Factivity and Presupposition in Dependent Type Semantics

Koji Mineshima
Ochanomizu University

Joint work with Ribeka Tanaka and Daisuke Bekki

Forum for Theoretical Linguistics, University of Oslo
November 22, 2017

Plan

1. Background: factivity presupposition
2. Introducing dependent types
3. Presupposition
4. Factivity
5. Demo: ccg2lambda + DTS

Ribeka Tanaka, Koji Mineshima and Daisuke Bekki. (2017). Factivity and Presupposition in Dependent Type Semantics. *Journal of Language Modelling*, **5**(2), 385–420.

`http://jlm.ipipan.waw.pl/index.php/JLM/article/view/153`

Factive and non-factive verbs (1)

The contents of the clausal complements of factivity verbs project over entailment-canceling operators (Kiparsky and Kiparsky, 1970)

(1) John knows that Mary is successful.
$\Rightarrow$ Mary is successful.

(2) a. John does not know that Mary is successful. Negation
b. Does John know that Mary is successful? Question
c. If John knows that Mary is successful, ... Conditional

Presuppositions do not survive in certain complex sentences (Filtering)

(3) a. If John is successful, he knows that he is.
b. John is successful and he knows that he is.

In contrast, non-factive verbs do not imply the clausal complements.

(4) John believes that Mary is successful.
$\not\Rightarrow$ Mary is successful.

Factive and non-factive verbs (2)

- Factive and non-factive predicates show different entailment patterns wrt NP-complements of the form *the N* (Vendler, 1972; Ginzburg, 1995; Uegaki, 2016)

(5) a. John believes the rumor that Mary came.
$\Rightarrow$ John believes that Mary came.

b. John knows the rumor that Mary came.
$\not\Rightarrow$ John knows that Mary came.

- Here, *N* is a non-veridical content noun, such as *rumor*, *story*, and *hypothesis*
- But, if *N* is a veridical content noun such as *fact*, the factive verb *know* also licenses the elimination inference:

(6) John knows the fact that Mary came.
$\Leftrightarrow$ John knows that Mary came.

Summary of inference patterns

Entailments ($\Rightarrow$) and presuppositions ($\triangleright$) associated with the factive verb *know* and the non-factive verb *believe*:

K1	x knows that P	$\triangleright$	P
K2	x knows the $N_{nonveridical}$ that P	$\not\Rightarrow$	x knows that P
K3	x knows the $N_{veridical}$ that P	$\Leftrightarrow$	x knows that P
K4	x knows the N that P	$\triangleright$	There is an N that P
B1	x believes that P	$\not\Rightarrow$	P
B2	x believes the N that P	$\Rightarrow$	x believes that P
B3	x believes the N that P	$\triangleright$	There is an N that P

Summary of inference patterns

Entailments ($\Rightarrow$) and presuppositions ($\triangleright$) associated with the factive verb *know* and the non-factive verb *believe*:

K1	x knows that P	$\triangleright$	P
K2	x knows the $N_{nonveridical}$ that P	$\not\Rightarrow$	x knows that P
K3	x knows the $N_{veridical}$ that P	$\Leftrightarrow$	x knows that P
K4	x knows the N that P	$\triangleright$	There is an N that P
B1	x believes that P	$\not\Rightarrow$	P
B2	x believes the N that P	$\Rightarrow$	x believes that P
B3	x believes the N that P	$\triangleright$	There is an N that P

Ultimate goal

1. Build a proof system that derives all the inference patterns given a suitable system of meaning representation
2. Implement it on an automated theorem proving (ATP) system
3. Integrate it with a real parser and compositional semantics
4. Evaluate the resulting system on a shared dataset ('benchmark'): cf. FraCaS (Cooper et al., 1994)

Factive and non-factive verbs (3)

Factive verbs take *wh*-complements, while non-factive verbs do not. (Hintikka, 1975; Karttunen, 1977; Egré, 2008)

(7) a. John knows whether Mary or Bob came.

b. John knows who came.

(8) a. * John believes whether Mary or Bob came.

b. * John believes who came.

Factive and non-factive verbs (4)

Elimination inferences of *wh*-complements (Groenendijk and Stokhof, 1982)

(9) John knows whether Mary or Bob came.
Mary came.
⇒ John knows that Mary came.

(10) John knows who came.
Mary came.
⇒ John knows that Mary came.

Presupposition inferences of *wh*-complements (Hintikka, 1962; Karttunen, 1977)

(11) a. John knows whether Mary or Bob came.
▷ Mary or Bob (but not both) came.

b. John knows who came.
▷ Someone came.

Additional set of inference patterns

Entailments ($\Rightarrow$) and presuppositions ($\rhd$) associated with *wh*-complements of the factive verb *know*

K5	x knows whether A or B, A	$\Rightarrow$	x knows that A
K6	x knows whether A or B, B	$\Rightarrow$	x knows that B
K7	x knows who F, $F(a)$	$\Rightarrow$	x knows that $F(a)$
K8	x knows who F	$\rhd$	someone F
K9	x knows whether A or B	$\rhd$	A or B (but not both)

Tanaka et al. (2017) does not discuss these inferences.

Proof-theoretic approach to factivity inferences

- The standard approach to attitude (factive and non-factive) verbs: Hintikka's possible world semantics
- There has been little attempt to formalize factivity inferences from a proof-theoretical perspective.
- Various proof systems for knowledge and belief have been developed in the context of epistemic logic (Meyer and van der Hoek, 2004).
- But they are mainly concerned with knowledge and belief themselves, not with how they are expressed in natural languages, nor with linguistic phenomena such as factivity presuppositions.
- Our study aims to fill this gap.

Proposal in a nutshell

1. Factive and non-factive verbs select for different semantic objects (Vendler, 1972; Parsons, 1993; Ginzburg, 1995)
   - Non-factive verb *believe* selects for a proposition
   - Factive verb *know* selects for a fact

   In a dependently-typed setting:

   $$\begin{aligned} \text{fact} &\approx \text{proof (evidence) of a proposition} \\ x \text{ knows that } P &\approx x \text{ has a proof (evidence) that } P \end{aligned}$$

2. Combining this idea with the anaphora/presupposition resolution mechanism in DTS.

   $$x \text{ knows that } P \approx x \text{ has } \underbrace{\text{evidence that } P}_{\textit{presupposition}}$$

Digression: Factives in Japanese (1)

Two types of complementizers in Japanese (Kuno, 1973)

1. factive complementizer: *koto-o* introduces a factive complement
2. non-factive complementizer: *to* introduces a non-factive complement

Pietroski (2005):

(12) a. John-wa Mary-ga kita *koto*-o setumeisi-ta.
John-TOP Mary-NOM came COMP-ACC explain-PAST.
'John explained the fact that Mary came.'

b. John-wa Mary-ga kita *to* setumeisi-ta.
John-TOP Mary-NOM came COMP-ACC explain-PAST.
'John explained that Mary came.'

- Only (12a) generates factivity inference.

Factives in Japanese (2)

- Factive verb *siru* 'know' can be combined with both factive and non-factive complementizers.

(13) a. John-wa Mary-ga kita *koto*-o
John-TOP Mary-NOM came COMP-ACC
sit-teiru.
know-RESULT-STATE.
'John knows (the fact) that Mary came.'

b. John-wa Mary-ga kita *to*
John-TOP Mary-NOM came COMP-ACC
sit-teiru.
know-RESULT-STATE.
'John knows that Mary came.'

- Both (13a) and (13b) generate factivity inferences.

Factives in Japanese (3)

- Some people judge that non-factive complementizers *to* in (13b) is less acceptable.
- But, if the verb takes the form *sit-ta* 'know' denoting change of states, non-factive complementizer *to* is perfectly acceptable.

(14) a. John-wa Mary-ga kita *koto-o*
John-TOP Mary-NOM came COMP-ACC
sit-ta.
know-CHANGE-OF-STATE.
'John came to know the fact that Mary came.'

b. John-wa Mary-ga kita *to*
John-TOP Mary-NOM came COMP
sit-ta.
know-CHANGE-OF-STATE.
'John came to know that Mary came.'

Question: Where does the factivity inference in (13a) and (14a) come from? (the factive verb *siru* or the factive complementizer *koto-o*)

Factives in Japanese (4)

- Non-factive verb *sinzi-teiru* 'believe' is combined with both factive complementizer *koto-o* and non-factive complementizer *to*.

(15) a. John-wa Mary-ga kita *koto-o*
John-TOP Mary-NOM came COMP-ACC
sinzi-teiru.
believe-RESULT-STATE.
'John believes the fact that Mary came.'

b. John-wa Mary-ga kita *to*
John-TOP Mary-NOM came COMP
sinzi-teiru.
believe-RESULT-STATE.
'John believes that Mary came.'

Introducing dependent types

Function types — Simple types

f is a function from natural numbers to natural numbers:

$$f : \text{Nat} \to \text{Nat}$$

sort is a function from lists to lists:

$$sort : \text{List} \to \text{List}$$

walk is a function from entities to propositions:

$$walk : \text{Entity} \to \text{Prop}$$

Introducing dependent types

Function types — Simple types

f is a function from natural numbers to natural numbers:

$$f : \text{Nat} \to \text{Nat}$$

sort is a function from lists to lists:

$$sort : \text{List} \to \text{List}$$

walk is a function from entities to propositions:

$$walk : \text{Entity} \to \text{Prop}$$

Function application

$$\frac{f : A \to B \quad a : A}{f(a) : B}$$

Propositions-as-Types principle (Curry-Howard correspondence):

Function type	$\approx$	Implication
$A \to B$		$A \to B$

Generalized function types (Π-types)

f is a function that given a natural number n returns a list of length n

$$f : (\Pi n : \mathrm{Nat})\, \mathrm{List}\, n$$

The range of a function depends on its domain

Generalized function application

$$\frac{f : (\Pi x : A)\, B(x) \qquad a : A}{f(a) : B(a)}$$

Generalized function type	$\approx$	Universal quantifier
$(\Pi x : A)\, B$		$(\forall x : A) B$

Generalized function types (Π-types)

f is a function that given a natural number n returns a list of length n

$$f : (\Pi n : \mathrm{Nat})\, \mathrm{List}\, n$$

The range of a function depends on its domain

Generalized function application

$$\frac{f : (\Pi x : A)\, B(x) \qquad a : A}{f(a) : B(a)}$$

Generalized function type	$\approx$	Universal quantifier
$(\Pi x : A)\, B$		$(\forall x : A) B$

- If the term n does not occur free in the range:

  $$f : (\Pi n : \mathrm{Nat})\, \mathrm{List}$$

  We can simply write:

  $$f : \mathrm{Nat} \to \mathrm{List}$$

- Implication $\to$ is a degenerate form of universal quantifier.

Generalized Pair types (Σ-types)

(a, b) is a pair of entities

$$(a, b) : \text{Entity} \times \text{Entity}$$

Pair type	$\approx$	Conjunction
$A \times B$		$A \wedge B$

Generalized Pair types (Σ-types)

(a, b) is a pair of entities

$$(a, b) : \text{Entity} \times \text{Entity}$$

Pair type	$\approx$	Conjunction
$A \times B$		$A \wedge B$

Generalized pair types (Σ-types)

(n, l_n) is a pair of natural number n and a list of length n.

$$(3, [a, b, c]) : (\Sigma x : \text{Nat})\, \text{List}\, x$$

Generalized pair type	$\approx$	Existential quantifier
$(\Sigma x : A)\, B$		$(\exists x : A)\, B$

- If x does not occur free in B, $(\Sigma x : A)\, B$ is written as $A \wedge B$
- Conjunction is a degenerate form of existential quantifier.
- Projection functions: $\pi_1(a, b) = a$ and $\pi_2(a, b) = b$

Two motivations for dependent types in NLS

1. Rich type structures provide the representation of partial function/predicate
   Cf. Type Presupposition (Asher, 2011)
2. The notion of proof terms
   Representing *contexts* as proof terms (Ranta, 1994)

Rich type structures

Example: predicates relativized to a world:

With simple types, we have:

$$\text{walk} : \text{World} \rightarrow \text{Entity} \rightarrow \text{Prop}$$

With dependent types, we can express a more fine-grained type:

$$\text{walk} : (\Pi w : \text{World})(\text{Entity}_w \rightarrow \text{Prop})$$

walk is a function from worlds w to functions from entities of world w to propositions (variable domain)

Applications to modal subordination (Tanaka et al., 2014)

Partial-function approach

Partial-function approach:

- Presupposition triggers express partial functions.

Heim and Kratzer (1998):

$[\![\texttt{the}]\!]$ = the function such that given a function F of type $e \to t$,

- it returns the unique x such that Fx, if there is exactly one F;
- undefined, otherwise.

$$[\![\texttt{the}]\!] = \lambda F : \underbrace{F \in D_{e \to t} \wedge \exists !\, x(Fx)}_{\text{definedness condition}} . \iota y\, F(y)$$

- If the definedness condition ($\approx$ presupposition) does not hold, the sentence fails to deliver a truth-value

Partial-function approach (cont.)

More generally, presupposition triggers denote partial functions of the form:

$$\lambda x : \varphi(x).\, \psi(x)$$

where $\varphi(x)$ is the definedness condition and $\psi(x)$ is the body of the function.

Questions:

- What is the type of $\lambda x : \varphi(x).\, \psi(x)$?
- What is a proper type system for partial functions?

A suggestive example

1. sort is a function such that given a list x, if there is a total order over x, it returns an ordered permutation of x.

$$\text{sort} : (\Pi x : \text{List})(\text{totally\_ordered}\, x \to \text{List})$$

We can freely mix propositions and types!

A suggestive example

1. sort is a function such that given a list x, if there is a total order over x, it returns an ordered permutation of x.

$$\text{sort} : (\Pi x : \text{List})(\text{totally\_ordered}\, x \to \text{List})$$

We can freely mix propositions and types!

2. index is an indexing function to access the i-th elements of lists of length n:

$$\text{index} : (\Pi i : \text{Nat})(\Pi n : \text{Nat})(i \leq n \to \text{List}\, n \to \text{Entity})$$

Propositions $\text{totally\_ordered}\, x$ and $i \leq n$ imposes a condition on inputs over which the function is defined.

General Scheme

If we have an n-place partial function having "type"

$$A_1 \to \cdots \to A_n \to B,$$

then we can define a predicate Φ that characterizes the inputs over which the function is defined.

The function will then be represented by a dependently typed function with $n+1$ arguments of the form:

$$(\Pi x_1 : A_1) \ldots (\Pi x_n : A_n)(\Phi(x_1, \ldots, x_n) \to B).$$

Example 1 : stop

(16) John stopped smoking.

In simple type theory:

$$\begin{aligned} \text{john} &: \text{entity} \\ \text{smoking} &: \text{entity} \to \text{Prop} \\ \text{stop} &: (\text{entity} \to \text{Prop}) \to \text{entity} \to \text{Prop} \end{aligned}$$

In dependent type theory:

$$\begin{aligned} \text{john} &: \text{entity} \\ \text{smoking} &: \text{entity} \to \text{Prop} \\ \text{used\_to} &: \text{Prop} \to \text{Prop} \\ \text{stop} &: (\Pi F : \text{entity} \to \text{Prop})(\Pi x : \text{entity})(\text{used\_to}\, F(x) \to \text{Prop}) \end{aligned}$$

Example 2. Definite descriptions

In simple type theory:

$$\text{the} : (\text{entity} \to \text{Prop}) \to \text{entity}$$

In dependent type theory:

$$\text{the} : (\Pi F : \text{entity} \to \text{Prop})((\Sigma x : \text{entity})\, F(x) \to \text{entity})$$

Example 3. know

In simple type theory:

$$\text{know} : \text{Prop} \to \text{entity} \to \text{entity}$$

In dependent type theory:

$$\text{know} : (\Pi P : \text{Prop})(\Pi x : \text{entity})(P \to \text{Prop})$$

(17) John knows that it is raining.
$\rightsquigarrow \text{know}([\![\texttt{it is raining}]\!])([\![\texttt{john}]\!])(t) : \text{Prop}$

Here, t is a *proof term* of the proposition $[\![\texttt{it is raining}]\!]$

The notion of proof terms

Propositions-as-types:

$$(\Pi x : \text{Entity})(\text{walk}\, x) : \text{Prop}$$
$$u : (\Pi x : \text{Entity})(\text{walk}\, x)$$

- u is a proof-term of the proposition ‘everyone walks’.
- Predicate walk(x) by itself is a dependent type

Representing contexts as proof terms

Donkey anaphora:

(18) *Someone* entered. *He* smiled.
$\exists x\,(\text{enter}\,(x)) \wedge \text{smile}(x)$

Two sentences are conjoined by Σ-types:

(19) *Someone* entered. *He* smiled.
$(\Sigma u : (\Sigma x : \text{Entity})\,\text{enter}\,(x))\,\text{smile}(\pi_1\,u)$

- The first sentence introduces a pair of an entity x and a proof that x entered.
- Generalization to a wider range of discourse phenomena (Ranta, 1994)

Representing contexts as proof terms

Donkey anaphora:

(18) *Someone* entered. *He* smiled.
$\exists x\,(\text{enter}\,(x)) \wedge \text{smile}(x)$

Two sentences are conjoined by Σ-types:

(19) *Someone* entered. *He* smiled.
$(\Sigma u : (\Sigma x : \text{Entity})\,\text{enter}\,(x))\,\text{smile}(\pi_1\,u)$

- The first sentence introduces a pair of an entity x and a proof that x entered.
- Generalization to a wider range of discourse phenomena (Ranta, 1994)
- What is the semantic representation of the pronoun in @?

(20) *Someone* entered. *He* smiled.
$(\Sigma u : (\Sigma x : \text{Entity})\,\text{enter}\,(x))\,\text{smile}(@)$

- calls for underspecification ($\rightsquigarrow$ Dependent Type Semantics, DTS)

Dependent interpretation of pronouns

- The interpretation of a pronoun can be sensitive to dependency relations between objects. (van den Berg 1996; Krifka 1996; Nouwen 2003; Brasoveanu 2008)

(21) Every[1] boy received a[2] present. They$_1$ opened it$_2$.

- When the first sentence receives subject wide-scope reading ($\forall$-$\exists$ reading), the second sentence receives the interpretation that each boy opened the present he received.

Tanaka et al. (2016):

$$(\Sigma f : [\![\text{Every boy received a present}]\!])\ [\![\text{they opened it}]\!](f)$$

1. The $\forall$-$\exists$ reading of the initial sentence induces a dependency relation between entities. $\rightsquigarrow$ represented by objects of Π-types
2. One needs to construct the referent of the pronoun *it* by using the information from two sentences.

Functional discourse referents (Ranta, 1994)

- Examples attributed to L. Karttunen in Hintikka and Carlson (1979); Ranta (1994)

(22) a. Every boy received a[1] present. Some boy opened it$_1$.

b. If every boy received a[1] present, some boy opened it$_1$.

(23) Every boy received a[1] present. Every young boy opened it$_1$. (van den Berg, 1996)

- A functional discourse referent is passed to the subsequent discourse by means of Σ-types.

$(\Sigma f : [\![\text{every boy received a present}]\!])\, [\![\text{some boy opend it}]\!](f)$

Dependent Type Semantics (DTS)

- ESSLLI2016 course:
  Daisuke Bekki and Koji Mineshima
  An Introduction to Dependent Type Semantics
  `http://esslli2016.unibz.it/?page_id=216`
  `https://www.dropbox.com/sh/u11zu09r9rshsgm/AADq8BmjkBNKOVY8sPmXGo6Ca?dl=0`

Notation

	Martin-Löf	Logic	Agda	DTS
Π-type	$(\Pi x : A)\, B$	$\forall x : A.\, B$	$(x{:}A) \to B$	$(x{:}A) \to B$
Σ-type	$(\Sigma x : A)\, B$	$\exists x : A.\, B$	$(x : A) \times B$	$\begin{bmatrix} x{:}A \\ B \end{bmatrix}$
Function-type Implication	$A \to B$	$A \to B$	$A \to B$	$A \to B$
Product-type Conjunction	$A \wedge B$	$A \wedge B$	$A \times B$	$\begin{bmatrix} A \\ B \end{bmatrix}$

Common nouns : types or predicates?

- Ranta (1994) and Luo (2012a,b) avoid the use of the type **entity** and represent common nouns themselves as types.
- Their approach is called **Modern Type Theory (MTT)**

(24) A man walks.

(25) **MTT**

a. $\begin{bmatrix} x : \text{man} \\ \text{walk}(x) \end{bmatrix}$

b. $(\exists x : \text{man})\, \text{walk}(x)$ notational variant

(26) Signature:

a. man : type

b. walk : man $\rightarrow$ type

Common nouns : types or predicates?

(24) A man walks.

(27) **DTS**

$$\left[\begin{array}{l} u\colon \left[\begin{array}{l} x\colon \text{entity} \\ \text{man}\,(x) \end{array}\right] \\ \text{walk}\,(\pi_1 u) \end{array}\right]$$

(28) Signature:

a. entity : type

b. man : entity $\rightarrow$ type

c. walk : entity $\rightarrow$ type

Common nouns : types or predicates?

(29) Every boy walks.

(30) **MTT**

a. $(x{:}\text{boy}) \rightarrow \text{walk}(x)$

b. $(\forall x : \text{boy})\, \text{walk}(x)$

- boy : type
- walk : boy $\rightarrow$ type

(31) **DTS**

$$\left(u{:}\begin{bmatrix} x{:}\,\text{entity} \\ \text{boy}\,(x) \end{bmatrix}\right) \rightarrow \text{walk}\,(\pi_1 u)$$

- entity : type
- boy : entity $\rightarrow$ type
- walk : entity $\rightarrow$ type

The problem of predicate nominals

How to interprete **predicational sentence** in MTT?

(32) a. John is a student.
b. This is a book.

(33) a. Bob considers Mary a genius.
b. Mary became a doctor.

1. Predicational sentences stand for judgements?

(34) a.	John is a student.	john : student
b.	John is not a student.	**??**
c.	If John is a student, then ...	**??**

- Judgement john : student can neither be negated nor embedded under a logical operator.

The problem of predicate nominals

2. The Russell-Montague's analysis?

(35) a. John is a man. $\begin{bmatrix} x\colon \text{man} \\ \text{john} =_{\text{man}} x \end{bmatrix}$

b. John is not a man. $\neg \begin{bmatrix} x\colon \text{man} \\ \text{john} =_{\text{man}} x \end{bmatrix}$

c. If John is a man, then ... $\begin{bmatrix} x\colon \text{man} \\ \text{john} =_{\text{man}} x \end{bmatrix} \rightarrow \cdots$

The problem of predicate nominals

Problem 1

(36) # John is a man^1. He$_1$ is .... (Kuno, 1970; Mikkelsen, 2005)

$$\left[\begin{array}{l} u\colon \left[\begin{array}{l} x\colon \mathsf{man} \\ \mathsf{john} =_{\mathsf{man}} x \end{array}\right] \\ \ldots \pi_1 u \ldots \end{array}\right]$$

- A predicate nominal does not introduce a discourse referent.
- The Russell-Montague analysis implemented on Σ-types predicts that it does.

The problem of predicate nominals

Problem 2

- john $=_{\text{man}} x$ is well-formed only if john : man is provable.

$$\frac{A : \text{type} \quad t : A \quad u : A}{t =_A u : \text{type}} = F$$

- (35b, c) presuppose that John is a man!

(35) b. John is not a man. $\neg \begin{bmatrix} x\text{: man} \\ \text{john} =_{\text{man}} x \end{bmatrix}$

c. If John is a man, then ... $\begin{bmatrix} x\text{: man} \\ \text{john} =_{\text{man}} x \end{bmatrix} \rightarrow \cdots$

- Chatzikyriakidis and Luo (2016) propose a new semantic for negation and conditional in MTT to avoid this problem.

Contexts as proof terms

(37) A man entered.

$$t : \begin{bmatrix} u{:} \begin{bmatrix} x{:}\,\text{entity} \\ \text{man}(x) \end{bmatrix} \\ \text{enter}(\pi_1 u) \end{bmatrix}$$

A proof term t must be a tuple $((j, p_1), p_2)$ consisting of:

- j : entity
- p_1 : man(j) i.e. a proof that j is a man.
- p_2 : enter(j) i.e. a proof that j entered.

proof terms $\approx$ (structured) assignment functions

Underspecified terms

What is the semantic representation (SR) for (38)?

(38) The elevator is clean.

(39) There is an elevator and it is clean.

Presupposition Assertion

▷ Anaphoric dependency across the two-dimensions

- Presupposition triggers introduce underspecified terms @.
- A term of the form @ A is called type annotation and specifies that the term @ has a type A.

SR for (38) in DTS:

(40) clean $\left(\pi_1 \left(@_i \begin{bmatrix} x\text{: entity} \\ \text{elevator}(x) \end{bmatrix} \right) \right)$

- The annotated type A in $@_i A$ represents the presupposition

Underspecified terms

(40) $\text{clean}\left(\pi_1\left(@_i \begin{bmatrix} x\text{: entity} \\ \text{elevator}(x) \end{bmatrix}\right)\right)$

- For (40) to be well-formed, one needs to construct a term for the type $\begin{bmatrix} x\text{: entity} \\ \text{elevator}(x) \end{bmatrix}$
- Take the first projection π_1 of a pair (x, p) where we have x : entity and p : elevator(x).

Nested presupposition

John's sister is happy.

$$\text{happy}\left(\pi_1\left(@_i\begin{bmatrix} x\text{: entity} \\ \text{sister}(x, \text{john}) \end{bmatrix}\right)\right)$$

John's sister's husband is happy.

$$\text{happy}\left(\pi_1\left(@_1\begin{bmatrix} x\text{: entity} \\ \text{husband}\left(x, \pi_1\left(@_2\begin{bmatrix} y\text{: entity} \\ \text{sister}(y, \text{john} \end{bmatrix}\right)\right)\right) \end{bmatrix}\right)\right)$$

Type checking: an example

The elevator is clean

$$\rightsquigarrow \ \text{clean}\left(\pi_1\left(@_1\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix}\right)\right)$$

$$\frac{}{\text{clean}\left(\pi_1\left(@_1\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix}\right)\right) : \text{type}}\ (\Pi E)$$

Type checking: an example

The elevator is clean

$$\rightsquigarrow \text{clean}\left(\pi_1\left(@_1\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix}\right)\right)$$

$$\dfrac{\dfrac{}{\text{clean} : \text{entity} \to \text{type}}}{\text{clean}\left(\pi_1\left(@_1\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix}\right)\right) : \text{type}}\ (\Pi E)$$

Type checking: an example

The elevator is clean

$$\rightsquigarrow \ \text{clean}\left(\pi_1\left(@_1\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix}\right)\right)$$

$$\dfrac{\dfrac{}{\text{clean} : \text{entity} \to \text{type}} \qquad \dfrac{}{\pi_1\left(@_1\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix}\right) : \text{entity}}\,(\Sigma E)}{\text{clean}\left(\pi_1\left(@_1\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix}\right)\right) : \text{type}}\,(\Pi E)$$

Type checking: an example

The elevator is clean

$$\rightsquigarrow \text{clean}\left(\pi_1\left(@_1\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix}\right)\right)$$

$$\dfrac{\text{clean} : \text{entity} \to \text{type} \qquad \dfrac{\dfrac{}{@_1\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix} : \begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix}}\,(@)}{\pi_1\left(@_1\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix}\right) : \text{entity}}\,(\Sigma E)}{\text{clean}\left(\pi_1\left(@_1\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix}\right)\right) : \text{type}}\,(\Pi E)$$

Type checking: an example

The elevator is clean

$$\rightsquigarrow \ \text{clean}\left(\pi_1\left(@_1\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix}\right)\right)$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix} : \text{type}}\,(@)}{@_1\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix} : \begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix}}\,(\Sigma E) \qquad \dfrac{}{\text{clean} : \text{entity} \to \text{type}}}{\text{clean}\left(\pi_1\left(@_1\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix}\right)\right) : \text{type}}\,(\Pi E)$$

$$\pi_1\left(@_1\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix}\right) : \text{entity}$$

Type checking: an example

The elevator is clean

$$\rightsquigarrow \ \text{clean}\left(\pi_1\left(@_1\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix}\right)\right)$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix} : \text{type}} \qquad \dfrac{\vdots}{? : \begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix}}\,(@)}{@_1\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix} : \begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix}}\,(\Sigma E) \qquad \dfrac{\text{clean} : \text{entity} \to \text{type}}{\pi_1\left(@_1\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix}\right) : \text{entity}}}{\text{clean}\left(\pi_1\left(@_1\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix}\right)\right) : \text{type}}\,(\Pi E)$$

Type checking: an example

The elevator is clean

$$\rightsquigarrow \ \text{clean}\left(\pi_1\left(@_1\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix}\right)\right)$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix} : \text{type}} \qquad \dfrac{\vdots}{? : \begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix}}\,(@)}{@_1\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix} : \begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix}}\,(\Sigma E) \qquad \dfrac{\text{clean} : \text{entity} \to \text{type}}{\pi_1\left(@_1\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix}\right) : \text{entity}}}{\text{clean}\left(\pi_1\left(@_1\begin{bmatrix} x\colon \text{entity} \\ \text{elevator}(x)\end{bmatrix}\right)\right) : \text{type}}\,(\Pi E)$$

Proof search + @-elimination

Suppose we have in a background context $\mathcal{K}$:

$$t : \text{entity},\ u : \text{elevator}(t).$$

Then we can prove:

$$\mathcal{K} \vdash (t, u) : \begin{bmatrix} x\text{: entity} \\ \text{elevator}(x) \end{bmatrix}$$

Replace the @-term with a constructed proof term:

$$\begin{aligned} \text{clean}\left(\pi_1\left(@_1 \begin{bmatrix} x\text{: entity} \\ \text{elevator}(x) \end{bmatrix}\right)\right) &\rightsquigarrow \text{clean}\left(\pi_1(t, u)\right) \\ &\to_\beta \text{clean}\left(t\right) \end{aligned}$$

Projection

- The projection inferences of presupposition can be naturally accounted for using the framework of DTS.
- Consider how to derive the presupposition projected out of negation.
- Note that negation is defined to be an implication of the form $\neg A \equiv A \rightarrow \bot$, where $\bot$ is the absurdity type, i.e., the type that has no inhabitants.
- Given the formation rule for the bottom type shown on the left below, the formation rule for negation can be derived as on the right:

$$\frac{}{\bot : \text{type}}\ (\bot F) \qquad \frac{A : \text{type}}{\neg A : \text{type}}\ (\neg F)$$

Projection

- According to the formation rule $(\neg F)$ for negation, the proposition A and its negation $\neg A$ have the same presupposition.

Example:

The king of France is not bald.

$$\textbf{SR} \quad \neg\text{bald}\left(\pi_1\left(@_1\begin{bmatrix} x\text{: entity} \\ \text{king}(x, f) \end{bmatrix}\right)\right)$$

Projection

Type checking:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\left[\begin{array}{l} x\colon \text{entity} \\ \text{king}(x,f) \end{array}\right] : \text{type} \qquad ? : \left[\begin{array}{l} x\colon \text{entity} \\ \text{king}(x,f) \end{array}\right]}{@_1 \left[\begin{array}{l} x\colon \text{entity} \\ \text{king}(x,f) \end{array}\right] : \left[\begin{array}{l} x\colon \text{entity} \\ \text{king}(x,f) \end{array}\right]}\,(@)}{\pi_1\left(@_1 \left[\begin{array}{l} x\colon \text{entity} \\ \text{king}(x,f) \end{array}\right]\right) : \text{entity}}\,(\Sigma E) \qquad \overline{\text{bald} : \text{entity} \to \text{type}}}{\text{bald}\left(\pi_1\left(@_1 \left[\begin{array}{l} x\colon \text{entity} \\ \text{king}(x,f) \end{array}\right]\right)\right) : \text{type}}\,(\Pi E)}{\neg\text{bald}\left(\pi_1\left(@_1 \left[\begin{array}{l} x\colon \text{entity} \\ \text{king}(x,f) \end{array}\right]\right)\right) : \text{type}}\,(\neg F)$$

Projection

- The same inference is triggered for the antecedent of a conditional sentence like (41):

  (41) If the king of France is wise, people will be happy.

- Note that the formation rule for implication is the following:

$$\frac{A : \text{type} \quad B : \text{type}}{A \to B : \text{type}} \; (\to F)$$

  This is a special case of Π-formation rule:

$$\frac{\begin{array}{c}\overline{u : A} \\ \vdots \\ A : \text{type} \quad B : \text{type}\end{array}}{(u : A) \to B : \text{type}} \; (\Pi F)$$

  If u does not occur in B, Π-formation rule reduces to $\to$-formation rule.

Filtering

- The present account can explain the filtering inference without further stipulation.
- Take a look at the case of a conditional sentence:

(42) If France has a king, **the king of France** is wise.

SR $\left(u{:}\begin{bmatrix} x{:}\ \text{entity} \\ \text{king}(x,f) \end{bmatrix}\right) \rightarrow \text{wise}\left(\pi_1\left(@_1\begin{bmatrix} x{:}\ \text{entity} \\ \text{king}(x,f) \end{bmatrix}\right)\right)$

Filtering

Formation rules for Π-type and Σ-type:

$$\frac{\begin{array}{c}\overline{x : A}\\ \vdots\\ A : \mathrm{type} \quad B(x) : \mathrm{type}\end{array}}{(x{:}A) \to B(x) : \mathrm{type}}\ \Pi F \qquad \frac{\begin{array}{c}\overline{x : A}\\ \vdots\\ A : \mathrm{type} \quad B(x) : \mathrm{type}\end{array}}{\begin{bmatrix} x{:}A \\ B(x) \end{bmatrix} : \mathrm{type}}\ \Sigma F$$

Compare:

$$\frac{\begin{array}{c}\overline{x : A}\\ \vdots\\ A : \mathrm{type} \quad B(x) : \mathrm{type}\end{array}}{\forall x{:}A.\, B(x) : \mathrm{type}} \qquad \frac{\begin{array}{c}\overline{x : A}\\ \vdots\\ A : \mathrm{type} \quad B(x) : \mathrm{type}\end{array}}{\exists x{:}A.\, B(x) : \mathrm{type}}$$

- How presuppositions are inherited can be read off from the formation rules under the propositions-as-types principle!

Filtering

Type checking:

$$\dfrac{\dfrac{\dfrac{\vdots}{\left[\begin{array}{l} x\colon \text{entity} \\ \text{king}(x,f) \end{array}\right] : \text{type}} \quad \dfrac{}{u : \left[\begin{array}{l} x\colon \text{entity} \\ \text{king}(x,f) \end{array}\right]}\,1}{@_1 \left[\begin{array}{l} x\colon \text{entity} \\ \text{king}(x,f) \end{array}\right] : \left[\begin{array}{l} x\colon \text{entity} \\ \text{king}(x,f) \end{array}\right]}\,(@) \qquad \dfrac{}{\text{wise} : \text{entity} \to \text{type}}}{\pi_1 \left(@_1 \left[\begin{array}{l} x\colon \text{entity} \\ \text{king}(x,f) \end{array}\right] \right) : \text{entity}}\,(\Sigma E)}{\dfrac{\dfrac{\vdots}{\left[\begin{array}{l} x\colon \text{entity} \\ \text{king}(x,f) \end{array}\right] : \text{type}} \quad \text{wise} \left(\pi_1 \left(@_1 \left[\begin{array}{l} x\colon \text{entity} \\ \text{king}(x,f) \end{array}\right] \right) \right) : \text{type}}{\left(u\colon \left[\begin{array}{l} x\colon \text{entity} \\ \text{king}(x,f) \end{array}\right] \right) \to \text{wise} \left(\pi_1 \left(@_1 \left[\begin{array}{l} x\colon \text{entity} \\ \text{king}(x,f) \end{array}\right] \right) \right) : \text{type}}\,(\Pi F), 1}\,(\Pi E)$$

Filtering

@-elimination: Replace the term $@_1 \begin{bmatrix} x\colon \text{entity} \\ \text{king}(x, f) \end{bmatrix}$ by u:

$$\dfrac{\dfrac{\vdots}{\begin{bmatrix} x\colon \text{entity} \\ \text{king}(x, f) \end{bmatrix} : \text{type}} \quad \dfrac{}{u : \begin{bmatrix} x\colon \text{entity} \\ \text{king}(x, f) \end{bmatrix}}\,1}{@_1 \begin{bmatrix} x\colon \text{entity} \\ \text{king}(x, f) \end{bmatrix} : \begin{bmatrix} x\colon \text{entity} \\ \text{king}(x, f) \end{bmatrix}}\,(@)$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\begin{bmatrix} x\colon \text{entity} \\ \text{king}(x, f) \end{bmatrix} : \text{type}} \quad \dfrac{}{\text{wise} : \text{entity} \to \text{type}}}{\text{wise}\left(\pi_1\left(@_1 \begin{bmatrix} x\colon \text{entity} \\ \text{king}(x, f) \end{bmatrix}\right)\right) : \text{type}}\,(\Pi F), 1}{\left(u\colon \begin{bmatrix} x\colon \text{entity} \\ \text{king}(x, f) \end{bmatrix}\right) \to \text{wise}\left(\pi_1\left(@_1 \begin{bmatrix} x\colon \text{entity} \\ \text{king}(x, f) \end{bmatrix}\right)\right) : \text{type}}$$

$$\dfrac{\dfrac{@_1 \begin{bmatrix} x\colon \text{entity} \\ \text{king}(x, f) \end{bmatrix} : \begin{bmatrix} x\colon \text{entity} \\ \text{king}(x, f) \end{bmatrix}}{\pi_1\left(@_1 \begin{bmatrix} x\colon \text{entity} \\ \text{king}(x, f) \end{bmatrix}\right) : \text{entity}}\,(\Sigma E)}{\text{wise}\left(\pi_1\left(@_1 \begin{bmatrix} x\colon \text{entity} \\ \text{king}(x, f) \end{bmatrix}\right)\right) : \text{type}}\,(\Pi E)$$

Filtering

@-elimination: Replace the term $@_1 \begin{bmatrix} x\colon \text{entity} \\ \text{king}(x, f) \end{bmatrix}$ by u:

$$\dfrac{\begin{bmatrix} x\colon \text{entity} \\ \text{king}(x, f) \end{bmatrix} : \text{type} \qquad \dfrac{\dfrac{}{\text{wise} : \text{entity} \to \text{type}} \qquad \dfrac{\dfrac{}{u : \begin{bmatrix} x\colon \text{entity} \\ \text{king}(x, f) \end{bmatrix}}\,1}{\pi_1\, u : \text{entity}}\,(\Sigma E)}{\text{wise}\,(\pi_1\, u) : \text{type}}\,(\Pi E)}{\left(u\colon \begin{bmatrix} x\colon \text{entity} \\ \text{king}(x, f) \end{bmatrix}\right) \to \text{wise}\,(\pi_1\, u) : \text{type}}\,(\Pi F), 1$$

Factive presupposition

Factive and non-factive predicates have different semantic types:

- A non-factive predicate like *believe* takes an entity and a proposition (= type) as argument:

$$\text{believe} : \text{type} \to \text{entity} \to \text{type}$$

- A factive predicate takes a proof term for the embedded proposition as an extra argument:

$$\text{know} : (p\colon \text{type}) \to \text{entity} \to p \to \text{type}$$

The SR know$(a, (P, e))$ can read as:

the agent a obtains evidence e of the proposition P

Factive presupposition

Lexical entry for *know*:
$\lambda p.\lambda x.\,\mathrm{know}\,(x,(p,@_i p)) : (S\backslash NP)/S$

- The argument p is a proposition expressed by the complement of *know*.
- The underspecified term $@_i p$ requires to construct a proof term of p, i.e., to find evidence of the proposition p.
- If such a proof term is constructed, it fills the third argument position of the predicate *know*.
- In sum, the sentence *x knows that P* presupposes that there is a proof (evidence) of P and asserts that the agent x obtains it, i.e., x has a proof (evidence) of the proposition P.

Factive presupposition

(43) John knows that Mary came.

CCG-Parsing and semantic composition:

$$\dfrac{\dfrac{\text{John}}{NP}}{\dfrac{\dfrac{\dfrac{\text{knows}}{(S\backslash NP)/S}}{S\backslash NP}>}{S}<}$$

$$\begin{array}{l} \text{John}: j \\ \text{knows}: \lambda p.\lambda x.\,\text{know}\,(x,(p,@_1 p)) \\ \text{that}: \lambda p.p \\ \text{Mary came}: \text{came}(m) \\ \text{Mary came}: \lambda x.\,\text{know}\,(x,(\text{came}(m),@_1\text{came}(m))) \\ \text{John knows that Mary came}: \text{know}\,(j,(\text{came}(m),@_1\text{came}(m))) \end{array}$$

Factive presupposition

(43) John knows that Mary came.

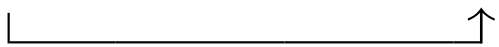
$$\textbf{SR}\ \ \text{know}\left(j, (\text{came}(m), @_1\ \text{came}(m))\right)$$

- The underspecified term $@_1$ has the annotated type came(m).
- Thus, in the same way as the examples we saw before, presuppositional inferences triggered by factive predicates can be derived as type checking derivations.
- It is easily checked that this is the case even when the factive predicate appears in negated sentences and conditionals .
- The projection and filtering inferences can be accounted for in the same way as before.

Binding problem

- A sentence like (44) is known to pose the so-called *binding problem* (Karttunen, 1971; Karttunen and Peters, 1979; Cooper, 1983).

(44) A student **regrets** that he talked.



Presupposition: $\exists x(\text{student}(x) \wedge \text{talk}(x))$
Assertion: $\exists x(\text{student}(x) \wedge \text{regret}(x, \text{talk}(x)))$

- The complement clause of the factive predicate *regret* contains the pronoun *he* which is bound by the outside quantifier *a student*.
- If one separates the assertion and presupposition, any student who talked can satisfy the presupposition, independently of the assertive content.
- In a standard setting, there is no way to bind a variable in both assertion and presupposition with the same quantifier.

Binding problem

- DTS gives a solution to this problem.

A student regrets that he talked.

SR
$$\left[\begin{array}{l} u: \left[\begin{array}{l} x\text{: entity} \\ \text{student}(x) \end{array}\right] \\ \text{regret}\left(\pi_1 u, @_2\left(\text{talk}\left(\pi_1\left(@_1\left[\begin{array}{l} x\text{: entity} \\ \text{male}(x) \end{array}\right]\right)\right)\right)\right) \end{array}\right]$$

- In this SR, the underspecified term $@_1$ occurs inside the type annotated to the underspecified term $@_2$
- For this SR to be well-formed, one first needs to resolve the inside underspecified term $@_1$:

$$u : \left[\begin{array}{l} x\text{: entity} \\ \text{student}(x) \end{array}\right] \vdash ? : \left[\begin{array}{l} x\text{: entity} \\ \text{male}(x) \end{array}\right]$$

Binding problem

- This makes a prediction that (44) presupposes that every student (in a given context) is male.
- This prediction is similar to Heim's (1983) dynamic semantics.
- Suppose that we have the following in the global context:

$$f : \left(u{:} \begin{bmatrix} x{:}\ \text{entity} \\ \text{student}(x) \end{bmatrix} \right) \to \text{male}(\pi_1 u)$$

Then we can construct a term for $@_1$ as:

$$\lambda c.\, (\pi_1(\pi_2 c), f(\pi_2 c))$$

- By substituting $@_1$ by @-elimination, we get:

$$\begin{bmatrix} u{:} \begin{bmatrix} x{:}\ \text{entity} \\ \text{student}(x) \end{bmatrix} \\ \text{regret}\,(\pi_1 u, @_2\,(\text{talk}\,(\pi_1 u))) \end{bmatrix}$$

Binding problem

- Finally, one needs to resolve the underspecified term $@_2$:

$$u : \begin{bmatrix} x\colon \text{entity} \\ \text{student}(x) \end{bmatrix} \vdash \; ? \; : \text{talk}(\pi_1 u)$$

- Suppose that the following is in the signature:

$$g : \left(u\colon \begin{bmatrix} x\colon \text{entity} \\ \text{student}(x) \end{bmatrix} \right) \to \text{talk}(\pi_1 u)$$

  Then we get a term for $@_2$ as gu.
- By substituting $@_2$ by gu, we get the final SR:

$$\begin{bmatrix} u\colon \begin{bmatrix} x\colon \text{entity} \\ \text{student}(x) \end{bmatrix} \\ \text{regret}\,(\pi_1 u, gu) \end{bmatrix}$$

- A quantifier (Σ-type or Π-type) in the asserted content can bind a variable in the presupposed content annotated to an underspecified term.

NP-complements

Ambiguity of *know* (cf. Uegaki, 2016)

- *evidence*-taking reading: know

$$\text{know} : \begin{bmatrix} \text{entity} \\ \begin{bmatrix} p\text{: type} \\ p \end{bmatrix} \end{bmatrix} \rightarrow \text{type}$$

- *acquaintance* reading: know_{np}

$$\text{know}_{np} : \begin{bmatrix} \text{entity} \\ \text{entity} \end{bmatrix} \rightarrow \text{type}$$

(45) a. John knows the man.

b. $\text{know}_{np}(x, \pi_1(@_i \begin{bmatrix} x\text{: entity} \\ \text{man}(x) \end{bmatrix}))$

(46) a. John knows the rumor that S.

b. $\text{know}_{np}(x, \pi_1(@_i \begin{bmatrix} p\text{: type} \\ \begin{bmatrix} p = S \\ \text{rumor}(p) \end{bmatrix} \end{bmatrix}))$

NP-complements

(47) x believes the rumor that S. $\Rightarrow$ x believes that S.

a. $\text{believe}(x, \pi_1(@_i \begin{bmatrix} p\text{: type} \\ \begin{bmatrix} p = S \\ \text{rumor}(p) \end{bmatrix} \end{bmatrix})) \equiv$ $\text{believe}(x, S)$

b. $\text{believe}(x, S)$

(48) x knows the rumor that S. $\not\Rightarrow$ x knows that S.

a. $\text{know}_{np}(x, \pi_1(@_i \begin{bmatrix} p\text{: type} \\ \begin{bmatrix} p = S \\ \text{rumor}(p) \end{bmatrix} \end{bmatrix})) \equiv$ $\text{know}_{np}(x, S)$

b. $\text{know}(x, @_i S)$

See Tanaka et al. (2017) for more details.

System demonstration — Implementation of DTS

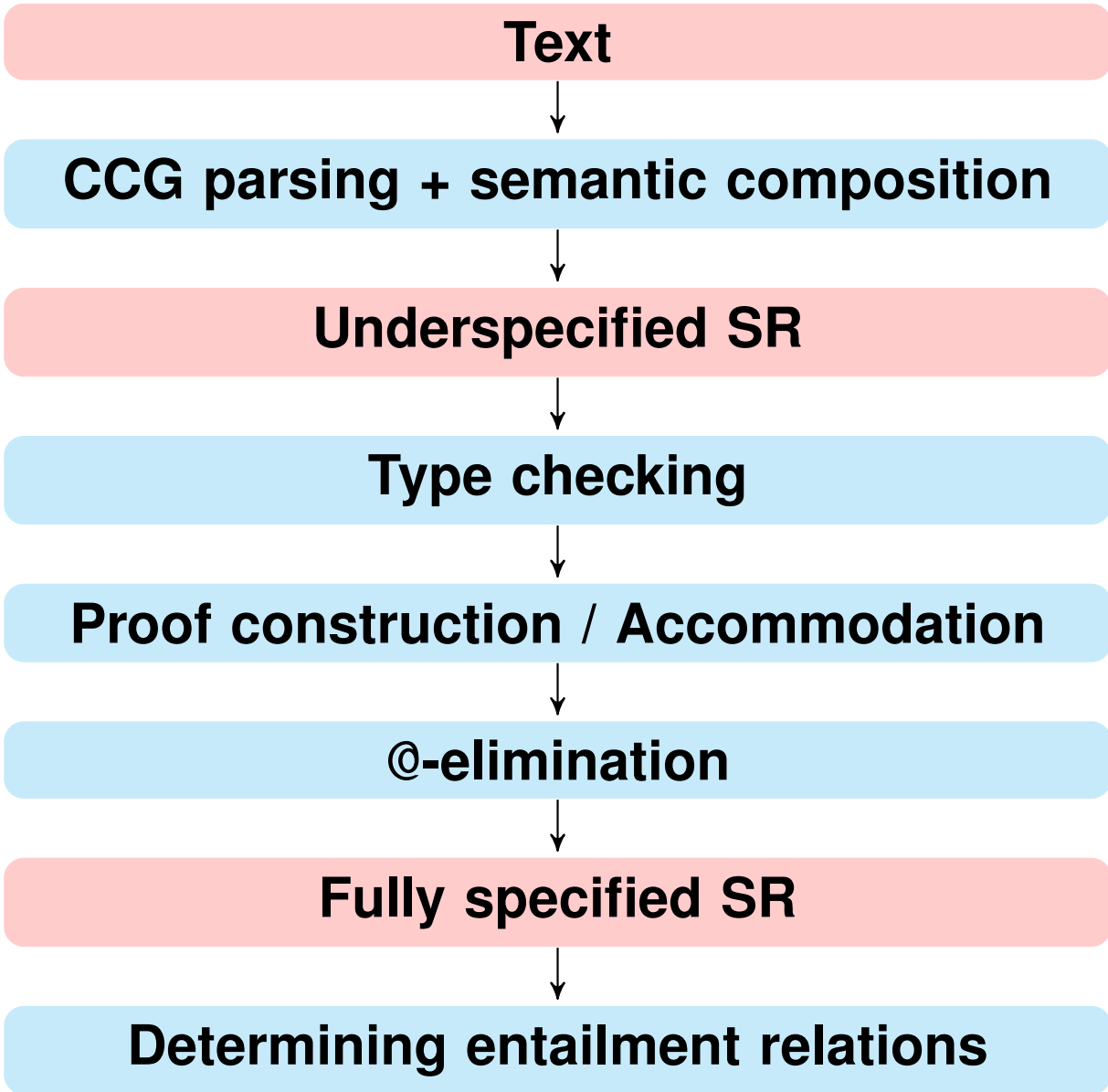
ccg2lambda + DTS for Japanese and English

`https://github.com/mynlp/ccg2lambda`

Grammar/parser learning from treebank

- English CCGbank (Hockenmaier and Steedman, 2007)
- Japanese CCGbank (Uematsu et al., 2015)

Full Pipeline



Summary

- Applying DTS to the analysis of factive presuppositions.
- Common nouns are not types but predicates.
- In DTS, proofs are taken as first-class objects
  $\Rightarrow$ Factive verbs are predicates over proof terms, while non-factive verbs are predicates over propositions.
- The notion of proof terms enables the unified treatment of factive presupposition and existence presupposition:
  $\Rightarrow$ the same proof-construction mechanism underlies the presuppositions of factive verbs (eg. *know that S*) and definite description (*the F*).
- DTS avoids the Binding problem
- An extension to *wh*-complements and NP-complements

Reference I

Asher, N. (2011) *Lexical Meaning in Context: A Web of Words*. Cambridge, Cambridge University Press.

Chatzikyriakidis, S. and Z. Luo. (2016) “On the Interpretation of Common Nouns: Types v.s. Predicates”, In: *Modern Perspectives in Type Theoretical Semantics*, Studies of Linguistics and Philosophy. Springer.

Cooper, R. (1983) *Quantification and Syntactic Theory*. Dordrecht, Reidel.

Cooper, R., R. Crouch, J. van Eijck, C. Fox, J. van Genabith, J. Jaspers, H. Kamp, M. Pinkal, M. Poesio, S. Pulman, et al. (1994) “FraCaS–A Framework for Computational Semantics”, *Deliverable* **D6**.

Egré, P. (2008) “Question-embedding and factivity”, *Grazer Philosophische Studien* **77**(1), pp.85–125.

Ginzburg, J. (1995) “Resolving Questions, I”, *Linguistics and Philosophy* **18**(5), pp.459–527.

Groenendijk, J. and M. Stokhof. (1982) “Semantic Analysis of *wh*-complements”, *Linguistics and Philosophy* **5**(2), pp.175–233.

Hintikka, J. (1962) *Knowledge and Belief*. Cornell University Press.

Reference II

Hintikka, J. (1975) “Different Constructions in Terms of the Basic Epistemological Verbs”, In: *The Intentions of Intensionality*. Kluwer, pp.1–25.

Hintikka, J. and L. Carlson. (1979) “Conditionals, Generic Quantifiers, and Other Applications of Subgames”, In: Esa Saarinen (ed.): *Game-Theoretical Semantics*, Vol. 5 of *Studies in Linguistics and Philosophy*. Springer Netherlands.

Hockenmaier, J. and M. Steedman. (2007) “CCGbank: a corpus of CCG derivations and dependency structures extracted from the Penn Treebank”, *Computational Linguistics* **33**(3), pp.355–396.

Karttunen, L. (1971) “Implicative Verbs”, *Language* **47**(2), pp.340–358.

Karttunen, L. (1977) “Syntax and Semantics of Questions”, *Linguistics and Philosophy* **1**(1), pp.3–44.

Karttunen, L. and S. Peters. (1979) “Conventional Implicatures”, In: C. K. Oh and D. A. Dineen (eds.): *Syntax and Semantics 11: Presupposition*. New York, Academic Press, pp.1–56.

Kiparsky, P. and C. Kiparsky. (1970) “Fact”, In: M. Bierwisch and K. E. Heidolph (eds.): *Progress in Linguistics*. de Gruyter Mouton, pp.143–173.

Reference III

Krahmer, E. and P. Piwek. (1999) “Presupposition Projection as Proof Construction”, In: H. Bunt and R. Muskens (eds.): *Computing Meaning*, Vol. 1. Dordrecht, Kluwer, pp.281–300.

Kuno, S. (1970) “Some Properties of Non-referential Noun Phrases”, In: R. Jakobson and S. Kawamoto (eds.): *Studies in General and Oriental Linguistics. Presented to S. Hattori on Occasion of his Sixtieth Birthday*. Tokyo, TEC, pp.348–373.

Kuno, S. (1973) *The Structure of the Japanese Language*. The MIT Press.

Luo, Z. (2012a) “Common nouns as types”, In the Proceedings of *Logical Aspects of Computational Linguistics*. pp.173–185.

Luo, Z. (2012b) “Formal semantics in modern type theories with coercive subtyping”, *Linguistics and Philosophy* **35**(6), pp.491–513.

Meyer, J.-J. and W. van der Hoek. (2004) *Epistemic Logic for AI and Computer Science*. Cambridge University Press.

Mikkelsen, L. (2005) *Copular Clauses. Specification, Predication and Equation*. Amsterdam, Benjamins.

Parsons, T. (1993) “On Denoting Propositions and Facts”, *Philosophical Perspectives* **7**, pp.441–460.

Reference IV

Pietroski, P. M. (2005) *Events and semantic architecture*. Oxford University Press.

Ranta, A. (1994) *Type-Theoretical Grammar*. Oxford, Oxford University Press.

Tanaka, R., K. Mineshima, and D. Bekki. (2014) “Resolving modal anaphora in Dependent Type Semantics”, In the Proceedings of *JSAI International Symposium on Artificial Intelligence*. pp.83–98.

Tanaka, R., K. Mineshima, and D. Bekki. (2016) “On the Interpretation of Dependent Plural Anaphora in a Dependently-Typed Setting”, In the Proceedings of *JSAI International Symposium on Artificial Intelligence*. pp.123–137.

Tanaka, R., K. Mineshima, and D. Bekki. (2017) “Factivity and presupposition in Dependent Type Semantics”, *Journal of Language Modelling* **5**(2), pp.385–420. `http://jlm.ipipan.waw.pl/index.php/JLM/article/view/153`.

Uegaki, W. (2016) “Content Nouns and the Semantics of Question-embedding”, *Journal of Semantics* **33**(4), pp.623–660.

Reference V

Uematsu, S., T. Matsuzaki, H. Hanaoka, Y. Miyao, and H. Mima. (2015) “Integrating multiple dependency corpora for inducing wide-coverage Japanese CCG resources”, *ACM Transactions on Asian and Low-Resource Language Information Processing* **14**(1), pp.1–24.

van den Berg, M. (1996) “Some aspects of the Internal Structure of Discourse. The Dynamics of Nominal Anaphora”, PhD thesis, University of Amsterdam.

Vendler, Z. (1972) *Res Cogitans: An Essay in Rational Psychology*. Cornell University Press.