Chapter 21

Underspecified dependent type theory

In the intricate landscape of natural language, the interpretation of a constituent is frequently contingent upon the meanings of other elements within its context; anaphora and presupposition serve as salient illustrations of this dependency. The behavior of such phenomena has been conceptualized as the *dynamics* of natural language, a subject that has captivated formal semantics researchers since the pioneering work of Karttunen (1976).

However, subsequent scholarly inquiry has progressively illuminated a profound challenge: reconciling this dynamic aspect with the principle of compositionality, which posits that the meaning of a complex expression is systematically derived from the meanings of its parts and their mode of combination. This inherent tension between dynamics and compositionality has persistently remained a central predicament in formal semantics, as elucidated, for instance, by Yana et al. (2019). This very tension represents one of the pivotal facets in the ongoing exploration of meaning theories for natural language.

This chapter introduces the concept of *underspecified types* within DTS. This notion is foundational for analyzing dynamic phenomena in natural language and critically distinguishes DTS from other semantic theories that employ dependent types. We contend that underspecified types represent the sole mechanism capable of accounting for the inherent dynamism observed in natural language. Consequently, given that underspecified types necessitate the expressive power of dependent types, a comprehensive theory of natural language meaning can only be adequately developed within a dependent type-theoretic framework.

21.1 Underspecified types

The underlying type theory adopted in DTS is Underspecified Dependent Type Theory (UDTT) **?**. This framework extends DTT with underspecified types, providing a crucial mechanism for representing the meanings of phenomena such as anaphora and presupposition while preserving semantic compositionality. Syntactically, an underspecified type takes the form

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

where A, B are types, with x acting as a variable. This can also be rendered in a

vertical form:

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

Intuitively, the variable x above functions as an *open proof* of type A. Within the context of B (should x occur freely therein), x behaves as a term of type A without requiring an actual proof, assuming the existence of such a proof for A.[*1]

UDTT is characterized by two core features: the existence of underspecified types, as described above, and the operation of *@-elimination*. The latter is a proof transformation mechanism designed to convert proof diagrams from UDTT into proof diagrams of standard DTT, thereby eliminating underspecified types.

The formation rule for an underspecified type stipulates the existence of a proof term for type A. Given that there generally exists more than one proof for any given type A, it follows that there can be multiple proof diagrams that satisfy the formation rule of a specific underspecified type. This implies that a UDTT proof diagram corresponds to a choice among various options for specifying underspecification.

Crucially, @-elimination is guaranteed to convert any UDTT proof diagram into a well-formed DTT proof diagram devoid of underspecified types (cf. Theorem 439). This guarantee establishes UDTT as a conservative extension of DTT, a proposition rigorously demonstrated as Corollary 441 at the conclusion of this chapter. The typing rules and @-elimination in UDTT are meticulously defined via simultaneous recursion, which we shall now proceed to examine in detail.

21.2 Syntax

The collection of preterms of UDTT extends that of DTT through the inclusion of underspecified types of the form $(x@A) \times B$. The full collection of UDTT preterms is thus obtained by incorporating these constructs into the standard DTT syntactic formations.

[*1] To elucidate, consider a mathematical proposition concerning a function $f : A \to B$: "a is mapped to b by the inverse function f^{-1}. This assertion inherently presupposes that f is a bijection; absent this condition, the entire proposition is rendered meaningless. Such presuppositional content cannot be adequately formalized within first-order logic (FoL), yet UDTT readily accommodates them.

UDTT achieves this by employing an underspecified type $(x@A) \times B$. Here, A represents the proposition "f is a bijection," and x is its underspecified proof. B then represents the proposition "a is mapped to b by f^{-1}," where f^{-1} is constructed using x. The formation rule for this underspecified type explicitly demands the existence of a proof diagram for A in its "upper part," ensuring that the proposition remains ill-formed if f is not, in fact, a bijection.

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 $c \in \mathcal{C}on$):

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

The notions of free variables and substitution are defined through a conventional set of rules tailored to this expanded syntax.

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

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

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

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

Judgments within UDTT exhibit a hybrid nature, comprising elements drawn from both UDTT and DTT.

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

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

It is paramount to observe that, with the sole exception of the UDTT preterm M, every component of a UDTT judgment is an element of DTT. The type assignment in UDTT, expressed as $M : A$, pairs a UDTT preterm M with a DTT preterm A. All typing rules of UDTT are subsequently predicated upon this specific form of type assignment.

21.3 Typing Rules

In UDTT, the typing rules and the @-elimination operation ($[\![\_]\!]$) are defined through a simultaneous recursion. The operation $[\![\_]\!]$ itself functions as a proof transformation, systematically converting a UDTT diagram into its corresponding DTT diagram. Conceptually, this entails the removal of @-rules from the UDTT diagram and their

replacement with equivalent DTT constructs.[*2]

The simultaneous recursive definition of typing rules and @-elimination in UDTT proceeds with respect to the depth d of a UDTT proof diagram. This recursive schema can be articulated as follows:

Base case 1: Typing rules for depth 1 UDTT proof diagrams, and @-elimination for depth 1 UDTT proof diagrams are defined.

Inductive step: Assuming that typing rules and @-elimination have been successfully defined for all UDTT proof diagram of depth less than d, typing rules and @-elimination are then defined for UDTT proof diagrams of depth d.

Our exposition starts with the definition of typing rules and the associated @-elimination operation for depth-1 proof diagrams within UDTT.

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

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

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

Definition 426 ($(\mathsf{type}F)$-rule and @-elimination)**.**

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

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

Note that specific rules within UDTT, such as the (CON)-rule and the (type F)-rule, are precisely congruent with their counterparts in standard DTT. Consequently, the operation of @-elimination exerts no effect upon these invariant rules.

Following this definition for depth-1 diagrams, we proceed to the inductive step. This involves extending the definition of typing rules and the @-elimination procedure to encompass UDTT proof diagrams of depths exceeding unity.

[*2] The notion of @-elimination is closely related with concepts in *open logic* and type checking that involve meta-variables. For a more exploration of these connections, one may consult, inter alia, Geuvers and Joigov (2002) and Norell (2007).

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

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

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

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

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

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

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

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

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

In UDTT, the standard rules governing Π-types undergo a crucial redefinition, replaced by the rules specified above. A salient point of divergence resides in the (ΠF)-rule, specifically between its upper-left and upper-right premises. In the former, the conclusion of $\mathcal{D}_A$ is stated as $A : s_1$, while the latter assumes $x : A'$, where A' might not simply be A but rather the preterm derived from applying @-elimination to the proof diagram $\mathcal{D}_A$, as stipulated in the proviso.

This distinction is necessitated by the potential for A and A' to diverge when A incorporates an underspecified type. The @-elimination operation, in such instances, resolves the underspecification, yielding a potentially distinct type A'. An analogous situation obtains with the (ΠI) rule, where a similar mechanism addresses the

implications of underspecification.

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

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

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

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

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

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

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

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

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

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

The behavior observed in the application of Π-rules finds a direct parallel in the Σ-rules of DTT. Consider, for instance, the premise posited in the upper right as-

sumption of the (ΣF)-rule: the assumption $x : A'$. This type A' is derived from $\mathcal{D}_A$ through the application of an @-elimination operation, a mechanism precisely mirroring that found in the (ΠF)-rule.

A crucial subtlety arises when the type A inherently contains underspecified components; in such instances, the derived type A' may indeed diverge from A. This same principle extends to the (ΣI)-rule, where the terms M and N may yield distinct M' and N', if they initially contain underspecified types. An analogous phenomenon is observed with the term M in the (ΣE)-rule.

Remark 433. The typing rules within UDTT are expressed as UDTT type assignments of the form $M : A$. A crucial property of UDTT is its relationship to a fully specified DTT via @-elimination. Specifically, if the UDTT preterm M contains no underspecified types, its @-elimination directly yields the corresponding DTT preterm, thereby aligning with the established typing rules of DTT.

Conversely, should M contain an underspecified type, its presence will manifest on the left-hand side of a type assignment when tracing back through the proof derivation. In such instances, a distinct @-rule is invoked to govern the typing process, addressing the inherent underspecification.

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

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

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

The @-rule, as defined in Definition 434, mandates that A be a well-formed type. Let A' denote the @-eliminated (i.e., DTT equivalent) version of A. Furthermore, let M be a proof inhabiting A', such that $B[M/x]$ is also a well-formed type. The overarching function of evaluating $\Gamma \vdash x@A : ?$ is to yield a set of proof diagrams for $\Gamma \vdash M : A'$, for all such valid M and A'.

The @-elimination process involves the substitution of the variable x in B with an empirically determined proof term M. This substitution ensures that the construction of proof M occurs externally to B. It is crucial to note that the locus of this proof search directly correlates with the syntactic position within the structural hierarchy of the lexical item triggering the $x@A$ construct.

The @-elimination rule serves as a mapping, transforming a proof diagram within UDTT into a corresponding proof diagram in DTT. This transformation is predicated on a two-fold strategy:

Replacement of Underspecified Terms: Underspecified terms are systematically replaced by concrete proof terms, which are themselves the product of an upward proof search triggered by the @-rule.

Recursive Application: For all other rules within the system, @-elimination is re-

cursively invoked.

Focusing on the first point, the upper segment of the @-elimination rule incorporates the necessary proof search—specifically, the search for a DTT proof diagram $\mathcal{D}_2$ for type A'. The underspecified terms are then substituted by the proof terms identified during this search. This process is formally defined as the recursive elimination of the underspecified term x within A, proceeding from the innermost elements outwards (i.e., any underspecified types within A are resolved first).

This recursive mechanism is pivotal, enabling the correct resolution of anaphora from the inside out, even in complex scenarios where presuppositions are embedded within other presuppositions (e.g., *If John has a wife, his wife is happy.*). Critically, it is guaranteed that the proof search invoked during an individual anaphora resolution is a DTT proof search, not a UDTT one, ensuring type soundness and consistency.

For clarity in describing these rules, we introduce the following notation:

Definition 435. Given a proof diagram D for a judgment $M : A$, we define $D^{\mathrm{tm}} \stackrel{def}{\equiv} M$ and $\mathrm{ty}(D) \stackrel{def}{\equiv} A$.

21.4 Reduction

The reduction calculus of UDTT extends that of standard DTT by incorporating a specific set of structure rules tailored for underspecified types. A crucial distinction arises from the nature of these underspecified types: lacking explicit introduction and elimination rules, they inherently preclude the definition of β-redexes.

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. Notwithstanding this distinction, the fundamental notions of reduction and equivalence are preserved. Multi-step β-reduction, denoted $\twoheadrightarrow_\beta$, is defined identically to its DTT counterpart as a reflexive and transitive binary relation over preterms. Consequently, β-equivalence, denoted $=_\beta$, retains its character as an equivalence relation, mirroring its definition in DTT.

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

The evolution of UDTT has witnessed distinct foundational approaches. In its nascent formulation, as presented by Bekki and Bekki and Sato (2015), UDTT emerged as a formal system conceptually separate from standard DTT. The connection between these systems was established through a defined proof transformation, termed @-elimination. This preliminary work laid the groundwork for a framework where UDTT and DTT coexist as distinct, yet interrelated, logical calculi.

In a subsequent and contrasting development, Bekki (2023) posits a reinterpretation wherein UDTT is not conceived as a discrete language. Instead, the @-operator is redefined as a control operator directly influencing the type-checking procedure within DTT itself. This shift integrates the underspecification mechanism more intimately with the core mechanics of DTT, rather than positing a parallel formal system.

This divergence in conceptualization prompts an inquiry, which remains as an open issue: What are the logical and computational implications of framing underspecification as a distinct formal system versus an internal control mechanism within an existing type theory?