Learning Knowledge with Neural DTS

Daisuke Bekki and **Ribeka Tanaka** and **Yuta Takahashi**
Ochanomizu University
{bekki|tanaka.ribeka|takahashi.yuta}@is.ocha.ac.jp

Abstract

Neural DTS is a framework that combines logic and machine learning by fusing dependent type semantics (DTS) and deep neural networks. In this paper, we propose a learning algorithm for Neural DTS, in which a collection of semantic representations of DTS is obtained from text through syntactic parsing and semantic composition, and the collection of positive predicates is generated through the deduction of DTS. We will also discuss the advantages of this method and the challenges it shall face.

1 Introduction

In the field of natural language understanding (NLU), type-logical semantics and neural language processing exhibit complementary benefits: The former has advantages in the systemic prediction of complex linguistic phenomena such as negation, conditional, quantification, anaphora, presupposition, and modality as they appear in various syntactic structures including embeddings, while the latter has a wide range of applications including similarity calculations, summarization, translation, dialogue processing, and even multimodal inference, as well as being robust and able to quickly process real texts. In recent years, hybrid methods that utilizes both of the techniques have been explored in neighboring fields of NLU, and methods adopted in previous studies can be roughly classified as follows:

1. Emulating symbolic reasoning by embedding into neural networks:
    - knowledge graphs (Guu et al., 2015)(Das et al., 2017)(Takahashi et al., 2018)
    - SAT problems (Selsam et al., 2019)
    - first-order logic (FoL) (Demeester et al., 2016)(Šourek et al., 2018)
2. Introducing a similarity measure between symbols by using distributional representations instead of symbols (Lewis and Steedman, 2013)(Rocktäschel and Riedel, 2017).
3. Controlling the direction of proof search guided by neural networks (Wang et al., 2017).

One method that differs from these approaches is Neural DTS (Bekki et al., 2021). Compared with previous studies, Neural DTS is unique in its use of DTS (Bekki and Mineshima, 2017), a higher-order type-logical semantic framework. Since DTS is based on Martin-Löf type theory (MLTT; Martin-Löf (1984)), a framework for intuitionistic mathematics, it allows for the construction of the real numbers necessary for developping deep neural networks.[1] Neural DTS replaces all of the predicates of DTS with neural classifiers, which provide the soft symbols that are required for soft reasoning. Nevertheless, the entire system is still within the framework of MLTT, and all of the components of the neural networks, such as loss functions, have proof terms.

In this paper, we propose a learning algorithm for Neural DTS that fits the parameters for the names and predicates of Neural DTS to the data generated by the proof system of DTS from the propositions obtained from the real texts. Using this algorithm, we investigate the claim of Bekki et al. (2021) that the symbols (names and predicates) of Neural DTS are learnable, one of the criteria for soft reasoning systems.

1.1 Dependent Type Semantics (DTS)

DTS is a framework for the proof-theoretic semantics of natural language. Unlike similar proof-theoretic attempts[2], DTS takes a verificationist approach, in which the meaning of a proposition in a given context is a collection of proofs of that proposition in that context. This position shows sharp

[1] See Appendix A.4 for details.

[2] cf. Francez and Dyckhoff (2010); Francez (2014).

contrasts to most of the standard model-theoretic semantics, but it provides a more fine-grained notion of meaning than the model-theoretic approaches in the following sense.

1. In model-theoretic semantics, all tautologies have the same meaning (i.e., the set consisting of all models), but in DTS, tautologies with different proofs have different meanings. The same is true for contradictions.

2. There are pairs of sentences that are indistinguishable in their truth conditions but which have different anaphoric potentials. For example (cf. Kamp et al. (2011)):

   (1) a. Some student did not show up. They must have overslept.

   b. Not every student showed up. *They must have overslept.

   To explain this contrast, DRT uses an extra layer called Discourse Representation Structure (DRS), where the difference in the anaphoric potential is explained via the notion of accessibility defined on DRS. In DTS, on the other hand, the difference is explained by the constructivity of the proofs, without introducing another layer to the semantic theory (Bekki, 2014). It has been pointed out that the extra-representational layer in DRT complicates the compositionality problem (Yana et al., 2019), while compositionality is preserved in DTS.

With these features, DTS opens up the analyses of many linguistic phenomena, some of which were previously unexplained by model-theoretic semantics, based on a proof-theoretic perspective where one can refer to a proof as an object.[3] There have also been studies on implementations of type checking (Bekki and Sato, 2015) and proof search (Daido and Bekki, 2020) in DTS, with the goal of developing a research program in which the predictions of formal semantics can be verified through implementation.

To supplement the DTS specifications that are relevant to the main purpose of this paper, first, the enumeration type[4] **entity** plays the role of type e in the standard semantics. Second, the role of type t (or type $prop$) in the standard semantics is played by the type type (namely, propositions, under the Curry-Howard isomorphism between types and propositions). Consequently, an n-place predicate has a type $\mathbf{entity}^n \to \mathsf{type}$. For example, **dog** is a unary predicate with type $\mathbf{entity} \to \mathsf{type}$ and the name $john$ has type **entity**. The $\mathbf{dog}(john)$ has type type, which is a collection of proofs that John is a dog. The truth condition in DTS tells us that $\mathbf{dog}(john)$ is true if and only if it is inhabited by at least one proof.

From the perspective of fusing symbolic and soft reasoning, however, this kind of specification by DTS shares a crucial property with the standard semantics, despite the differences between type e and type **entity**, and type t and type type: the symbols in DTS are neither comparable nor learnable when compared with the distributional representations that are often found in neural networks, which have become a trend in recent neural language processing technology. These properties are often considered to be shortcomings of symbolic reasoning but the advantages of soft reasoning.

1.2 Neural DTS

Bekki et al. (2021) proposed Neural DTS as a framework for combining DTS with some aspects of soft reasoning. Neural DTS is obtained by replacing n-place predicates in DTS (that is, constant symbols of type $\mathbf{entity}^n \to \mathsf{type}$) with neural classifiers, which are also DTS terms of type $\mathbf{entity}^n \to \mathsf{type}$. Since the descriptive power of DTS allows for the construction of real numbers and real functions (or complex numbers and complex functions, if necessary) internally through the notion of setoids[5], it also allows for the implementation of neural networks and neural classifiers in the following way.

Consider DTS with a given signature where:

1. the type **entity** has n-introduction rules (namely, **entity** has the form $\{a_1, \ldots, a_n\}$).

2. there are k-many unary predicates $P_1, \ldots, P_k$, each of which is of type $\mathbf{entity} \to \mathsf{type}$.

Let $\mathbb{ENT}$ be a setoid $(\mathbf{entity}, =_{\mathbf{entity}})$, and let onehot be a setoid function from $\mathbb{ENT}$ to $\mathbb{R}^n$

[3] For recent developments regarding the semantic analyses in DTS, see the discussions on the overwriting problem (Yana et al., 2019), generalized quantifiers (Tanaka, 2021), proviso problems (Yana et al., 2021), and weak crossover (Bekki, 2021).

[4] See Appendix A.2 for the definition of the enumeration type.

[5] See Appendix A.3 for the definition of setoid in DTS.

$$\Gamma,\ p_{\mathsf{emb\_ent}} \in \mathbb{R}^{n\times m},\ p_{\mathsf{emb\_pred}} \in \mathbb{R}^{k\times l},\ p_{\mathsf{hidden}} \in \mathbb{R}^{(m+l)\times o},\ p_{\mathsf{out}} \in \mathbb{R}^{o\times 1}$$
$$\vdash\ \mathsf{pred}(a_i, j) \stackrel{def}{\equiv} \mathsf{sigmoid}(W_{\mathsf{out}}(\mathsf{sigmoid}(W_{\mathsf{hidden}}(W_{\mathsf{emb\_ent}}(\mathsf{onehot}(a_i)) \oplus W_{\mathsf{emb\_pred}}(e_j))))) \in \mathbb{R}$$

Figure 1: Neural predicate in DTS

$$\Gamma,\ p_{\mathsf{emb\_ent}} \in \mathbb{R}^{n\times m},\ p_{\mathsf{emb\_pred}} \in \mathbb{R}^{k\times l},\ p_{\mathsf{hidden}} \in \mathbb{R}^{(m+l)\times o},\ p_{\mathsf{out}} \in \mathbb{R}^{o\times 1}$$
$$\vdash\ \lambda x.\mathsf{pred}(x, j) \geq \mathsf{threshold} : \textbf{entity} \to \mathsf{type}$$

Figure 2: Neural classifier in DTS

that maps each entity (namely, each element a_i of $\{a_1, \ldots, a_n\}$) to the i-th element in the standard basis $e_1, \ldots, e_n$ of an n-dimensional real vector space (defined as a product setoid). Additionally, let W be a linear setoid function from $\mathbb{R}^n$ to $\mathbb{R}^m$ that maps $e_1, \ldots, e_n$ to their respective embeddings in an m-dimensional real vector space.

Next, let assume the following parameters:

$$\begin{aligned} p_{\mathsf{emb\_ent}} &\in \mathbb{R}^{n\times m} \\ p_{\mathsf{emb\_pred}} &\in \mathbb{R}^{k\times l} \\ p_{\mathsf{hidden}} &\in \mathbb{R}^{(m+l)\times o} \\ p_{\mathsf{out}} &\in \mathbb{R}^{o\times 1} \end{aligned}$$

by which we define the following matrices as linear setoid functions:[6]

$$\begin{aligned} W_{\mathsf{emb\_ent}} &\in \mathbb{R}^n \to \mathbb{R}^m \\ W_{\mathsf{emb\_pred}} &\in \mathbb{R}^k \to \mathbb{R}^l \\ W_{\mathsf{hidden}} &\in \mathbb{R}^{(m+l)} \to \mathbb{R}^o \\ W_{\mathsf{out}} &\in \mathbb{R}^o \to \mathbb{R} \end{aligned}$$

Non-linear functions such as $\mathsf{sigmoid} \in \mathbb{R} \to \mathbb{R}$ can also be defined as setoid functions. By combining these components, a simple neural predicate with only one hidden layer can be defined as in Figure 1[7]. Note that neural predicates with more complicated structures such as RNNs and Transformers, are also definable in the same manner as the setoid functions in DTS.

Let us call this setoid function $\mathsf{pred}(a_i, j)$, where a_i is an entity and j indicates that this is the j-th unary predicate ($1 \leq j \leq k$) in the signature. Suppose that $\mathsf{threshold}$ is a real number hyperparameter between 0 and 1. Then the inequality between $\mathsf{pred}(a_i, j)$ and $\mathsf{threshold}$ is a DTS relation. Thus, a neural classifier of type **entity** $\to$ type obtained, as in Figure 2. Since **entity** $\to$ type is a type for one-place predicates in DTS, we may safely replace it with a corresponding neural classifier. It is straightforward to implement n-ary classifiers in a similar manner.

2 Learning Algorithm for Neural DTS

The next step for Neural DTS, which is absent in previous work, is to *fit* the parameters to data. The main contribution of this paper is to propose such an algorithm. We consider a setting where knowledge is given via real texts with credible content, and where "true" (or grounded) propositions can be extracted from the syntactic-semantic theory of natural language (this will be done in practice by CCG parsers). The algorithm optimizes the parameters in Neural DTS to make this set of propositions true. DTS is the best framework candidate since it has a proof theory (unlike most of the model-theoretic semantics) that enables this kind of proposition extraction, which has already been implemented, as in Bekki and Sato (2015) and Daido and Bekki (2020). The learning algorithm is described as follows.

2.1 Preprocessing

First, we prepare a collection of texts $(\mathsf{TXT}_d)_{d\in D}$ whose contents we assume to be correct. This will

[6] From the theoretical point of view, it is worth noting that all of these are defined within the framework of MLTT. However, from the perspective of implementation, we may simply import an off-the-shelf deep learning library such as pytorch (`https://pytorch.org/`) or tensorflow (`https://www.tensorflow.org/`), and use arrays instead of product setoids. In other words, we don't have to implement setoids on top of the implementation of the DTS's type system.

[7] $\mathsf{sigmoid}$ is a sigmoid function defined as a setoid function. The one applied to the hidden layer is broadcasted to $\mathbb{R}^o$. The $\oplus$ operator is the vector concatenation.

be used as the training data for Neural DTS. Also, let each

$$p_e \in \mathbb{R}^{(n\times m)+(k\times l)+((m+l)\times o)+(o\times 1)}$$

be the parameter of the neural network at epoch e.

1. Perform syntactic analysis on each $(\mathsf{TXT}_d)_{d\in D}$ by using CCG parser. Let $(\mathsf{SYN}_d)_{d\in D}$ be the resulting (1-best) syntactic structures.

2. Perform semantic composition on each $(\mathsf{SYN}_d)_{d\in D}$. Taking DTS to be the semantic theory, it is ensured that a syntactic structure will yield a well-formed semantic representation via syntax-semantics transparency. Let $(\mathsf{SEM}_d)_{d\in D}$ be the resulting DTS representations, where each SEM_d is a type that contains the parameter p_1 (the initial parameter) as a free variable.

2.2 Training Loop

Let e be the epoch. Loop the following steps, starting from $e = 1$.

1. Obtain a collection of *positive* predicates $(\mathsf{pred}(a,j))_{(a,j)\in T^+}$, where T^+ is a subset of $\mathbf{entity} \times \{1,\ldots,k\}$, each of which being deduced from one of the $(\mathsf{SEM}_d)_{d\in D}$ by using only the elimination rules (such as (ΠE) and (ΣE)).

2. Prepare a collection of *negative* predicates $(\mathsf{pred}(a,j))_{(a,j)\in T^-}$, where T^- is a randomly selected subset of $(\mathbf{entity} \times \{1,\ldots,k\}) - T^+$ such that $|T^-| = |T^+|$.

3. The loss function loss_e is a setoid function defined for each epoch e as follows:

$$\begin{aligned}\mathsf{loss}_e \stackrel{def}{\equiv} &\sum_{(a,j)\in T^+} |1-\mathsf{pred}(a,j)|^2 \\ &+ \sum_{(a,j)\in T^-} |0-\mathsf{pred}(a,j)|^2\end{aligned}$$

   loss_e is minimized when each of $(\mathsf{pred}(a,j))_{(a,j)\in T^+}$ is 1 and each of $(\mathsf{pred}(a,j))_{(a,j)\in T^-}$ is 0.

   Notice that loss_e contains a free variable p_e. We need to show that $\Gamma \vdash \lambda p_e.\mathsf{loss}_e \in \mathbb{R}$ is a differentiable function, which is a repetition of the standard argument in analysis but in terms of setoids.

4. Update the parameters. Stop the loop if loss_e falls below a certain threshold; or if e reaches a certain value. Otherwise set $e = e + 1$ and go to the step 1.

   We refer to this set of procedures as *one round*.

The reason for using only the elimination rules in step 1 is that it is computationally faster, in the sense that substitution does not arise in the premise part of the elimination rules. At the same time, however, this restricts the deduction power of DTS at the minimum level. Looser restrictions with greater computational burden should also be investigated.

Note that in this algorithm, the value of the loss function is not guaranteed to converge since the logical deduction of DTS is used in the training loop to generate the training data for each epoch. This feature of the algorithm is a drawback from an engineering viewpoint, but from a cognitive viewpoint, it may reflect the process of how humans learn knowledge, where our beliefs do not always converge by the increase of knowledge. Also, this algorithm proposes a particular method for the interaction of logical deduction and machine learning at the level of the training loop. These are left as topics for further study.

2.3 Evaluation Methods

One way to evaluate this learning algorithm is to check its soundness and completeness: the direct computation is compared with the training set. In this case, precision corresponds to soundness and recall to completeness.

Next, for the embedding of predicates, we would evaluate whether empirically similar predicates have similar embeddings on the Neural DTS. This would follow the standard methods, where measures like 5best, 1best, among others, are valid.

3 Discussion

The advantage of Neural DTS over other representation learning methods using DNN is that it is sensitive to polarities such as negation and conditional clauses in the text. For example, when the proposition A is included in the scope of the negation in one of $(\mathsf{SEM}_d)_{d\in D}$, as in the example below, it is not added to the training data since A is not deduced from $\neg(A \times B)$.

$$\neg(A \times B) \quad \not\vdash \quad A$$

Also, A is not added to the training data when it is included in the antecedent part of the conditional

sentence, as in the example below, since A is not deduced from $A \to B$.

$$A \to B \quad \not\vdash \quad A$$

In other words, due to the soundness of the logical deduction, once the mapping from text to semantic representation is performed, the set of positive predicates obtained from it by deduction is guaranteed to be correct, as long as the text is correct. This is a major advantage of having a logic system like DTS at the core of the knowledge learning system.

On the other hand, the reliability of the training data is sensitive to the validity of the syntactic theory that generates the semantic representations in DTS; here, the CCG parser and the lexicon do the job. Given that the accuracy of off-the-shelf CCG parsers is not yet sufficient, one might rather consider methods that use only dependency parsers more prospective, or even an end-to-end neural system that does not assume the division of labor among these modules from the beginning, that is, a language model such as an RNN or a transformer whose final layer is trained as classifier for predicates. These approaches may seem more robust for some researchers, but less precise for the rest of the researchers. Comparing these approaches is methodologically difficult since they are based on different views on language faculty.

However, using the CCG parser and the approach of making lexical semantics more precise is the study of formal syntax and of formal semantics itself. In other words, in this enterprise, the improvement of the formal syntax and the formal semantics is directly related to the improvement of the lexical semantics and the cognitive capacity.

4 Conclusions and Future Work

This paper proposed a learning algorithm for Neural DTS for acquiring knowledge representations from text, and discussed its features, expected advantages, and difficulties.

The next step is to implement, experiment with, and evaluate this algorithm. Some difficulties are expected to be caused by errors in the syntactic parsing, by ambiguity of predicate symbols, and by the problem of entity size (namely, the number of entities forming the enumeration type).

Moreover, many of the philosophical issues raised in Bekki et al. (2021) are left open. For example, unlike DTS, all predicates have canonical proofs in Neural DTS, the implication of which is still an open question.

We would like to leave these issues, both computational and philosophical, for future discussion.

Acknowledgments We sincerely thank the anonymous reviewers of NALOMA22 for their comments. This work was partially supported by the Japan Science and Technology Agency (JST), CREST Grant Number JPMJCR20D2.22.

Appendix

A Dependent Type Theory (DTT)

A.1 Syntax

Definition A.1 (Alphabet) *An alphabet is a pair $(\mathcal{V}ar, \mathcal{C}on)$ where $\mathcal{V}ar$ is a collection of variables and $\mathcal{C}on$ is a collection of constant symbols.*

Definition A.2 (Preterms) *The collection of preterms of DTT (notation Λ) under an alphabet $(\mathcal{V}ar, \mathcal{C}on)$ is defined by the following BNF grammar, where $x \in \mathcal{V}ar$ and $c \in \mathcal{C}on$.*

$$\begin{array}{l}
\Lambda := x \mid c \mid \mathsf{type} \\
\mid (x:\Lambda) \to \Lambda \mid \lambda x.\Lambda \mid \Lambda\Lambda \\
\mid (x:\Lambda) \times \Lambda \mid (\Lambda,\Lambda) \mid \pi_1(\Lambda) \mid \pi_2(\Lambda) \\
\mid \Lambda \oplus \Lambda \mid \iota_1(\Lambda) \mid \iota_2(\Lambda) \mid \mathsf{unpack}_L\,(M,N) \\
\mid \{a_1,\ldots,a_n\} \mid a_1 \mid \ldots \mid a_n \mid \mathsf{case}^{\Lambda}_{\Lambda}(\Lambda,\ldots,\Lambda) \\
\mid \Lambda =_{\Lambda} \Lambda \mid \mathsf{refl}_{\Lambda}(\Lambda) \mid \mathsf{idpeel}^{\Lambda}_{\Lambda}(\Lambda) \\
\mid \mathbb{N} \mid \mathsf{0} \mid \mathsf{s}(\Lambda) \mid \mathsf{natrec}^{\Lambda}_{\Lambda}(\Lambda,\Lambda)
\end{array}$$

Free variables, substitutions, β-reductions are defined in the standard way. The full version of DTT also employs well-ordered types and universes, as adopted in Martin-Löf (1984), the detail of which I omit here for the sake of space.

Definition A.3 (Vertical/Box notation)

$$\left[\begin{array}{l} x:A \\ B \end{array}\right] \stackrel{def}{\equiv} (x:A) \times B$$

Definition A.4 (Logical operators) [8]

$$\begin{array}{rcll}
A \to B & \stackrel{def}{\equiv} & (x:A) \to B & where\ x \notin fv(B). \\
\left[\begin{array}{l} A \\ B \end{array}\right] & \stackrel{def}{\equiv} & \left[\begin{array}{l} x:A \\ B \end{array}\right] & where\ x \notin fv(B). \\
\bot & \stackrel{def}{\equiv} & \{\} & \\
\neg A & \stackrel{def}{\equiv} & A \to \bot &
\end{array}$$

[8] $\bot$ is defined as an empty enumeration type.

A.2 Type System

Definition A.5 (Signature) *A collection of signatures (notation σ) for an alphabet $(\mathcal{V}ar, \mathcal{C}on)$ is defined by the following BNF grammar:*

$$\sigma \quad ::= \quad () \mid \sigma, c : A$$

where () is an empty signature, $c \in \mathcal{C}on$ and $\vdash_\sigma A : \mathsf{type}$.

Definition A.6 (Context) *A collection of contexts under a signature σ (notation Γ) is defined by the following BNF grammar:*

$$\Gamma \quad ::= \quad () \mid \Gamma, x : A$$

where () is an empty context, $x \in \mathcal{V}ar$ and $\Gamma \vdash_\sigma \mathsf{type}$.

Definition A.7 (Judgment) *A judgment of DTT is the following form*

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

where Γ is a context under a signature σ and M and A are preterms, which states that there exists a proof diagram of DTT from the context Γ to the type assignment $M : A$. The subscript σ may be omitted when no confusion arises.

Definition A.8 (Truth) *The judgment of the form $\Gamma \vdash A\ true$ states that there exists a term M that satisfies $\Gamma \vdash M : A$.*

Definition A.9 (Structural Rules)

$$\frac{A : \mathsf{type}}{x : A}\,(VAR)$$

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

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

$$\frac{M : A \quad N : B}{M : A}\,(WK)$$

$$\frac{M : A}{M : B}\,(CONV) \quad where\ A =_\beta B.$$

Definition A.10 (Π-types)

$$\frac{\begin{array}{c}\overline{x : A}^{\,i} \\ \vdots \\ A : \mathsf{s}_1 \quad B : \mathsf{s}_2\end{array}}{(x : A) \to B : \mathsf{s}_2}\,(\Pi F),i$$

$$where\ (\mathsf{s}_1, \mathsf{s}_2) \in \left\{ \begin{array}{l} (\mathsf{type}, \mathsf{type}), \\ (\mathsf{type}, \mathsf{kind}) \end{array} \right\}.$$

$$\frac{\begin{array}{c}\overline{x : A}^{\,i} \\ \vdots \\ M : B\end{array}}{\lambda x.M : (x : A) \to B}\,(\Pi I),i$$

$$\frac{M : (x : A) \to B \quad N : A}{MN : B[N/x]}\,(\Pi E)$$

Definition A.11 (Σ-types)

$$\frac{\begin{array}{c}\overline{x : A}^{\,i} \\ \vdots \\ A : \mathsf{type} \quad B : \mathsf{type}\end{array}}{(x : A) \times B : \mathsf{type}}\,(\Sigma F),i$$

$$\frac{M : A \quad N : B[M/x]}{(M, N) : (x : A) \times B}\,(\Sigma I)$$

$$\frac{M : (x : A) \times B}{\pi_1(M) : A}\,(\Sigma E)$$

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

Definition A.12 (Disjoint Union Types)

$$\frac{A : \mathsf{type} \quad B : \mathsf{type}}{A \uplus B : \mathsf{type}}\,(\uplus F)$$

$$\frac{M : A}{\iota_1(M) : A \uplus B}\,(\uplus I)$$

$$\frac{N : B}{\iota_2(N) : A \uplus B}\,(\uplus I)$$

$$\frac{\begin{array}{l} L : A \uplus B \\ P : (A \uplus B) \to \mathsf{type} \\ M : (x : A) \to P(\iota_1(x)) \\ N : (x : B) \to P(\iota_2(x)) \end{array}}{\mathsf{unpack}_L^P\,(M, N) : P(L)}\,(\uplus E),i$$

Definition A.13 (Enumeration Types)

$$\frac{}{\{a_1, \ldots, a_n\} : \mathsf{type}}\,(\{\}F)$$

$$\frac{}{a_i : \{a_1, \ldots, a_n\}}\,(\{\}I)$$

$$\frac{\begin{array}{l} M : \{a_1, \ldots, a_n\} \\ P : \{a_1, \ldots, a_n\} \to \mathsf{type} \\ N_1 : P(a_1) \\ \ldots \\ N_n : P(a_n) \end{array}}{\mathsf{case}_M^P\,(N_1, \ldots, N_n) : P(M)}\,(\{\}E)$$

Definition A.14 (Intensional Equality Types)

$$\frac{A : \mathsf{type} \quad M : A \quad N : A}{M =_A N : \mathsf{type}}\ (=F)$$

$$\frac{M : A}{\mathsf{refl}_A(M) : M =_A M}\ (=I)$$

$$\frac{\begin{array}{l} E : M =_A N \\ P : (x : A) \to (y : A) \to (x =_A y) \to \mathsf{type} \\ R : (x : A) \to Pxx(\mathsf{refl}_A(x)) \end{array}}{\mathsf{idpeel}^P_E(R) : PMNE}\ (=E)$$

Definition A.15 (Natural Number Types)

$$\frac{}{\mathbb{N} : \mathsf{type}}\ (\mathbb{N}F)$$

$$\frac{}{\mathsf{0} : \mathbb{N}}\ (\mathbb{N}I) \qquad \frac{n : \mathbb{N}}{\mathsf{s}(n) : \mathbb{N}}\ (\mathbb{N}I)$$

$$\frac{\begin{array}{l} n : \mathbb{N} \\ P : \mathbb{N} \to \mathsf{type} \\ e : P(\mathsf{0}) \\ f : (k : \mathbb{N}) \to P(k) \to P(\mathsf{s}(k)) \end{array}}{\mathsf{natrec}^P_n(e, f) : P(n)}\ (\mathbb{N}E)$$

A.3 Setoids

Definition A.16 *A setoid is a pair $(\underline{X}, \sim_X)$ consisting of a type $\underline{X}$ and an equivalence relation $\sim_X$ on $\underline{X}$.*

Definition A.16 can be rewritten in the form of the formation rule as follows.

Definition A.17 (Setoid formation)

$$\frac{\begin{array}{l} \underline{X} : \mathsf{type} \\ \sim_X : \underline{X} \times \underline{X} \to \mathsf{type} \\ equiv(\sim_X)\ true \end{array}}{(\underline{X}, \sim_X)\ setoid}$$

Definition A.18 (Setoid membership)

$$\frac{(\underline{X}, \sim_X)\ setoid \quad x : \underline{X}}{x \in (\underline{X}, \sim_X)}$$

Definition A.19 (Setoid function) *A setoid function f from a setoid X to a setoid Y is a pair $(\underline{f}, ext_f)$ consisting of a function $\underline{f} : \underline{X} \to \underline{Y}$ and a proof term ext_f that proves the extensionality of $\underline{f}$:*

$$ext_f \ : \ (x, y : \underline{X}) \to (x \sim_X y) \to (\underline{f}x \sim_Y \underline{f}y)$$

Definition A.20 (Exponential of setoids) *The exponential of setoids $X \equiv (\underline{X}, \sim_X)$ and $Y \equiv (\underline{Y}, \sim_Y)$ is $(\underline{X \to Y}, \sim_E)$ (notation: $X \to Y$), where $\underline{X \to Y}$ is a type defined as:*

$$\underline{X \to Y} \stackrel{def}{\equiv} \big(\underline{f} : \underline{X} \to \underline{Y}\big) \times (x, y : \underline{X}) \to (x \sim_X y) \to (\underline{f}x \sim_Y \underline{f}y)$$

and $\sim_E$ is a binary relation defined as:

$$(\underline{f}, \_) \sim_E (\underline{g}, \_) \stackrel{def}{\equiv} (x : \underline{X}) \to \underline{f}x \sim_Y \underline{g}x$$

A function application operator ev is defined for each domain-codomain pair of setoids.

$$\frac{(\underline{f}, \_) \in X \to Y \quad a \in X}{ev_{X,Y}((\underline{f}, \_), a) \stackrel{def}{\equiv} \underline{f}a \in Y}$$

Definition A.21 (Product of setoids) *The product of setoids $X \equiv (\underline{X}, \sim_X)$ and $Y \equiv (\underline{Y}, \sim_Y)$ is the pair $(\underline{X \times Y}, \sim_P)$ (notation: $X \times Y$), where $\underline{X \times Y}$ is a type defined as:*

$$\underline{X \times Y} \stackrel{def}{\equiv} \underline{X} \times \underline{Y}$$

and $\sim_P$ is a binary relation defined as:

$$(x, y) \sim_P (u, v) \stackrel{def}{\equiv} (x \sim_X u) \times (y \sim_Y v)$$

Projections work as expected.

$$\frac{p \in X \times Y}{\pi_1(p) \in X} \qquad \frac{p \in X \times Y}{\pi_2(p) \in Y}$$

Definition A.22 (Relation on setoids) *A binary relation R between setoids X and Y is a pair $(\underline{R}, ext_R)$ consisting of a binary relation $\underline{R}$ such that $x : \underline{X}, y : \underline{Y} \vdash \underline{R}(x, y) : \mathsf{type}$ together with a proof term ext_R that proves the extensionality of $\underline{R}$:*

$$ext_R \ : \ \big(x, x' : \underline{X}\big) \to \big(y, y' : \underline{Y}\big) \to x \sim_X x' \to y \sim_Y y' \to \underline{R}xy \to \underline{R}x'y'$$

Definition A.23 (Quotient setoids) *Let $X \equiv (\underline{X}, \sim_X)$ be a setoid and $\sim$ be a binary relation on $\underline{X}$ such that $x : \underline{X}, y : \underline{X} \vdash x \sim y : \mathsf{type}$. If $\sim$ is an equivalence relation on $\underline{X}$, we define a quotient setoid $X/\sim$ as:*

$$X/\sim \stackrel{def}{\equiv} (\underline{X}, \sim)$$

with a setoid function $q : X \to X/\sim$ defined by identity function on $\underline{X}$ and its extensionality.

Remark A.24 *By the extensionality of the relation, the following holds for any $x, y \in X$:*

$$x \sim_X y \to x \sim y$$

Thus the equivalence relation $\sim_X$ is finer than $\sim$ on the type $\underline{X}$.

Definition A.25 (Subsetoids) *Let X be a setoid. A subsetoid of X is a pair $(\partial S, i_S)$, where ∂S is a setoid and $i_S : \partial S \to X$ is an injective setoid function.*

Definition A.26 (Subsetoid membership) *Let X be a setoid. An element $a \in X$ is a member of the subsetoid S of X if there exists an element $s : \partial S$ such that $a \sim_X i(s)$, namely:*

$$a \in_X (\partial S, i_S) \stackrel{def}{\equiv} (s : \partial S) \times (a \sim_X i_S(s))$$

Note that $a \in_X (\partial S, i_S)$ is a type. If A and B are subsetoids of X, then

$$A \subseteq_X B \stackrel{def}{\equiv} (x : X) \to x \in_X A \to x \in_X B$$

Definition A.27 (Separation of subsets) *Let $X = (\underline{X}, \sim_X)$ be a setoid and A a type. A subset $\{\ x \in X \mid A\ \}$ of X is defined as:*

$$\{\ x \in X \mid A\ \} \stackrel{def}{\equiv} (((x : \underline{X}) \times A, \sim_S), i)$$

where $(x, \_) \sim_S (y, \_) \stackrel{def}{\equiv} x \sim_X y$

$$i(x, \_) \stackrel{def}{\equiv} x$$

A.4 Setoids of Natural Numbers, Integers, Rationals, and Reals

Definition A.28 *The setoid of natural numbers $\mathbb{N}$ is obtained by embedding the natural number type N with its intensional equality.*

$$\mathbb{N} \stackrel{def}{\equiv} (N, =_N)$$

Definition A.29 *The setoid of integers $\mathbb{Z}$ is defined as:*

$$\mathbb{Z} \stackrel{def}{\equiv} (\mathbb{N} \times \mathbb{N}) / \sim_{\mathbb{Z}}$$

where $\sim_{\mathbb{Z}}$ is defined as $(m, n) \sim_{\mathbb{Z}} (p, q) \stackrel{def}{\equiv} m + q =_N p + n$.

Definition A.30 *The setoid of rational numbers $\mathbb{Q}$ is defined as:*[9]

$$\mathbb{Q} \stackrel{def}{\equiv} (\mathbb{Z} \times \{\ z \in \mathbb{Z} \mid \neg(z \sim_{\mathbb{Z}} 0_{\mathbb{Z}})\ \}) / \sim_{\mathbb{Q}}$$

where $\sim_{\mathbb{Q}}$ is defined as

$$(a, (b, \_)) \sim_{\mathbb{Q}} (c, (d, \_)) \stackrel{def}{\equiv} a \cdot d =_{\sim_{\mathbb{Z}}} c \cdot b$$

Definition A.31 (Cauchy sequence) $\mathsf{Cauchy}(seq)$ *is a proposition that a sequence $seq \in \mathbb{N} \to \mathbb{Q}$ is a Cauchy sequence, defined as follows:*[10]

$$\begin{aligned}&\mathsf{Cauchy}(seq) \stackrel{def}{\equiv}\\ &(i : \mathbb{N}) \to (i \neq 0) \to (k : \mathbb{N}) \times\\ &(j_1, j_2 : \mathbb{N}) \to (j_1 > k) \to (j_2 > k)\\ &\to |(\underline{seq}(j_1)) - (\underline{seq}(j_2))| < \frac{1}{\mathsf{asRational}^*(i)}\end{aligned}$$

Definition A.32 *The setoid of real numbers $\mathbb{R}$ is defined as:*

$$\mathbb{R} \stackrel{def}{\equiv} \{\ seq \in \mathbb{N} \to \mathbb{Q} \mid \mathsf{Cauchy}(seq)\ \} / \sim_{\mathbb{R}}$$

where $\sim_{\mathbb{R}}$ is defined as:

$$\begin{aligned}&s_1 \sim_{\mathbb{R}} s_2 \stackrel{def}{\equiv}\\ &(i : \mathbb{N}) \to (i \neq 0) \to (k : \mathbb{N}) \times (j : \mathbb{N}) \to (j > k)\\ &\to |\underline{\pi_1(s_1)}(j) - \underline{\pi_1(s_2)}(j)| < \frac{1}{\mathsf{asRational}^*(i)}\end{aligned}$$

[9] $0_{\mathbb{Z}}$ is defined as $(0, 0) : \underline{\mathbb{Z}}$.

[10] The function $\mathsf{asRational}^*$ is defined as a compostion:

$$\mathsf{asRational}^* \stackrel{def}{\equiv} \mathsf{asRational} \circ \mathsf{asInteger} : \underline{\mathbb{N}} \to \underline{\mathbb{Q}}$$

where each *casting* function is defined as follows:

$$\begin{aligned}\mathsf{asInteger} &\stackrel{def}{\equiv} \lambda n.(n, 0) : \underline{\mathbb{N}} \to \underline{\mathbb{Z}}\\ \mathsf{asRational} &\stackrel{def}{\equiv} \lambda z.(z, 1_{\mathbb{Z}}) : \underline{\mathbb{Z}} \to \underline{\mathbb{Q}}\end{aligned}$$

References

Daisuke Bekki. 2014. Representing anaphora with dependent types. In *Logical Aspects of Computational Linguistics (8th international conference, LACL2014, Toulouse, France, June 2014 Proceedings), LNCS 8535*, pages 14–29. Springer, Heiderburg.

Daisuke Bekki. 2021. Proof-theoretic analysis of weak crossover. In *Logic and Engineering of Natural Language Semantics 18 (LENLS18)*, pages 75–88.

Daisuke Bekki and Koji Mineshima. 2017. *Context-passing and Underspecification in Dependent Type Semantics*, Studies of Linguistics and Philosophy, pages 11–41. Springer.

Daisuke Bekki and Miho Sato. 2015. Calculating projections via type checking. In *TYpe Theory and LExical Semantics (TYTLES), ESSLLI2015 workshop*.

Daisuke Bekki, Ribeka Tanaka, and Yuta Takahashi. 2021. Integrating deep neural network with dependent type semantics. In *the Symposium Logic and Algorithms in Computational Linguistics 2021 (LACompLing2021)*, page p.37. Stockholm University, 2021, DiVA Portal for Digital Publications.

Hinari Daido and Daisuke Bekki. 2020. Development of an automated theorem prover for the fragment of dts. In *the 17th International Workshop on Logic and Engineering of Natural Language Semantics (LENLS17)*.

Rajarshi Das, Arvind Neelakantan, David Belanger, and Andrew McCallum. 2017. Chains of reasoning over entities, relations, and text using recurrent neural networks. Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers, pages 132–141. Association for Computational Linguistics.

Thomas Demeester, Tim Rocktäschel, and Sebastian Riedel. 2016. Lifted rule injection for relation embeddings. Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 1389–1399. Association for Computational Linguistics.

Nissim Francez. 2014. *The Granularity of Meaning in Proof-Theoretic Semantics*, pages 96–106. Springer, Toulouse.

Nissim Francez and Roy Dyckhoff. 2010. Proof-theoretic semantics for a natural language fragment. *Linguistics and Philosophy*, 33(6):447–477.

Kelvin Guu, John Miller, and Percy Liang. 2015. Traversing knowledge graphs in vector space. Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 318–327. Association for Computational Linguistics.

Hans Kamp, J. van Genabith, and Uwe Reyle. 2011. *Discourse Representation Theory*, volume 15, pages 125–394. Springer, Doredrecht.

Mike Lewis and Mark Steedman. 2013. Combined distributional and logical semantics. *Transactions of the Association for Computational Linguistics*, 1:179–192.

Per Martin-Löf. 1984. *Intuitionistic Type Theory*, volume 17. Italy: Bibliopolis, Naples.

Tim Rocktäschel and Sebastian Riedel. 2017. End-to-end differentiable proving. In *the 31st International Conference on Neural Information Processing Systems*. Curran Associates, Inc.

Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. 2019. Learning a SAT solver from single-bit supervision. In *ICLR (Poster)*.

Ryo Takahashi, Ran Tian, and Kentaro Inui. 2018. Interpretable and compositional relation learning by joint training with an autoencoder. Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 2148–2159. Association for Computational Linguistics.

Ribeka Tanaka. 2021. *Natural Language Quantification and Dependent Types*. Doctoral dissertation.

Gustav Šourek, Vojtěch Aschenbrenner, Filip Železný, Steven Schockaert, and Ondřej Kuželka. 2018. Lifted relational neural networks: efficient learning of latent relational structures. *J. Artif. Int. Res.*, 62(1):69–100.

Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. 2017. Premise selection for theorem proving by deep graph embedding. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 2783–2793. Curran Associates Inc.

Yukiko Yana, Koji Mineshima, and Daisuke Bekki. 2019. Variable handling and compositionality: Comparing drt and dts. *Journal of Logic, Language and Information*, 28(2):261–285.

Yukiko Yana, Koji Mineshima, and Daisuke Bekki. 2021. The proviso problem from a proof-theoretic perspective. In *Logical Aspects of Computational Linguistics (LACL) 2021*, pages 159–176.