Calculating Projections via Type Checking ⋆

Daisuke Bekki[1,2,3] and Miho Satoh[1]

[1] Ochanomizu University, Graduate School of Humanities and Sciences ⋆⋆
[2] National Institute of Informatics ⋆⋆⋆
[3] CREST, Japan Science and Technology Agency †

1 Projection in Dependent Type Semantics

In recent years, formal semantics in the context of dependent type theory [9], which originates in Sundholm's [15] and Ranta's [12] seminal works, has achieved gradual progress: Cooper's type theory with records [4], Luo and Asher's type theory with coercive subtyping [7][1], and Martin and Pollard's [8] dynamic categorial grammar, among others. Meanwhile, dependent type semantics (DTS [2][3]) is a compositional framework of natural language semantics whose calculation of projective contents (namely, presupposition, anaphora, and conventional implicatures) reduces to *type checking* in dependent type theory, and whose presupposition binding/anaphora resolution reduces to a proof search (along the lines of Krahmer and Piwek [5][11] and Mineshima [10]). For example, in (1), for each pair of sentences, the left side of ⇒ has the right side of ⇒ as its projective content, and this empirical relation is calculated by type checking in DTS.[4]

(1) a. Sweden does not cherish its king.
⇒ Sweden has a king.
b. If Sweden is a monarchy, Sweden cherishes its king.
⇒ If Sweden is a monarchy, Sweden has a king.
c. Every monarchy cherishes its king.
⇒ Every monarchy has a king.
d. Sweden, a monarchy, cherishes its king.
⇒ Sweden is a monarchy/has a king.

⋆ Our sincere thanks to Kenichi Asai, Koji Mineshima, Ribeka Tanaka for many helpful discussions. I also thank the anonymous reviewers of TYTLES for their insightful comments. Daisuke Bekki is partially supported by JST, CREST.
⋆⋆ 2-1-1 Ohtsuka, Bunkyo-ku, Tokyo 112-8610, Japan.
⋆⋆⋆ 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, 101-8430, Japan.
† 4-1-8 Honcho, Kawaguchi, Saitama, 332-0012, Japan.
[4] Due to the page limit, the following intriguing issues are not discussed in the abstract: the intermediate accommodation case in (1c), and comparison between our "proof theoretic" account and pragmatic accounts of projection, such as the ones in Simons et al. [13] and Tonhauser et al. [16], among others.

Two concepts in DTS that enable the unity are *underspecified terms* and *local contexts*. An underspecified term (notated as $@_i$, where i is a natural number) is a proof term for a projective content. A local context is a proof term of the preceding discourse for a sentence whose representation in DTS is a function that takes a local context and returns a type (that is, a data type for proof terms). Most underspecified terms are functions that receive a local context as their argument in the context-passing mechanism of DTS. Then, anaphora resolution and presupposition binding are formulated as a substitution of underspecified terms by some proof terms, which are to be found via proof search that is unified with the inference system for calculating entailments. Different proof terms correspond to different choices of antecedents.

The whole system is integrated within a standard Montagovian compositional setting; for instance, the sentences in (1) are derived by the lexical items listed in Figure 1[5], where presupposition/CI triggers and anaphora introduce an underspecified term[6,7], which yields the semantic representations in (2).

An underspecified term of type A in DTS requires the existence of a proof term of type A, which means that A (as a proposition) is *true* regardless of the truth of the sentence itself; thus, A is projective.

Note that local contexts received by underspecified terms @ in the semantic representations in (2) vary according to their syntactic configurations. In (2a), $@_1$ receives the proof term c of its preceding discourse alone. In (2b), the proof term u of the antecedent part of the implication is also passed to $@_1$. In (2c), a given proof term is further abstracted. In (2d), $@_3$, which corresponds to the CI content, does not receive any local context.

Thus, the type of an underspecified term depends on the type of the local context it receives, but this information is recoverable by type checking. This is the reason that projective contents can be calculated via type checking in DTS.

2 Type Checking/Inference Algorithm

Despite the above, the type checking/inference algorithm for DTS remains undefined in [2]. This is due to two technical problems: 1) the undecidability of type checking in dependent type theory [14], which DTS is based on, and 2) the existence of underspecified terms.

---

[5] We adopt the notation $\begin{bmatrix} x{:}A \\ B(x) \end{bmatrix}$ for the dependent sum type $\Sigma x : A.B(x)$, and write $(x : A) \to B(x)$ for the dependent functional type $\Pi x : A.B$.

[6] The representation of the word *its* contains two underspecified terms $@_i$ and $@_j$: the former (i.e. an anaphoric part) takes a local context and returns a pair of some entity and a proof of its non-humanness. The latter (i.e. possessive presupposition) takes a local context and returns a triple of some entity, a proof that it belongs to the nominal category specified by the preceding common noun, and a proof that it is owned by *it*.

[7] The compositional analysis of the appositive in (2d) is due to Bekki and McCready [3].

PF	Syntactic categories	Semantic representations
if	$S/S/S$	$\lambda p.\lambda q.\lambda c.\,(u{:}pc) \to (q(c,u))$
every$_{nom}$	$S/(S\backslash NP)/N$	$\lambda n.\lambda p.\lambda c.\,(x{:}\mathbf{entity}) \to (u{:}nx(c,x)) \to (px(c,(x,u)))$
a$_{acc}$	$(S\backslash NP)\backslash(S\backslash NP/NP)/N$	$\lambda n.\lambda p.\lambda x.\lambda c.\,\left[\begin{array}{l} y{:}\mathbf{entity} \\ \left[\begin{array}{l} v{:}ny(c,y) \\ pyx(c,(y,v)) \end{array}\right] \end{array}\right]$
king	N	$\lambda x.\lambda c.\mathbf{king}(x)$
cherish	$S\backslash NP/NP$	$\lambda y.\lambda x.\lambda c.\mathbf{cherish}(x,y)$
its$_{acc}$	$(S\backslash NP)\backslash(S\backslash NP/NP)/N$	$\lambda n.\lambda p.\lambda x.\lambda c.p\left(\pi_1\left(@_j c :: \left[\begin{array}{l} y{:}\mathbf{entity} \\ \left[\begin{array}{l} nyc \\ \mathbf{have}\left(\pi_1\left(@_i c :: \left[z{:}\mathbf{entity} \atop \neg\mathbf{human}(z)\right]\right), y\right) \end{array}\right] \end{array}\right]\right)\right) xc$

Fig. 1. Lexical items in DTS (where $@_i$ and $@_j$ are underspecified terms)

(2) a. $\lambda c.\neg\mathbf{cherish}\left(sweden, \pi_1\left(@_1(c) :: \left[\begin{array}{l} y{:}\mathbf{entity} \\ \left[\begin{array}{l} \mathbf{king}(y) \\ \mathbf{have}\left(\pi_1\left(@_2(c) :: \left[z{:}\mathbf{entity} \atop \neg\mathbf{human}(z)\right]\right), y\right) \end{array}\right] \end{array}\right]\right)\right)$

b. $\lambda c.(u : \mathbf{monarchy}(sweden)) \to \mathbf{cherish}\left(sweden, \pi_1\left(@_1(c,u) :: \left[\begin{array}{l} y{:}\mathbf{entity} \\ \left[\begin{array}{l} \mathbf{king}(y) \\ \mathbf{have}\left(\pi_1\left(@_2(c,u) :: \left[z{:}\mathbf{entity} \atop \neg\mathbf{human}(z)\right]\right), y\right) \end{array}\right] \end{array}\right]\right)\right)$

c. $\lambda c.(x : \mathbf{entity}) \to (u : \mathbf{monarchy}(x)) \to \mathbf{cherish}\left(x, \pi_1\left(@_1(c,(x,u)) :: \left[\begin{array}{l} y{:}\mathbf{entity} \\ \left[\begin{array}{l} \mathbf{king}(y) \\ \mathbf{have}\left(\pi_1\left(@_2(c,(x,u)) :: \left[z{:}\mathbf{entity} \atop \neg\mathbf{human}(z)\right]\right), y\right) \end{array}\right] \end{array}\right]\right)\right)$

d. $\lambda c.\left[\begin{array}{l} \mathbf{cherish}\left(sweden, \pi_1\left(@_1(c) :: \left[\begin{array}{l} y{:}\mathbf{entity} \\ \left[\begin{array}{l} \mathbf{king}(y) \\ \mathbf{have}\left(\pi_1\left(@_2(c) :: \left[z{:}\mathbf{entity} \atop \neg\mathbf{human}(z)\right]\right), y\right) \end{array}\right] \end{array}\right]\right)\right) \\ @_3 =_{\mathbf{monarchy}(sweden)} @_3 \end{array}\right]$

Fig. 2. Semantic representations for (1)

With regard to 1), the type checking system of the proof assistant Agda[8] employs the *annotation* construction of the form $M :: A$, which behaves just like the term M except that its type is specified as A (which turns a checkable term M into an inferable term), and presents checkable and inferable fragments of dependent type theory [6]. We adopt this strategy as well.

With regard to 2), what we need for DTS is not a type checker in the genuine sense of a functional programming language, but, instead, a checker that also *infers* a type for an underspecified term that appears in a given term. For this purpose, the type A of an underspecified term @ should be calculated from its surroundings, and when the algorithm tries to *check* whether @ has a type A within the given surroundings, it is supposed to update the list of judgements for underspecified terms by adding $\Gamma \vdash @ : A$.

However, the algorithm in [6] requires that the function in a function application construction must be an inferable term, which means that the type of an underspecified term @, in the form of function, must be inferable from the structure of @ itself. This is not the case.

[8] http://wiki.portal.chalmers.se/agda/

$$
\begin{aligned}
v &::= n \mid \mathsf{type} \mid \mathsf{kind} \mid \top \mid \bot \mid () \mid (x{:}v) \to v' \mid \begin{bmatrix} x{:}v \\ v' \end{bmatrix} \mid \lambda x.v \mid (v, v') \mid v \to v' \mid \begin{bmatrix} v \\ v' \end{bmatrix} \\
n &::= x \mid c \mid @_i \mid nv \mid \pi_i n \\
M_\uparrow &::= x \mid c \mid \mathsf{type} \mid (x{:}M_\downarrow) \to M_\downarrow \mid M_\uparrow M_\downarrow \mid \begin{bmatrix} x{:}M_\downarrow \\ M_\downarrow \end{bmatrix} \mid (M_\uparrow, M_\uparrow) \mid \pi_i M_\uparrow \mid M_\downarrow :: M_\downarrow \mid M_\downarrow \to M_\downarrow \mid \begin{bmatrix} M_\downarrow \\ M_\downarrow \end{bmatrix} \mid () \mid \top \mid \bot \\
M_\downarrow &::= M_\uparrow \mid \lambda x.M_\downarrow \mid M_\downarrow M_\uparrow \mid (M_\downarrow, M_\downarrow) \mid @_i
\end{aligned}
$$

Fig. 3. Values, neutral terms, inferable and checkable terms

(3) a. (for (2a)(2d)) $\delta : \mathsf{type},\ c : \delta \vdash @_2 : \delta \to \begin{bmatrix} z{:}\mathbf{entity} \\ \neg\mathbf{human}(z) \end{bmatrix}$

b. (for (2b)) $\delta : \mathsf{type},\ c : \delta,\ u : \mathbf{monarchy}(sweden) \vdash @_2 : \begin{bmatrix} \delta \\ \mathbf{monarchy}(sweden) \end{bmatrix} \to \begin{bmatrix} z{:}\mathbf{entity} \\ \neg\mathbf{human}(z) \end{bmatrix}$

c. (for (2c)) $\delta : \mathsf{type},\ c : \delta,\ x : \mathbf{entity},\ u : \mathbf{monarchy}(x) \vdash @_2 : \begin{bmatrix} \delta \\ \begin{bmatrix} \mathbf{entity} \\ \mathbf{monarchy}(x) \end{bmatrix} \end{bmatrix} \to \begin{bmatrix} z{:}\mathbf{entity} \\ \neg\mathbf{human}(z) \end{bmatrix}$

(4) a. (for (2a)(2d)) $\delta : \mathsf{type},\ c : \delta \vdash @_1 : \delta \to \begin{bmatrix} y{:}\mathbf{entity} \\ \begin{bmatrix} \mathbf{king}(y) \\ \mathbf{have}(sweden, y) \end{bmatrix} \end{bmatrix}$

b. (for (2b)) $\delta : \mathsf{type},\ c : \delta \vdash @_1 : \begin{bmatrix} \delta \\ \mathbf{monarchy}(sweden) \end{bmatrix} \to \begin{bmatrix} y{:}\mathbf{entity} \\ \begin{bmatrix} \mathbf{king}(y) \\ \mathbf{have}(sweden, y) \end{bmatrix} \end{bmatrix}$

c. (for (2c) $\delta : \mathsf{type},\ c : \delta,\ x : \mathbf{entity},\ u : \mathbf{monarchy}(x) \vdash @_1 : \begin{bmatrix} \delta \\ \begin{bmatrix} \mathbf{entity} \\ \mathbf{monarchy}(x) \end{bmatrix} \end{bmatrix} \to \begin{bmatrix} y{:}\mathbf{entity} \\ \begin{bmatrix} \mathbf{king}(y) \\ \mathbf{have}(x, y) \end{bmatrix} \end{bmatrix}$

(5) a. (for (2d)) $\delta : \mathsf{type},\ c : \delta \vdash @_3 : \mathbf{monarchy}(sweden)$

Fig. 4. Projective contents of (2)

The key observation to solve the second problem is that an underspecified term @ in DTS is always a simply typed function. Therefore, it suffices for the type of a simply typed function @ to be inferable from the type of a local context c and the type of the application $@(c)$ (the same situation arises for simply typed pairs, but we do not discuss the details of that case here). The main contributions of this work are as follows.

(i) We extend the fragment of dependent type theory given in [6] to include underspecified terms and dependent sum types, as defined in Figure 3, where $v, n, M_\uparrow$ and $M_\downarrow$ are the collections of *values*, *neutral terms*, *inferable terms*, and *checkable terms*, respectively.

(ii) We extend the type checking/inference rules of [6] as in Figure 5 for the fragment in Figure 3, where simply typed functions (and simply typed pairs) are inferable terms.

Note that any annotations that are required for the representation language of DTS to be confined to the fragment in Figure 3 can be naturally specified within the lexical representations of presupposition triggers, as shown in Figure 1.

Note also that the soundness of this algorithm—namely, that every judgment that is checked as true or inferred in this algorithm is also derivable in the original

$$\frac{[L]\ \Gamma \vdash_\sigma M :_\uparrow v\ [L']}{[L]\ \Gamma \vdash_\sigma M :_\downarrow v\ [L']}\ (CHK) \quad \frac{(x,v) \in \Gamma}{[L]\ \Gamma \vdash_\sigma x :_\uparrow v\ [L]}\ (VAR) \quad \frac{(c,v) \in \sigma}{[L]\ \Gamma \vdash_\sigma c :_\uparrow v\ [L]}\ (CON) \quad \frac{}{[L]\ \Gamma \vdash_\sigma \mathsf{type} :_\uparrow \mathsf{kind}\ [L]}\ (TYPE)$$

$$\frac{[L]\ \Gamma \vdash_\sigma A :_\downarrow \mathsf{s}_1\ [L'] \quad A \Downarrow_\beta v \quad [L']\ \Gamma,\ x : v \vdash_\sigma B :_\downarrow \mathsf{s}_2\ [L'']}{[L]\ \Gamma \vdash_\sigma (x{:}A) \to B :_\uparrow \mathsf{s}_2\ [L'']}\ (\Pi F) \quad (\mathsf{s}_1, \mathsf{s}_2 \in \{\mathsf{type}, \mathsf{kind}\})$$

$$\frac{[L]\ \Gamma,\ x : v \vdash_\sigma M :_\downarrow v'\ [L']}{[L]\ \Gamma \vdash_\sigma \lambda x.M :_\downarrow (x{:}v) \to v'\ [L']}\ (\Pi I) \quad \frac{[L]\ \Gamma \vdash_\sigma M :_\uparrow (x{:}v) \to v'\ [L'] \quad [L']\ \Gamma \vdash_\sigma N :_\downarrow v\ [L''] \quad v'[N/x] \Downarrow_\beta v''}{[L]\ \Gamma \vdash_\sigma MN :_\uparrow v''\ [L'']}\ (\Pi E)$$

$$\frac{[L]\ \Gamma \vdash_\sigma A :_\downarrow \mathsf{s}_1\ [L'] \quad [L']\ \Gamma \vdash_\sigma B :_\downarrow \mathsf{s}_2\ [L'']}{[L]\ \Gamma \vdash_\sigma A \to B :_\uparrow \mathsf{s}_2\ [L'']}\ (\to F) \quad (\mathsf{s}_1, \mathsf{s}_2 \in \{\mathsf{type}, \mathsf{kind}\})$$

$$\frac{[L]\ \Gamma,\ x : v \vdash_\sigma M :_\downarrow v'\ [L']}{[L]\ \Gamma \vdash_\sigma \lambda x.M :_\downarrow v \to v'\ [L']}\ (\to I) \quad \frac{[L]\ \Gamma \vdash_\sigma N :_\uparrow v\ [L'] \quad [L']\ \Gamma \vdash_\sigma M :_\downarrow v \to v'\ [L'']}{[L]\ \Gamma \vdash_\sigma MN :_\downarrow v'\ [L'']}\ (\to E)$$

$$\frac{[L]\ \Gamma \vdash_\sigma A :_\downarrow \mathsf{s}_1\ [L'] \quad A \Downarrow_\beta v \quad [L']\ \Gamma,\ x : v \vdash_\sigma B :_\downarrow \mathsf{s}_2\ [L'']}{[L]\ \Gamma \vdash_\sigma \begin{bmatrix} x{:}A \\ B \end{bmatrix} :_\uparrow \mathsf{s}_2\ [L'']}\ (\Sigma F) \quad (\mathsf{s}_1, \mathsf{s}_2 \in \{\mathsf{type}, \mathsf{kind}\})$$

$$\frac{[L]\ \Gamma \vdash_\sigma M :_\downarrow v\ [L'] \quad v'[M/x] \Downarrow_\beta v'' \quad [L']\ \Gamma \vdash_\sigma N :_\downarrow v''\ [L'']}{[L]\ \Gamma \vdash_\sigma (M,N) :_\downarrow \begin{bmatrix} x{:}v \\ v' \end{bmatrix}\ [L'']}\ (\Sigma I)$$

$$\frac{[L]\ \Gamma \vdash_\sigma M :_\uparrow \begin{bmatrix} x{:}v \\ v' \end{bmatrix}\ [L']}{[L]\ \Gamma \vdash_\sigma \pi_1 M :_\uparrow v\ [L']}\ (\Sigma E) \quad \frac{[L]\ \Gamma \vdash_\sigma M :_\uparrow \begin{bmatrix} x{:}v \\ v' \end{bmatrix}\ [L'] \quad v'[\pi_1 M/x] \Downarrow_\beta v''}{[L]\ \Gamma \vdash_\sigma \pi_2 M :_\uparrow v''\ [L']}\ (\Sigma E)$$

$$\frac{[L]\ \Gamma \vdash_\sigma A :_\downarrow \mathsf{s}_1\ [L'] \quad [L']\ \Gamma \vdash_\sigma B :_\downarrow \mathsf{s}_2\ [L'']}{[L]\ \Gamma \vdash_\sigma \begin{bmatrix} x{:}A \\ B \end{bmatrix} :_\uparrow \mathsf{s}_2\ [L'']}\ (\to F) \quad (\mathsf{s}_1, \mathsf{s}_2 \in \{\mathsf{type}, \mathsf{kind}\})$$

$$\frac{[L]\ \Gamma \vdash_\sigma M :_\uparrow v\ [L'] \quad [L']\ \Gamma \vdash_\sigma N :_\uparrow v'\ [L'']}{[L]\ \Gamma \vdash_\sigma (M,N) :_\uparrow \begin{bmatrix} v \\ v' \end{bmatrix}\ [L'']}\ (\wedge I) \quad \frac{[L]\ \Gamma \vdash_\sigma M :_\uparrow \begin{bmatrix} v \\ v' \end{bmatrix}\ [L']}{[L]\ \Gamma \vdash_\sigma \pi_1 M :_\uparrow v\ [L']}\ (\wedge E) \quad \frac{[L]\ \Gamma \vdash_\sigma M :_\uparrow \begin{bmatrix} v \\ v' \end{bmatrix}\ [L']}{[L]\ \Gamma \vdash_\sigma \pi_2 M :_\uparrow v'\ [L']}\ (\wedge E)$$

$$\frac{[L]\ \Gamma \vdash_\sigma A :_\downarrow \mathsf{s}\ [L'] \quad A \Downarrow_\beta v \quad [L']\ \Gamma \vdash_\sigma M :_\downarrow v\ [L'']}{[L]\ \Gamma \vdash_\sigma M :: A :_\uparrow v\ [L'']}\ (ANN) \quad (\mathsf{s} \in \{\mathsf{type}, \mathsf{kind}\})$$

$$\frac{}{[L]\ \Gamma \vdash_\sigma \top :_\uparrow \mathsf{type}\ [L]}\ (\top F) \quad \frac{}{[L]\ \Gamma \vdash_\sigma () :_\uparrow \top\ [L]}\ (\top I) \quad \frac{}{[L]\ \Gamma \vdash_\sigma \bot :_\uparrow \mathsf{type}\ [L]}\ (\bot F)$$

$$\frac{}{[L]\ \Gamma \vdash_\sigma @_i :_\downarrow v\ [L, (\Gamma \vdash_\sigma @_i : v)]}\ (ASP)$$

Fig. 5. Type checking/inference rules for DTS

typing rule of dependent type theory—should be proven by a straightforward induction on constructions.

3 Implementation

We implemented the algorithm in Figure 5, using the functional programming language Haskell. The program includes two main functions: `typeInfer` and `typeCheck`. The function `typeInfer` 1) takes a global context Γ and a term M, and returns a type A such that $\Gamma \vdash M : A$; and 2) updates the current list $[L]$ of judgments for underspecified terms. The function `typeCheck` 1) takes a global context Γ, a term M, and a type A, and returns a Boolean value indicating

whether $\Gamma \vdash M : A$ holds; and 2) updates the current list $[L]$ of judgments for underspecified terms.

4 Calculating Projective Contents

Given terms (2) as inputs, our type checking program checks whether it has a type $\delta \to \mathsf{type}$ within a global context $\delta : \mathsf{type},\ c : \delta$ (δ is a propositional content of its preceding discourse and c is its proof term), and updates the list of judgments for underspecified terms so as to contain those in Figure 4. The lists (3) and (4) are, respectively, for $@_2$ and $@_1$, which correspond to the anaphoric part and the possessive presupposition part of *its*. The list (5) is for the appositive CI in (2d).

The judgments in (4) are the ones obtained after anaphora resolution for $@_2$ has been executed: the output of the program contains the occurrence of $@_2$ within the types for $@_1$ (this means that the possessive presuppositions in (1) contain the anaphora antecedents). The proof terms corresponding to the intended reading in (1) (i.e. *it* refers to *Sweden* in (1a)(1b)(1d), and *every monarchy* in (1c)) are as follows:

$$@_2 = \lambda c.(\mathit{sweden}, n(\mathit{sweden})m)$$
$$@_2 = \lambda c.(\pi_1\pi_2(c), n(\pi_1\pi_2(c))(\pi_2\pi_2(c))),$$

where m is a proof term for $\mathbf{monarchy}(\mathit{sweden})$ (*Sweden is a monarchy*), and n is a proof term for $(x : \mathbf{entity}) \to \mathbf{monarchy}(x) \to \neg\mathbf{human}(x)$ (*Monarchy is not a human*). Then, (4) is obtained by substituting $@_2$ with these proof terms.

The judgments for $@_1$ in (4) correctly represent the projective contents in (1): the type of (4a) states that the preceding discourse entails that Sweden has a king (which is a projective content of (1a)). The type of (4b) states that the preceding discourse entails that, if Sweden is a monarchy, then Sweden has a king. (This is a projective content of (1b). The type of (4c) states that the preceding discourse entails that every monarchy has a king (which is a projective content of (1c).) The type of (5) is the projected CI content in (1d), stating that Sweden is a monarchy.

5 Conclusion

We show that the projective contents of (1) are automatically calculated from the semantic representations via the type checking/inference algorithm in Figure 5. In future work, we plan to connect the input of this program to a robust CCG parser, or, independently, to connect the output of this program to a theorem prover, aiming at establishing a pipeline from sentences to their entailment relation, including projective contents such as presuppositions, anaphora and conventional implicatures.

References

1. Asher, N., Luo, Z.: Formalisation of coercions in lexical semantics. In: Sinn und Bedeutung 17. pp. 63–80. Paris (2012)
2. Bekki, D.: Representing anaphora with dependent types. In: Asher, N., Soloviev, S.V. (eds.) Logical Aspects of Computational Linguistics (8th international conference, LACL2014, Toulouse, France, June 2014 Proceedings), LNCS 8535. pp. 14–29. Springer, Heiderburg, Toulouse (2014)
3. Bekki, D., McCready, E.: Ci via dts. In: LENLS11. pp. 110–123. Tokyo (2014)
4. Cooper, R.: Austinian truth, attitudes and type theory. Research on Language and Computation 3, 333–362 (2005)
5. Krahmer, E., Piwek, P.: Presupposition projection as proof construction. In: Bunt, H., Muskens, R. (eds.) Computing Meanings: Current Issues in Computational Semantics. Studies in Linguistics Philosophy Series, Kluwer Academic Publishers, Dordrecht (1999)
6. Löh, A., McBride, C., Swierstra, W.: A tutorial implementation of a dependently typed lambda calculus. Fundamenta Informaticae - Dependently Typed Programming 102(2), 177–207 (2010)
7. Luo, Z.: Common nouns as types. In: Béchet, D., Dikovsky, A. (eds.) Logical Aspects of Computational Linguistics, 7th International Conference, LACL2012, Nantes, France, July 2012 Proceedings, pp. 173–185. Springer (2012)
8. Martin, S., Pollard, C.J.: A dynamic categorial grammar. In: Formal Grammar 19, LNCS 8612 (2014)
9. Martin-Löf, P.: Intuitionistic Type Theory, vol. 17. Italy: Bibliopolis, Naples (1984), sambin, Giovanni (ed.)
10. Mineshima, K.: A presuppositional analysis of definite descriptions in proof theory. In: Satoh, K., Inokuchi, A., Nagao, K., Kawamura, T. (eds.) JSAI 2007 LNAI, vol. 4914, pp. 214–227. Springer, Heidelberg (2008)
11. Piwek, P., Krahmer, E.: Presuppositions in context: Constructing bridges. In: Bonzon, P., Cavalcanti, M., Nossum, R. (eds.) Formal Aspects of Context. Applied Logic Series, Kluwer Academic Publishers, Dordrecht (2000)
12. Ranta, A.: Type-Theoretical Grammar. Oxford University Press (1994)
13. Simons, M., Tonhauser, J., Beaver, D.I., Roberts, C.: What projects and why. In: SALT 20. pp. 309–327 (2010)
14. Sørensen, M., Urzyczyn, P.: Lectures on the Curry–Howard Isomorphism. Studies in Logic and the Foundation of Mathematics, Elsevier (2006)
15. Sundholm, G.: Proof theory and meaning. In: Gabbay, D., Guenthner, F. (eds.) Handbook of Philosophical Logic, vol. III, pp. 471–506. Kluwer, Reidel (1986)
16. Tonhauser, J., Beaver, D.I., Roberts, C., Simons, M.: Towards a taxonomy of projective content. Language 89(1) (2013)