenomenon is difficult to explain simultaneously with *compositionality*, in which the meaning of the whole is constructed from the meaning of its parts, and the tension between the dynamics and compositionality has always remained a challenge in formal semantics[*1]. This is one of the key aspects in exploration of meaning theories of natural language.

In this chapter, we introduce *underspecified types*, a concept in DTS that provides the basis for the analysis of dynamics in DTS, and distinguishes DTS from other semantic theories that use dependent types. We will go on to argue that underspecified types are the only device that can account for the dynamics of natural language, and since underspecified types require dependent types, a meaning theory of natural language can only be developed by a theory of dependent types.

21.1 Underspecified types

The underlying type theory adopted in DTS is underspecified dependent type theory (UDTT), which extends Martin-Löf type theory (or dependent type theory; DTT) with *underspecified types* (**?**). Underspecified terms are the key devices by which DTS represents the meanings of pronouns and presupposition triggers while keeping the semantic theory compositional. Syntactically, underspecified types takes the form

(499) $(x@A) \times B$

where x is a variable, A is a checkable type A, and B is an inferable type. There is alternative notation for underspecified types, as shown below, which is called a *vertical form*.

(500) $\left[\begin{array}{l} x@A \\ B \end{array}\right]$

[*1] See e.g. the discussion in Yana et al. (2019).

Intuitively, a variable x is an *open proof* of type A, which behaves within B (if x occurs free in B) as a term of type A without any actual proof, assuming the existence of a proof of A. [*2]

UDTT is characterised by the existence of underspecified types and the operation *@-elimination*, which is a proof transformation from proof diagrams of UDTT to proof diagrams of DTT. An underspecified type requires in its formation rule that there exists a proof term for type A. Since there is generally more than one proof for type A, there is also generally more than one proof diagram that verifies the formation rule of a given underspecified type. In other words, a proof diagram for UDTT corresponds to choosing one of several options to specify underspecification.

@-elimination converts a UDTT proof diagram into a DTT proof diagram without underspecified types. This operation is guaranteed (by Theorem 439), and therefore UDTT is a conservative extention of DTT, which will be proved as Corollary 441 at the end of this chapter.

Typing rules and @-elimination in UDTT are defined by simultaneous recursion. Let us look at them in the following order.

21.2 Syntax

A collection of preterms of UDTT contains underspecified terms of the form $(x@A)\times B$.

Definition 421 (Preterms of UDTT)**.** A collection of *preterms* (notation Λ) under $(\mathcal{V}ar, \mathcal{C}on)$ is recursively defined by the following BNF grammar (where $x \in \mathcal{V}ar$ and $c \in \mathcal{C}on$):

$$
\begin{array}{rcl}
\Lambda & ::= & x \mid c \mid \mathsf{type} \mid \mathsf{kind} \\
& & \mid (x:\Lambda) \to \Lambda \mid \lambda x.\Lambda \mid \Lambda\Lambda \\
& & \mid (x:\Lambda) \times \Lambda \mid (\Lambda,\Lambda) \mid \pi_1(\Lambda) \mid \pi_2(\Lambda) \\
& & \mid \Lambda + \Lambda \mid \iota_1(\Lambda) \mid \iota_2(\Lambda) \mid \mathsf{unpack}^{\Lambda}_{\Lambda}(\Lambda,\Lambda) \\
& & \mid \bot \mid \top \mid () \mid \mathsf{e} \mid \mathit{john} \mid \mathit{mary} \mid \ldots \mid \mathsf{case}^{A}_{A}(\Lambda,\ldots,\Lambda) \\
& & \mid \Lambda =_{\Lambda} \Lambda \mid \mathsf{refl}_{\Lambda}(\Lambda) \mid \mathsf{idpeel}^{\Lambda}_{\Lambda}(\Lambda) \\
& & \mid \mathbb{N} \mid 0 \mid \mathsf{s}(\Lambda) \mid \mathsf{natrec}^{\Lambda}_{\Lambda}(\Lambda,\Lambda) \\
& & \mid (x@\Lambda)\times\Lambda
\end{array}
$$

Free variables and substitution are defined by the following set of rules.

Definition 422 (Free variables in UDTT)**.**

$$fv((x@A)\times B) \stackrel{def}{\equiv} fv(A) \cup (fv(B) - \{x\})$$

[*2] Let us take an example in a mathematical context: When talking about a function $f : A \to B$, a proposition such as "a is mapped to b by the inverse function f^{-1}" actually presupposes that f is a bijection, and if this is not satisfied, the whole proposition is meaningless. This kind of proposition cannot be represented by FoL, while UDTT can do it, by using an underspecified type $(x@a)\times B$, setting A as a type for the proposition "f is a bijection" and x its proof, without giving the actual proof, while B is a type for the proposition "a is mapped to b by f^{-1}", in which f^{-1} is costructed by using x. This proposition makes no sense if f is not a bijection, since the formation rule of the underspecified type requires the proof diagram of A in the upper part.

Definition 423 (Substitution in UDTT)**.**

$$((x@A) \times B)[L/x] \stackrel{def}{\equiv} x@A[L/x] \times B$$
$$((y@A) \times B)[L/x] \stackrel{def}{\equiv} y@A[L/x] \times B[L/x] \quad \textit{where } x \notin fv(B) \textit{ or } y \notin fv(L).$$

Definition 424 (Judgment)**.** Let σ be a DTT signature, Γ a DTT context, M a UDTT preterm, and A a DTT preterm, then the following form is a judgment in UDTT.

$$\Gamma \vdash_\sigma M : A$$

Note here that a UDTT judgment (except M) consists of elements of DTT. The type assignment of UDTT is of the form $M : A$ by a UDTT preterm M and a DTT preterm A, and all typing rules of UDTT are also defined based on the type assignment of UDTT.

21.3 Typing Rules

In UDTT, the typing rules and @-elimination $[\![\_]\!]$, which is a proof transformations, are defined by the simultaneous recursion. The @-Elimination $[\![\_]\!]$ is an operation that converts a UDTT diagram into a DTT diagram. In other words, it is an operation that removes the @-rule from the UDTT diagram and replaces it with the DTT diagram. [*3]

Typing rules and @-elimination in UDTT are defined by simultaneous recursion with respect to preterm depth d. Namely,

1) Typing rules and @-elimination are defined for preterms of depth 1
2) Define typing rule and @-elimination rule for preterms of depth d, assuming that typing rule and @-elimination have been defined for preterms of depth less than d.

Definition 425 ((CON)-rule and @-elimination)**.**

$$\frac{}{c : A}\,(CON) \quad \textit{where } c : A \in \sigma.$$

$$\left[\!\!\left[\frac{}{c : A}\,(CON) \right]\!\!\right] = \frac{}{c : A}\,(CON)$$

[*3] @-elimination is also closely related to open logic or type checking in the presence of meta-variable. See Geuvers and Joigov (2002), Norell (2007), among others.

Definition 426 ((type$F$)-rule and @-elimination)**.**

$$\frac{}{\text{type} : \text{kind}}\,{}^{(\text{type}F)}$$

$$\left[\!\!\left[\frac{}{\text{type} : \text{kind}}\,{}^{(\text{type}F)}\right]\!\!\right] = \frac{}{\text{type} : \text{kind}}\,{}^{(\text{type}F)}$$

The (CON)-rule and (type F)-rule of the UDTT are identical to those of DTT and are not changed by @-elimination.

Definition 427 ((ΠF)-rule and @-elimination)**.**

$$\frac{\begin{array}{c}\overline{x : A'}^{\,i}\\ \vdots\\ \mathcal{D}_A \qquad B : \mathrm{s}_2\\ A : \mathrm{s}_1\end{array}}{(x : A) \to B : \mathrm{s}_2}\,{}^{(\Pi F),i} \quad \textit{where} \quad \left[\!\!\left[\begin{array}{c}\mathcal{D}_A\\ A : s_1\end{array}\right]\!\!\right] = \begin{array}{c}[\![\mathcal{D}_A]\!]\\ A' : s_1\end{array}\ ,\ (\mathrm{s}_1, \mathrm{s}_2) \in \left\{\begin{array}{l}(\text{type}, \text{type}),\\ (\text{type}, \text{kind}),\\ (\text{kind}, \text{kind})\end{array}\right\}.$$

$$\left[\!\!\left[\frac{\begin{array}{cc}\overline{x : A'}^{\,i}\\ \mathcal{D}_A \quad \mathcal{D}_B\\ A : \mathrm{s}_1 \quad B : \mathrm{s}_2\end{array}}{(x : A) \to B : \mathrm{s}_2}\,{}^{(\Pi F),i}\right]\!\!\right] = \frac{\begin{array}{cc}\overline{x : A'}^{\,i}\\ [\![\mathcal{D}_A]\!] \quad [\![\mathcal{D}_B]\!]\\ A' : \mathrm{s}_1 \quad B' : \mathrm{s}_2\end{array}}{(x : A') \to B' : \mathrm{s}_2}\,{}^{(\Pi F),i}$$

Definition 428 ((ΠI)-rule and @-elimination)**.**

$$\frac{\begin{array}{c}\overline{x : A'}^{\,i}\\ \vdots\\ \mathcal{D}_A \qquad M : B\\ A : \mathrm{s}\end{array}}{\lambda x.M : (x : A) \to B}\,{}^{(\Pi I),i} \quad \textit{where} \quad \left[\!\!\left[\begin{array}{c}\mathcal{D}_A\\ A : s\end{array}\right]\!\!\right] = \begin{array}{c}[\![\mathcal{D}_A]\!]\\ A' : s\end{array}\ ,\ \mathrm{s} \in \{\text{type}, \text{kind}\}\,.$$

$$\left[\!\!\left[\frac{\begin{array}{cc}\overline{x : A'}^{\,i}\\ \mathcal{D}_1 \quad \mathcal{D}_2\\ A : \mathrm{s}_1 \quad M : B\end{array}}{\lambda x.M : (x : A) \to B}\,{}^{(\Pi I),i}\right]\!\!\right] = \frac{\begin{array}{cc}\overline{x : A'}^{\,i}\\ [\![\mathcal{D}_1]\!] \quad [\![\mathcal{D}_2]\!]\\ A' : \mathrm{s}_1 \quad M' : B'\end{array}}{\lambda x.M' : (x : A') \to B'}\,{}^{(\Pi I),i}$$

Definition 429 ((ΠE)-rule and @-elimination)**.**

$$\frac{\begin{array}{cc}\mathcal{D}_M\\ M : (x : A) \to B \quad N : A'\end{array}}{MN : B}\,{}^{(\Pi E)} \quad \textit{where} \quad \left[\!\!\left[\begin{array}{c}\mathcal{D}_M\\ M : (x : A) \to B\end{array}\right]\!\!\right] = \begin{array}{c}[\![\mathcal{D}_M]\!]\\ M' : (x : A') \to B'\end{array}\ .$$

$$\left[\!\!\left[\frac{\begin{array}{cc}\mathcal{D}_M \quad \mathcal{D}_N\\ M : (x : A) \to B \quad N : A'\end{array}}{MN : B}\,{}^{(\Pi E)}\right]\!\!\right] = \frac{\begin{array}{cc}[\![\mathcal{D}_M]\!] \quad [\![\mathcal{D}_N]\!]\\ M' : (x : A') \to B' \quad N' : A'\end{array}}{M'N' : B'[N'/x]}\,{}^{(\Pi E)}$$

Definition 430 ((ΣF)-rule and @-elimination)**.**

$$\frac{\begin{array}{cc} & \overline{x : A'}^{\,i} \\ \mathcal{D}_A & \vdots \\ A : \text{type} & B : \text{type} \end{array}}{\begin{bmatrix} x : A \\ B \end{bmatrix} : \text{type}}\,(\Sigma F),i \qquad \textit{where} \quad \left[\!\!\left[\begin{matrix} \mathcal{D}_A \\ A : \text{type} \end{matrix} \right]\!\!\right] = \begin{matrix} [\![\mathcal{D}_A]\!] \\ A' : \text{type} \end{matrix} .$$

$$\left[\!\!\left[\frac{\begin{array}{cc} & \overline{x : A'}^{\,i} \\ \mathcal{D}_A & \mathcal{D}_B \\ A : \text{type} & B : \text{type} \end{array}}{\begin{bmatrix} x : A \\ B \end{bmatrix} : \text{type}}\,(\Sigma F),i \right]\!\!\right] = \frac{\begin{array}{cc} & \overline{x : A'}^{\,i} \\ [\![\mathcal{D}_A]\!] & [\![\mathcal{D}_B]\!] \\ A' : \text{type} & B' : \text{type} \end{array}}{\begin{bmatrix} x : A' \\ B' \end{bmatrix} : \text{type}}\,(\Sigma F),i$$

Definition 431 ((ΣI)-rule and @-elimination)**.**

$$\frac{\begin{array}{cc} \mathcal{D}_M & \\ M : A & N : B[M'/x] \end{array}}{(M, N) : \begin{bmatrix} x : A \\ B \end{bmatrix}}\,(\Sigma I) \qquad \textit{where} \quad \left[\!\!\left[\begin{matrix} \mathcal{D}_M \\ M : A \end{matrix} \right]\!\!\right] = \begin{matrix} [\![\mathcal{D}_M]\!] \\ M' : A' \end{matrix} .$$

$$\left[\!\!\left[\frac{\begin{array}{cc} \mathcal{D}_M & \mathcal{D}_N \\ M : A & N : B[M'/x] \end{array}}{(M, N) : \begin{bmatrix} x : A \\ B \end{bmatrix}}\,(\Sigma I) \right]\!\!\right] = \frac{\begin{array}{cc} [\![\mathcal{D}_M]\!] & [\![\mathcal{D}_N]\!] \\ M' : A' & N' : B'[M'/x] \end{array}}{(M', N') : \begin{bmatrix} x : A' \\ B' \end{bmatrix}}\,(\Sigma I)$$

Definition 432 ((ΣE)-rule and @-elimination)**.**

$$\frac{M : \begin{bmatrix} x : A \\ B \end{bmatrix}}{\pi_1(M) : A}\,(\Sigma E) \qquad \frac{M : \begin{bmatrix} x : A \\ B \end{bmatrix}}{\pi_2(M) : B[\pi_1(M)/x]}\,(\Sigma E)$$

$$\left[\!\!\left[\frac{M : \begin{bmatrix} x : A \\ B \end{bmatrix}}{\pi_1(M) : A}\,(\Sigma E) \right]\!\!\right] = \frac{[\![\mathcal{D}_M]\!]}{M' : \begin{bmatrix} x : A' \\ B' \end{bmatrix}}\,(\Sigma E)$$

$$\left[\!\!\left[\frac{M : \begin{bmatrix} \mathcal{D}_M \\ x : A \\ B \end{bmatrix}}{\pi_2(M) : B'[\pi_1(M)/x]}\,(\Sigma E) \right]\!\!\right] = \frac{[\![\mathcal{D}_M]\!]}{M' : \begin{bmatrix} x : A' \\ B' \end{bmatrix}}\,(\Sigma E)$$

Remark 433. UDTT typing rules consist of UDTT type assignments (in the form $M : A$), but if there is no underspecified type in UDTT preterm M, the @-elimination of M returns the corresponding DTT preterm.allows the DTT's corresponding typing rule to The @-elimination is defined to match the corresponding typing rule of the

DTT if there is no underspecified type in M. On the other hand, if there is an underspecified type in M, the underspecified term appears somewhere on the left-hand side of the type assignment when going back through the proof diagram, and the following @-rule is applied.

Definition 434 ((@)-rule and @-elimination)**.**

$$\frac{\begin{array}{ccc}\mathcal{D}_A & & \\ A : \text{type} & M : A' & B[M/x] : \text{type}\end{array}}{\left[\begin{array}{c} x@A \\ B \end{array}\right] : \text{type}}\,(@) \quad \textit{where} \quad \left[\!\!\left[\begin{array}{c}\mathcal{D}_A \\ A : \text{type}\end{array}\right]\!\!\right] = \begin{array}{c}[\![\mathcal{D}_A]\!] \\ A' : \text{type}\end{array}.$$

$$\left[\!\!\left[\frac{\begin{array}{ccc}\mathcal{D}_A & \mathcal{D}_M & \mathcal{D}_B \\ A : \text{type} & M : A' & B[M/x] : \text{type}\end{array}}{\left[\begin{array}{c} x@A \\ B \end{array}\right] : \text{type}}\,(@)\right]\!\!\right] = \begin{array}{c}[\![\mathcal{D}_B]\!] \\ B'[M/x] : \text{type}\end{array}$$

The @-rule in Definition 434 requires thhat A is a type, and let A' be the "@-eliminated" version (that is, the DTT version) of A, and A' inhabits some proof M, and $B[M/x]$ is a type.

The @-elimination substitutes the variable x in B with a found proof M, thereby ensuring that the proof M be constructed outside B. Note that the position at which the proof search takes place reflects the syntactic position in the structural hierarchy of the lexical item that triggers $@A$.

The @-rule @-Elimination is a function that transforms a proof diagram of UDTT to a proof diagram of DTT. The basic idea is twofold:

- Replace underspecified terms by proof terms obtained by (@)rule upper proof search.
- For any other rule, call @-Elimination recursively. First, for 1.

The upper part of the @-Elimination rule includes the proof search (in the diagram above, the search for the proof diagram $\mathcal{D}_2$ of DTT for type A'). The @-Elimination is replaced for underspecified terms by the proof terms found there. It is defined as the process of erasing the underspecified term x in A from the inside (i.e. if there is an underspecified type in A, it is erased first).

It is defined as the process of erasing the underspecified term x in A from the inside (i.e. if there is an underspecified type in A, it is erased first). This recursion allows anaphora resolution to be correctly performed from the inside even in cases where the presupposition contains other presuppositions (e.g. *If John has a wife, his wife is happy.*). The anaphora resolution is also performed correctly from the inside. It is also ensured (importantly) that the proof search invoked in an individual anaphora resolution is a DTT proof search and not a UDTT one.

we introduce some notation for describing the rules.

Definition 435. If D is a proof diagram for a judgment $M : A$, then $D^{\text{tm}} \stackrel{def}{\equiv} M$ and $\text{ty}(D) \stackrel{def}{\equiv} A$.

21.4 Reduction

The UDTT's reduction rules are the DTT's reduction rules with the following structure rules (for underspecified types). Since underspecified types do not have introduction/elimination rules, beta redex is not defined.

Definition 436 (One-step β-reduction for UDTT)**.**

Structure rule

$$\frac{A \to_\beta A'}{\left[\begin{array}{l} x@A \\ B \end{array}\right] \to_\beta \left[\begin{array}{l} x@A' \\ B \end{array}\right]} \qquad \frac{B \to_\beta B'}{\left[\begin{array}{l} x@A \\ B \end{array}\right] \to_\beta \left[\begin{array}{l} x@A \\ B' \end{array}\right]}$$

Remark 437. Multi-step β-reduction, denoted as $\twoheadrightarrow_\beta$, is defined in the same way as in DTT, which is a reflexive and transitive binary relation between preterms. β-equivalence $=_\beta$ is also defined in the same way as in DTT, which is an equivalence relation.

21.5 Theorems on @-elimination

Theorem 438. For any proof diagram of DTT is a proof diagram of UDTT.

Proof. Straightforward from Theorem 439 and Theorem 440. □

Theorem 439. For any proof diagram $\mathcal{D}$ of UDTT, $[\![\mathcal{D}]\!]$ is a proof diagram of DTT.

Proof. By induction on the depth of $\mathcal{D}$. □

Theorem 440. If a proof diagram $\mathcal{D}$ of UDTT does not contain any (@)-rule, then $[\![\mathcal{D}]\!]{=}\mathcal{D}$.

Proof. By induction on the depth of $\mathcal{D}$. □

Corollary 441. UDTT is a conservative extention of DTT.

History and Further readings

There have been different versions of UDTT: Bekki and Sato (2015) is a preliminary study on the formulation and the implementation of UDTT, akin to the current

formulation of UDTT where UDTT and DTT are different formal systems that are connected with proof transformation (which we call @-elimination). Bekki (2023) does not treat UDTT as a separate language: in Bekki (2023), @-operator is defined as a *control operator* on the type checking procedure of DTT.