Modern Type Theories for NL Semantics

Zhaohui Luo
Royal Holloway, Univ. of London
Natural Language Semantics

- Semantics – study of meaning (communicate = convey meaning)

- Various kinds of theories of meaning
  - Meaning is reference (“referential theory”)
    - Word meanings are things (abstract/concrete) in the world.
    - c.f., Plato, ...
  - Meaning is concept (“internalist theory”)
    - Word meanings are ideas in the mind.
    - c.f., Aristotle, ..., Chomsky.
  - Meaning is use (“use theory”)
    - Word meanings are understood by their uses.
    - c.f., Wittgenstein, ..., Dummett, Brandom.
Formal semantics

❖ Model-theoretic semantics
  ❖ Meaning is given by denotation.
  ❖ c.f., Tarski, ..., Montague.
  ❖ e.g., Montague grammar (MG)
     ❖ NL $\rightarrow$ simple type theory $\rightarrow$ set theory

❖ Proof-theoretic semantics
  ❖ In logics, meaning is inferential use (proof/consequence).
  ❖ c.f., Gentzen, Prawitz, ..., Martin-Löf.
  ❖ e.g., Martin-Löf’s meaning theory
Simple example for MTS and PTS

- Model-theoretic semantics
  - John is happy. \( \Rightarrow \) happy(john)
  - John is a member of the set of entities that are happy.
  - Montague’s semantics is model-theoretic – it has a wide coverage (powerful).

- Proof-theoretic semantics
  - How to understand a proposition like happy(john)?
  - In logic, its meaning can be characterised by its uses – two respects:
    - How it can be arrived at (proved)?
    - How it can be used to lead to other consequences?

(*)
Montague’s semantics and MTT-semantics

- **Formal semantics (MG)**
  - Montague Grammar Church’s simple type theory (Montague, 1930–1971), dominating in linguistic semantics since 1970s
  - Other development of formal semantics in last decades (e.g., Discourse Representation Theory & Situation Semantics)

- **MTT-semantics: formal semantics in modern type theories**
  - Early use of dependent type theory in formal semantics (cf, Ranta 1994)
  - Recent development (since 2009) – full-scale alternative to MG
  - Advantages: both model/proof-theoretic, proof technological support, ...
  - Refs at [http://www.cs.rhul.ac.uk/home/zhaohui/lexsem.html](http://www.cs.rhul.ac.uk/home/zhaohui/lexsem.html), including
    - Chatzikyriakidis and Luo (eds.) Modern Perspectives in Type Theoretical Semantics. Springer, 2017. (Collection on rich typing in NL semantics)
TTs as foundational languages for NL semantics

What is a type theory?
- \( a : A \)
  - \( a \) is an object of type \( A \)
  - the most basic “judgement” to make in type theory

The worlds of types – examples:
- Simply typed \( \lambda \)-calculus (with \( A \rightarrow B \))
- Church’s simply type theory as in Montague’s semantics (\( A \rightarrow B \) with HOL of formulas like \( P \supset Q \) and \( \forall x:A.P \))
- Richer types (eg, in MTTs: dependent, inductive, ...; see latter)

Logical language (often part of type theory)
- In Church/Montague: formulas & provability/truth
- In modern type theories (MTTs): formulas-as-types & proofs-as-objects
  E.g., \( \forall x:\text{Man. handsome}(x) \rightarrow \neg \text{ugly}(x) \) can be seen as a type (later)
What typing is not:

- “a : A” is not a logical formula.
  - 7 : Nat, j : Man, ...
  - Different from logical formulae nat(7)/man(j), where nat/man are predicates. (Note: whether a formula is true is undecidable, while the : judgement is.)
- “a : A” is different from the set-theoretic membership relation “a \in S” (the latter is a logical formula in FOL).

What typing is related to (some example notions):

- Meaningfulness (ill-typed $\Rightarrow$ meaningless)
- Semantic/category errors (eg, “A table talks.” – later)
- Type presuppositions (Asher 2011)
This course – MTTs in NL semantics

- **MTTs – Modern Type Theories**
  - Rich type structures
    - much richer than simple type theory in MG
  - Proof-theoretically specified by rules
    - proof-theoretic meanings (e.g., Martin-Löf’s meaning theory)
  - Embedded logic
    - based on propositions-as-types principle

- **Informally, MTTs, for NL semantics, offer**
  - “Real-world” modelling as in model-theoretic semantics
  - Effective inference based on proof-theoretic semantics

*Remark: New perspective & new possibility not available before!*
An episode: MTT-based technology and applications

- Proof technology based on type theories
  - Proof assistants
    - MTT-based: ALF/Agda, Coq, Lego, NuPRL, Plastic, ...
    - HOL-based: Isabelle, HOL, ...

- Applications of proof assistants
  - Math: formalisation of mathematics – eg,
    - 4-colour theorem (on map colouring) in Coq
    - Kepler conjecture (on sphere packing) in Isabelle/HOL
  - Computer Science:
    - program verification and advanced programming
  - Computational Linguistics
    - E.g., MTT-sem based NL reasoning in Coq (Chatzikyriakidis & Luo 2014)
A focus of the course

- However, this course
  - is not one on MTT-semantics only;
  - is one on MTTs with examples in MTT-semantics!

- Reason for this focus:
  - Learning MTTs is laborious, even for logic-oriented semanticists
  - New logical concepts: judgement, context, inductive & dependent types, universe, subtyping, ...
  - Hope: making learning MTTs (hence MTT-semantics) easier!

- Goal: learning MTTs as well as MTT-semantics
Overview of the Course

❖ This lecture:
  ❖ Introduction to MTT-semantics (a first taste)

❖ Each lecture from L2-5 will consist of two parts:
  ❖ Some key MTT concepts/mechanisms
  ❖ Introduction of some MTT types with several applications in MTT-semantics.
  ❖ Example: Lecture 2 of “Judgements and Π-polymorphism” introduces these in MTTs and then uses Π-polymorphism to model coordination, predicate-modifying adverbs (quickly) and subsective adjectives (large).

❖ Goal: learn MTTs with examples in MTT-semantics
Material available on the web:

- Lecture slides
- Course proposal (good summary, but the organisation and descriptions of lectures are )
- Papers/books on MTT-semantics available at
  [http://www.cs.rhul.ac.uk/home/zhaohui/lexsem.html](http://www.cs.rhul.ac.uk/home/zhaohui/lexsem.html)
I. Type-theoretical semantics: introduction

- Introduction to MG and MTT-semantics, starting with examples
- Two basic semantic types in MG/MTT-semantics

<table>
<thead>
<tr>
<th>Category</th>
<th>MG’s type</th>
<th>MTT-semantic type</th>
</tr>
</thead>
<tbody>
<tr>
<td>S (sentence)</td>
<td>t</td>
<td>Prop</td>
</tr>
<tr>
<td>IV (verb)</td>
<td>e→t</td>
<td>A→Prop (A: “meaningful domain”)</td>
</tr>
</tbody>
</table>
Simple example

- John talks.
  - Sentences are (interpreted as) logical propositions.
  - Individuals are entities or objects in certain domains.
  - Verbs are predicates over entities or certain domains.

<table>
<thead>
<tr>
<th></th>
<th>Montague</th>
<th>MTT-semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>john</td>
<td>e</td>
<td>Human</td>
</tr>
<tr>
<td>talk</td>
<td>e→t</td>
<td>Human→Prop</td>
</tr>
<tr>
<td>talk(john)</td>
<td>t</td>
<td>Prop</td>
</tr>
</tbody>
</table>
Three issues: a first taste

❖ Selection restriction
  ❖ (*) The table talks.
  ❖ Is (*) meaningful?
  ❖ In MG, yes: (*) has a truth value
    ❖ talk(the table) is false in the intended model.
  ❖ In MTT-semantics, no: (*) is not meaningful
    ❖ since “the table” : Table and it is not of type Human and, hence,
      talk(the table) is ill-typed as talk requires that its argument be of type
      Human.
    ❖ So, in MTT-semantics, meaningfulness = well-typedness
Subtyping

- Necessary for a multi-type language such as MTTs
- Example: What if John is a man in “John talks”?
  - john : Man
  - talk : Human \rightarrow Prop
  - talk(john)? (john is not of type Human ...?)
- Problem solved if Man ≤ Human
  - A ≤ B and a : A \rightarrow a : B
  - Man ≤ Human and john : Man \rightarrow john : Human
  - Hence, talk(john) : Prop

Later (Lecture 3): “coercive subtyping”, and we use it in modelling various linguistic features such as sense selection & copredication.
Propositions as types in MTTs

- Formula $A$ is provable/true if, and only if, there is a proof of $A$, i.e., an object $p$ of type $A$ ($p : A$).

<table>
<thead>
<tr>
<th>formula</th>
<th>type</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \Rightarrow B$</td>
<td>$A \rightarrow B$</td>
<td>If ..., then ...</td>
</tr>
<tr>
<td>$\forall x : A. B(x)$</td>
<td>$\Pi x : A. B(x)$</td>
<td>Every man is handsome.</td>
</tr>
</tbody>
</table>

MTTs have a consistent logic based on the propositions-as-types principle.
Two more basic MG/MTT-semantic types

<table>
<thead>
<tr>
<th>Category</th>
<th>MG’s Type</th>
<th>MTT-semantic type</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>t</td>
<td>Prop</td>
</tr>
<tr>
<td>IV</td>
<td>e→t</td>
<td>A→Prop</td>
</tr>
<tr>
<td>CN (book, man)</td>
<td>e→t</td>
<td>types (Book, ∑x:Man.handsome(x))</td>
</tr>
<tr>
<td>Adj (CN/CN)</td>
<td>(e→t)→(e→t) or e→t</td>
<td>A→Prop (A: meaningful domain)</td>
</tr>
</tbody>
</table>
Adjective modifications of CNs

One of the possible/classical classifications:

<table>
<thead>
<tr>
<th>classification</th>
<th>property</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intersective</td>
<td>Adj(N) $\Rightarrow$ Adj &amp; N</td>
<td>handsome man</td>
</tr>
<tr>
<td>Subsectional</td>
<td>Adj(N) $\Rightarrow$ N</td>
<td>large mouse</td>
</tr>
<tr>
<td>Privative</td>
<td>Adj(N) $\Rightarrow$ $\neg$N</td>
<td>fake gun</td>
</tr>
<tr>
<td>Non-committal</td>
<td>Adj(N) $\Rightarrow$ ?</td>
<td>alleged criminal</td>
</tr>
</tbody>
</table>
Intersective adjectives

- Example: handsome man

<table>
<thead>
<tr>
<th></th>
<th>Montague</th>
<th>MTT-semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>man</td>
<td>man : e→t</td>
<td>Man : Type</td>
</tr>
<tr>
<td>handsome</td>
<td>handsome : e→t</td>
<td>Man→Prop</td>
</tr>
<tr>
<td>handsome man</td>
<td>(\lambda x.) man(x) &amp; handsome(x)</td>
<td>(\Sigma)(Man,handsome)</td>
</tr>
</tbody>
</table>

- In general:

<table>
<thead>
<tr>
<th></th>
<th>Montague</th>
<th>MTT-semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNs</td>
<td>predicates</td>
<td>types</td>
</tr>
<tr>
<td>Adjectives</td>
<td>predicates</td>
<td>predicates</td>
</tr>
<tr>
<td>CNs modified by intersective adj</td>
<td>Predicate by conjunction</td>
<td>(\Sigma)-type</td>
</tr>
</tbody>
</table>
adjective : CNs → CNs

- In MG, predicates to predicates.
- In MTT-semantics, types to types.

Proposals in MTT-sem (Chatzikyriakidis & Luo, FG13 & JoLLI17)

<table>
<thead>
<tr>
<th>classification</th>
<th>example</th>
<th>types employed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intersective</td>
<td>handsome man</td>
<td>Σ-types (of pairs)</td>
</tr>
<tr>
<td>Subsectional</td>
<td>large mouse</td>
<td>Π-types (polymorphism)</td>
</tr>
<tr>
<td>Privative</td>
<td>fake gun</td>
<td>disjoint union types</td>
</tr>
<tr>
<td>Non-committal</td>
<td>alleged criminal</td>
<td>belief contexts</td>
</tr>
</tbody>
</table>
**Σ-types: a taste of dependent types**

- First, we start with “product types” of pairs:
  - A x B of pairs (a,b) such that a:A and b:B
  - Rules to specify these product types:
    - Formation rule for A x B
    - Introduction rule for pairs (a,b) : A x B
    - Elimination rules for projections \( \pi_1(p) \) and \( \pi_2(p) \)
    - Computation rule: \( \pi_1(a,b)=a \) and \( \pi_2(a,b)=b \).

- This generalises to Σ-types of “dependent pairs” (next page)
“Family” of types
- Type-valued function
  - Dog(John) = \{d\}, Dog(Mary)={d_1, d_2}, ...
  - Dog : Human \rightarrow \text{Type}

\Sigma\text{-types of “dependent pairs”:}
- \Sigma(A,B) of dependent pairs (a,b) such that a:A and b:B(a), where A:Type and B : A \rightarrow \text{Type}.

Rules for \Sigma\text{-types:}
- Formation rule for \Sigma(A,B) for B : A \rightarrow \text{Type}
- Introduction rule for dependent pairs (a,b) : \Sigma(A,B)
- Elimination rules for projections \( \pi_1(p) : A \) and \( \pi_2(p) : B(\pi_1(p)) \)
- Computation rule: \( \pi_1(a,b)=a \) and \( \pi_2(a,b)=b \).
“handsome man” is interpreted as type \( \Sigma(\text{Man}, \text{handsome}) \)

So,
- A handsome man is an object of the above type
- It is a pair \((m, p)\) such that \(m : \text{Man}\) and \(p : \text{handsome}(m)\), i.e., \(m\) is a man and \(p\) is a proof that \(m\) is handsome.
II. Judgements and $\Pi$-polymorphism

II.1. Overview of Modern Type Theories
- Difference from simple type theory
- Example MTTs
- Judgements (basic “statements” in MTTs)

II.2. Dependent product types ($\Pi$-types)
- Basic constructions
- $\rightarrow$-types as special cases of $\Pi$-types (examples in semantics)

II.3. Universes – $\Pi$-polymorphism and examples like
- Coordination
- Quantifiers and Adverbs (predicate modifying)
- Subsective adjectives (e.g., large)
II.1. Modern Type Theories: overview

- Simple v.s. Modern Type Theories
- Church’s simple type theory (1940)
  - As in Montague semantics
  - Types ("single-sorted"): e, t, e→t, ...
  - HOL (e.g., membership of `sets’)
- Modern type theories
  - Many types of entities – “many-sorted”
    - Table, Man, Human, $\sum x: \text{Man}.\text{handsome}(x)$, Phy•Info, ...
  - Dependent types: “types segmented by indexes”
    - List $\rightarrow$ Vect(n) with n:Nat (lists of length n)
    - Event $\rightarrow$ Evt(h) with h:Human (events performed by h)
- Examples of MTTs:
  - Martin-Löf’s TT (predicative; non-standard FOL; proof assistants Agda/NuPRL)
  - CICp (Coq) & UTT (Luo 1994) (impredicative; HOL; Coq/Lego/Plastic/Matita)
Predicativity/impredicativity: technical jargon

- This refers to a possibility of forming a logical proposition “circularly”:
  - $\forall X: \text{Prop}.X : \text{Prop}$
  - Quantifying over all propositions to form a new proposition.
  - Is this OK? Martin-Löf thinks not, while Ramsey (1926) thinks yes (it is circular, but it is not vicious.)

- Allowing the above leads to impredicative type theories, which have in particular, Prop:
  - Impredicative universe of logical propositions (cf, t in MG)
  - Internal totality (a type, and can hence form types, eg Table $\rightarrow \text{Prop}$, Man $\rightarrow \text{Prop}$, $\forall X: \text{Prop}.X, ...$)
A statement in an MTT is a judgement, one of whose forms (the most important form) is

\((*)\) \(\Gamma \vdash a : A\)

which says that “\(a\) is of type \(A\) under context \(\Gamma\)”.

- **Types** represent collections (they are different from sets, although they both represent collections) or propositions.
- \(\Gamma \equiv x_1 : A_1, \ldots, x_n : A_n\) is a context, which is a sequence of “membership entries” declaring that \(x_i\) is a variable of type \(A_i\).
  - When \(\Gamma\) is empty, (*) is non-hypothetical; (in this case, we may just write \(a : A\) by omitting “\(\Gamma \vdash\”).)
  - When \(\Gamma\) is non-empty, (*) is hypothetical.
Examples of judgements

- John is a man.
  
  $\Rightarrow$ john : Man, where Man is a type.
  
  (non-hypothetical)

- If John is a student, he is happy.
  
  $\Rightarrow$ j : Student $\vdash$ p : happy(j) (for some p)
  
  (hypothetical)

- Truth of a formula:
  
  - “happy(j) true”
  
  - The above is a shorthand for “p : happy(j) for some p”
Other forms of judgements (1)

❖ $\Gamma$ valid
  ❖ $\Gamma$ is a valid ("legal") context
  ❖ When is $\Gamma \equiv x_1 : A_1, \ldots, x_n : A_n$ valid? (1) $x_i$’s are different; (2) $A_i$’s are types in the prefix on their left.

❖ Question:
  ❖ Why is this necessary?
  ❖ In traditional logics, we do not need this – just consider a set of formulas – this would seem enough ...
  ❖ Answer: because we have dependent types – it is possible that $x_i$’s occur freely in the $A_j$’s after them!
  ❖ Eg, we can have a context
    $$x: \text{Man}, \ldots, y: \text{handsome}(x), \ldots$$
Situations represented as contexts: an example

- **Beatles’ rehearsal**
  - **Domain:** \( \Sigma_1 \equiv D : \text{Type}, \)
    
    \[ \begin{array}{c}
    \text{John} : D, \quad \text{Paul} : D, \quad \text{George} : D, \quad \text{Ringo} : D, \quad \text{Brian} : D, \quad \text{Bob} : D
    \end{array} \]
  - **Assignment:** \( \Sigma_2 \equiv B : D \rightarrow \text{Prop}, \ b_J : B(\text{John}), \ ... , \ b_B : \neg B(\text{Brian}), \ b'_B : \neg B(\text{Bob}), \)
    
    \[ \begin{array}{c}
    G : D \rightarrow \text{Prop}, \ g_J : G(\text{John}), \ ... , \ g_G : \neg G(\text{Ringo}), \ ...
    \end{array} \]
  - **Context representing the situation of Beatles’ rehearsal:**
    \[ \Sigma \equiv \Sigma_1, \ \Sigma_2, \ ... , \ \Sigma_n \]
  - **We have, for example,**
    
    \( \Sigma \vdash G(\text{John}) \text{ true } \) and \( \Sigma \vdash \neg B(\text{Bob}) \text{ true } \)
    
    i.e., under \( \Sigma \), “John played guitar” & “Bob was not a Beatle”. 
Other forms of judgements (2)

- $\Gamma \vdash A$ type
  - $A$ is a type under $\Gamma$.
  - E.g. when is $A \times B$ or $\sum x:A.B$ a valid type?
- $\Gamma \vdash A = B$ and $\Gamma \vdash a = b : A$ (equality judgements)
  - $A$ and $B$ are (computationally) the same types.
  - $a$ and $b$ are (computationally) the same objects of type $A$.
  - E.g., do we have $\pi_1(a,b) = a$?

Now let’s illustrate by types of pairs.
Σ-types: a taste of dependent types

First, we start with “product types” of pairs:

- A x B of pairs (a,b) such that a:A and b:B
- Rules to specify these product types:
  - Formation rule for A x B
  - Introduction rule for pairs (a,b) : A x B
  - Elimination rules for projections π₁(p) and π₂(p)
  - Computation rule: π₁(a,b)=a and π₂(a,b)=b.

This generalises to Σ-types of “dependent pairs” (next page)
“Family” of types
  - $B[x]$ type – type “indexed” by $x : A$
  - $Dog[x]$ type for $x : \text{Human}$
  - $Dog[John] = \{d\}$, $Dog[Mary] = \{d_1, d_2\}$, ...
    (Here, $\{\ldots\}$ are finite types.)

$\Sigma$-types of “dependent pairs”:
  - $\Sigma x:A.B[x]$ of dependent pairs $(a,b)$ such that $a:A$ and $b:B[a]$.

Rules for $\Sigma$-types:
  - Formation rule for $\Sigma x:A.B$
  - Introduction rule for dependent pairs $(a,b) : \Sigma x:A.B[x]$
  - Elimination rules for projections $\pi_1(p) : A$ and $\pi_2(p) : B[\pi_1(p)]$
  - Computation rule: $\pi_1(a,b) = a$ and $\pi_2(a,b) = b$. 
“handsome man” is interpreted as type
\( \Sigma x : \text{Man}. \text{handsome}(x) \)

So,

- A handsome man is an object of the above type.
- It is a pair \((m, p)\) such that \(m : \text{Man}\) and \(p : \text{handsome}(m)\), i.e., \(m\) is a man and \(p\) is a proof that \(m\) is handsome.
Judgements v.s. Formulas/Types

- First, judgements are **not** formulas/propositions.
  - Propositions correspond to types (P in p : P).
  - For example, “P is true” corresponds to “p : P for some p”.
- You may think judgements as meta-level statements that cannot be used “internally”.
  - For example, unlike a formula, you cannot form, for example, \( \neg J \) for a judgement J.
  - This is similar to subtyping judgements \( A \leq B \). Such assumptions may be considered in “signatures” – see my LACL14 invited talk/paper and work in Lungu’s thesis (2017).

We stop here: Further discussions are out of the scope here, but relevant papers are available, if requested.
II.2. Dependent product types (Π-types)

- **Informally** (borrowing set-theoretical notations, formal rules next slide),
  \[ \Pi x : A. B[x] = \{ f \mid \text{for any } a : A, f(a) : B[a] \} \]
- **Examples**
  - \( \lambda x : \text{Nat.}[1,\ldots,x] : \Pi x : \text{Nat. Vect}(x) \)
  - \( \forall x : \text{Student. work\_hard}(x) \)
    - This is just another notation for \( \Pi x : \text{Student. work\_hard}(x) \)
  - \( \forall x : \text{Man. handsome}(x) \supset \neg \text{ugly}(x) \)
- **Notational conventions:**
  - \( A \to B \) stands for \( \Pi x : A. B(x) \) when \( x \notin \text{FV}(B) \).
  - \( P \supset Q \) stands for \( \forall x : A. B(x) \) when \( x \notin \text{FV}(Q) \).
  - In other words, \( A \to B / P \supset Q \) are just special cases of \( \Pi \)-types.
\( \Pi \)-types/\( \forall \)-propositions

\[
\begin{align*}
(\Pi_T) & \quad \Gamma \vdash A \ \text{type} \quad \Gamma, \ x : A \vdash B[x] \ \text{type} \\
& \quad \Gamma \vdash \Pi x : A. B[x] \ \text{type} \\
(\Pi_P) & \quad \Gamma \vdash A \ \text{type} \quad \Gamma, \ x : A \vdash P[x] \ \text{prop} \\
& \quad \Gamma \vdash \Pi x : A. P[x] \ \text{prop} \\
(\lambda) & \quad \Gamma, \ x : A \vdash b : B \\
& \quad \Gamma \vdash \lambda x : A. b : \Pi x : A. B[x] \\
(app) & \quad \Gamma \vdash f : \Pi x : A. B[x] \quad \Gamma \vdash a : A \\
& \quad \Gamma \vdash f(a) : B[a]
\end{align*}
\]

\( \Pi_T \) for \( \Pi \)-types and \( \Pi_P \) for universal quantification
Π-polymorphism – a first informal look

- Use of Π-types for polymorphism – an example:
  - How to model predicate-modifying adverbs (e.g., quickly)?
  - Informally, it can take a verb and return a verb.
  - Montague:
    
    \[
    \text{quickly} : (\text{e} \to \text{t}) \to (\text{e} \to \text{t}) \\
    \text{quickly(run)} : \text{e} \to \text{t}
    \]
  - MTT-semantics, where \( A_q \) is the domain/type for quickly:
    
    \[
    \text{quickly} : (A_q \to \text{Prop}) \to (A_q \to \text{Prop})
    \]
    
    What about other verbs? \( A_{\text{talk}} = \text{Human}, ... \) Can we do it generically with one type of all adverbs?
  - Π-types for polymorphism come for a rescue:
    
    \[
    \text{quickly} : \Pi A : \text{CN}. (A \to \text{Prop}) \to (A \to \text{Prop})
    \]
  - Question: What is CN?
    
    Answer: CN is a universe of types – next slide.
II.3. Universes and \( \Pi \)-polymorphism

- **Universe of types**
  - Martin-Löf introduced the notion of universe (1973, 1984)
  - A universe is a type of types (Note: the collection Type of all types is not a type itself – logical paradox if one allowed \( \Pi \)-quantification over Type.)

- **Examples**
  - Math: needing to define type-valued functions
    - \( f(n) = \underbrace{N \times \ldots \times N}_{n \text{ times}} \)
  - MTT-semantics: for example,
    - CN is the universe of types that are (interpretations of) CNs. We have: Human : CN, Book : CN, \( \Sigma \text{(Man, handsome)} : \text{CN}, \ldots \)
    - We can then have: quickly : \( \Pi A : \text{CN}. (A \rightarrow \text{Prop}) \rightarrow (A \rightarrow \text{Prop}) \)
    - Note: one cannot have \( \Pi A : \text{Type} \ldots \), since Type is not a type.
Modelling subsective adjectives

- Nature of such adjectives
  - Their meanings are dependent on the nouns they modify.
  - Eg, “a large mouse” is not a large animal

- This leads to our following proposal:

  - large : $\Pi A:CN. (A \rightarrow Prop)$
    - CN – type universe of all (interpretations of) CNs
    - $\Pi$ is the type of dependent functions
      - large(Mouse) : Mouse $\rightarrow$ Prop
      - $[\text{large mouse}] = \sum x: \text{Mouse}. \text{large(Mouse)}(x)$

  - skilful : $\Pi A:CN_H. (A \rightarrow Prop)$
    - $CN_H$ – sub-universe of CN of subtypes of Human
      - skilful(Doctor) : Doctor $\rightarrow$ Prop
      - Skilful doctor = $\sum x: \text{Doctor}. \text{skilful(Doctor)}(x)$
    - Excludes expressions like “skilful car”.

ESSLLI 2017
Another example – type of quantifiers

- Generalised quantifiers
  - Examples: some, three, a/an, all, ...
  - In sentences like: “Some students work hard.”

- With \( \Pi \)-polymorphism, the type of binary quantifiers is:
  \( \Pi A:CN. (A \rightarrow Prop) \rightarrow Prop \)

For Q of the above type

\[
N : CN, \ V : N \rightarrow Prop \rightarrow Q(N,V) : Prop
\]

E.g., Student : CN, work_hard : Human \( \rightarrow Prop \)

\( \Rightarrow \) Some(Student, work_hard) : Prop

Note: the above only works because Student \( \leq \) Human – subtyping, a topic to be studied in the next lecture.
Modelling NL coordination

- Examples of conjoinable types
  - John walks and Mary talks. (sentences)
  - John walks and talks. (verbs)
  - A friend and colleague came. (CNs)
  - Every student and every professor came. (quantified NPs)
  - Some but not all students got an A. (quantifiers)
  - John and Mary went to Italy. (proper names)
  - I watered the plant in my bedroom but it still died slowly and agonizingly. (adverbs)
- Question: can we consider coordination generically?
Consider a universe LType

- LType – the universe of “linguistic types”, with formal rules in the next slide.

Example types in Ltype:
- Type CN of common nouns
- Type of predicate-modifying adverbs:
  $\Pi A:CN. (A \rightarrow \text{Prop}) \rightarrow (A \rightarrow \text{Prop})$
- Type of quantifiers:
  $\Pi A:CN. (A \rightarrow \text{Prop}) \rightarrow \text{Prop}$
- ...

\[
\begin{align*}
\text{PType} & : \text{Type} \\
\text{Prop} & : \text{PType} \\
\text{LTType} & : \text{Type} \\
\text{CN} & : \text{LTType} \\
\end{align*}
\]

\[
\begin{array}{c}
A : \text{LTType} \\
P(x) : \text{PType} \\
\Pi x : A . P(x) : \text{PType} \\
A : \text{CN} \\
\end{array}
\]

\[
\begin{array}{c}
\text{A} : \text{PType} \\
\text{A} : \text{LTType} \\
\end{array}
\]

**Fig. 1.** Some (not all) introduction rules for \( \text{LTType} \).
Then, coordination can be considered generically:

- Every (binary) coordinator is of the following type:
  \[ \Pi A : LType. A \rightarrow A \rightarrow A \]

- For example,
  \[ \Pi A : LType. A \rightarrow A \rightarrow A \]

- We can then type the coordination examples we have considered.

- Remark: of course, there are further considerations such as collective readings verses distributive readings – beyond our discussions here.
Plan of Lecture III

- Brief recap of \(\Pi\)-types and polymorphism
  - Illustrate the use of \(\Pi\) and universes by GQs/coordination
- Subtyping in MTTs and applications
  - Subsumptive v.s. coercive subtyping
  - Uses of coercive subtyping in
    - Sense selection
    - Copredication
    - ... ...
  - Adequacy of coercive subtyping for MTTs

Let’s start with two slides seen yesterday.
II.2. Dependent product types ($\Pi$-types)

- Informally (borrowing set-theoretical notations, formal rules next slide),

\[ \Pi x : A. B[x] = \{ f \mid \text{for any } a : A, f(a) : B[a] \} \]

- Examples
  - $\lambda x : \text{Nat.}[1,\ldots,x] : \Pi x : \text{Nat.}\text{Vect}(x)$
  - $\forall x : \text{Student. work\_hard}(x)$
    - This is just another notation for $\Pi x : \text{Student. work\_hard}(x)$
  - $\forall x : \text{Man. handsome}(x) \supset \neg \text{ugly}(x)$

- Notational conventions:
  - $A \rightarrow B$ stands for $\Pi x : A. B(x)$ when $x \notin \text{FV}(B)$.
  - $P \supset Q$ stands for $\forall x : A. B(x)$ when $x \notin \text{FV}(Q)$.
  - In other words, $A \rightarrow B/P \supset Q$ are just special cases of $\Pi$-types.
II.3. Universes and Π-polymorphism

❖ Universe of types
  ❖ Martin-Löf introduced the notion of universe (1973, 1984)
  ❖ A universe is a type of types (Note: the collection Type of all types is not a type itself – logical paradox if one allowed Π-quantification over Type.)

❖ Examples
  ❖ Math: needing to define type-valued functions
    ❖ f(n) = N x ... x N (n times)
  ❖ MTT-semantics: for example,
    ❖ CN is the universe of types that are (interpretations of) CNs. We have:
      Human : CN, Book : CN, Σ(Man, handsome) : CN, ... 
    ❖ We can then have: quickly : ΠA:CN. (A→Prop)→(A→Prop)
    ❖ Note: one cannot have ΠA:Type..., since Type is not a type.
Another example – type of quantifiers

- Generalised quantifiers
  - Examples: some, three, a/an, all, ...
  - In sentences like: “Some students work hard.”

- With \(\Pi\)-polymorphism, the type of binary quantifiers is:

\[
\Pi A: \text{CN}. \ (A \to \text{Prop}) \to \text{Prop}
\]

- For \(Q\) of the above type
  \[
  N : \text{CN}, \ V : N \to \text{Prop} \\
  \Rightarrow \ Q(N,V) : \text{Prop}
  \]

- E.g., for Some of the above type
  \[
  \text{Student} : \text{CN}, \ \text{work\_hard} : \text{Human} \to \text{Prop} \\
  \Rightarrow \ \text{Some(Student,work\_hard)} : \text{Prop}
  \]

Note: This only works because \(\text{Student} \subseteq \text{Human}\) – subtyping, a topic to be studied later.
Modelling NL coordination

- Examples of conjoinable types
  - John walks and Mary talks. (sentences)
  - John walks and talks. (verbs)
  - A friend and colleague came. (CNs)
  - Every student and every professor came. (quantified NPs)
  - Some but not all students got an A. (quantifiers)
  - John and Mary went to Italy. (proper names)
  - I watered the plant in my bedroom but it still died slowly and agonizingly. (adverbs)
- ... ...

- Question: can we consider coordination generically?
Consider a universe LType

- LType – the universe of “linguistic types”, with formal rules in the next slide.

Example types in LType:

- Prop of logical propositions (sentence coordination)
- Type of predicates (verb coordination)
- CN of common nouns (CN coordination)
- Type of predicate-modifying adverbs:
  \[ \Pi A:CN. (A \rightarrow Prop) \rightarrow (A \rightarrow Prop) \] (adverb coordination)
- Type of quantifiers:
  \[ \Pi A:CN. (A \rightarrow Prop) \rightarrow Prop \] (quantifier coordination)

...
\[
\begin{align*}
\text{PType} & : \text{Type} & \text{Prop} & : \text{PType} & A : \text{LType} & P(x) : \text{PType} \ [x:A] \\
\text{LType} & : \text{Type} & \text{CN} & : \text{LType} & A : \text{CN} & A : \text{PType} \\
\end{align*}
\]

**Fig. 1.** Some (not all) introduction rules for \text{LType}.
Then, coordination can be considered generically:

- Every (binary) coordinator is of the following type:
  \[ \Pi A : \text{LType. } A \rightarrow A \rightarrow A \]
- For example,
  \[ \text{and : } \Pi A : \text{LType. } A \rightarrow A \rightarrow A \]

With this typing for coordinators like and, we can then type the coordination examples we have considered.

Remark: Further considerations such as collective verses distributive readings can be dealt with similarly – beyond our discussions here.
III. Subtyping

- Basics on subtyping
  - Subsumptive v.s. coercive subtyping
  - Adequacy for MTTs
- Importance and applications of subtyping in NL sem.
  - Crucial for MTT-semantics
  - Several uses, including
    - Sense selection via overloading
    - Dot-types for copredication

(Here, we shall illustrate applications first and, if time allows, adequacy issue afterwards.)
Subsumptive subtyping: traditional notion

- Subsumptive subtyping:
  \[
  a : A \quad A \leq B
  \]
  \[
  \quad a : B
  \]

  This is called the “subsumption rule”.

- Fundamental principle of subtyping
  \[\text{If } A \leq B \text{ and, wherever a term of type } B \text{ is required, we can use a term of type } A \text{ instead.}\]

  For example, the subsumption rule realises this.
Coercive subtyping: basic idea

- A ≤ B if there is a coercion c from A to B:

  ![Diagram showing coercion from A to B]

  Eg. Even ≤ Nat; Man ≤ Human; ∑(Man, handsome) ≤ Man; ...

- Subtyping as abbreviations:
  a : A ≤c B
  ⇒ “a” can be regarded as an object of type B
  ⇒ C_B[a] = C_B[c(a)], ie, “a” stands for “c(a)”

- This is more general than subsumptive subtyping and adequate for MTTs as well.
Coercive subtyping: summary

- Inadequacy of subsumptive subtyping
  - Canonical objects
  - Canonicity: key for MTTs (TTs with canonical objects)
  - Subsumptive subtyping violates canonicity.

- Adequacy of coercive subtyping for MTTs
  - Coercive subtyping preserves canonicity & other properties.
  - Conservativity (Soloviev & Luo 2002, Luo, Soloviev & Xue 2012)

- Historical development and applications in CS
  - Implementations in proof assistants: Coq, Lego, Plastic, Matita
III.1. Modelling Advanced Linguistic Features

- **MTTs**
  - Very useful in modelling various linguistic features

- **Why? Partly because of**
  - Rich/powerful typing mechanisms
  - Subtying
  - ... ...
Remark on anaphora analysis

- Various treatments of “dynamics”
  - DRTs, dynamic logic, ...
  - MTTs provide a suitable (alternative) mechanism.

- Donkey sentences
  - Eg, “Every farmer who owns a donkey beats it.”
  - Montague semantics
    \[
    \forall x. \text{farmer}(x) \land [\exists y. \text{donkey}(y) \land \text{own}(x,y)] \Rightarrow \text{beat}(x,?y)
    \]
  - Modern TTs (\(\Pi\) for \(\forall\) and \(\Sigma\) for \(\exists\); Sundholme):
    \[
    \Pi x:\text{Farmer}\Pi z: [\Sigma y: \text{Donkey}. \text{own}(x,y)] \text{beat}(x,\pi_1(z))
    \]

- But, this is only an interesting point ... We shall focus on several other things.
Uses of coercive subtyping in MTT-semantics

1. Needs for subtyping in MTT-semantics
2. Sense enumeration/selection via. overloading
3. Linguistic coercions
4. Dot-types and copredication
1. Subtyping: basic need in MTT-semantics

- What about, eg,
  - “A man is a human.”
  - “A handsome man is a man”? 
  - “Paul walks”, with \( p = [\text{Paul}] : [\text{handsome man}] \)?

- **Solution: coercive subtyping**
  - \( \text{Man} \leq \text{Human} \)
  - \([\text{handsome man}] = \sum_{x: \text{Man}} \text{handsome}(x) \leq_{\pi_1} \text{Man} \)
  - \([\text{Paul walks}] = \text{walk}(p) : \text{Prop} \)

  because

  \( \text{walk} : \text{Human} \rightarrow \text{Prop} \)

  and

  \( p : [\text{handsome man}] \leq_{\pi_1} \text{Man} \leq \text{Human} \)
2. Sense selection via overloading

- Sense enumeration (cf, Pustejovsky 1995 and others)
  - Homonymy
  - Automated selection
  - Existing treatments (eg, Asher et al via +\-types)

- For example,
  1. John runs quickly.
  2. John runs a bank.

with homonymous meanings

1. \([\text{run}]_1\) : Human \(\rightarrow\) Prop
2. \([\text{run}]_2\) : Human \(\rightarrow\) Institution \(\rightarrow\) Prop

“run” is overloaded – how to disambiguate?
Overloading via coercive subtyping

- Overloading can be represented by coercions
  Eg. 
  \[ \begin{align*}
  c_1 : [\text{run}]_1 : & \text{Human} \to \text{Prop} \\
  \text{run} : & 1_{\text{run}} \\
  c_2 : [\text{run}]_2 : & \text{Human} \to \text{Institution} \to \text{Prop}
  \end{align*} \]

- Homonymous meanings can be represented so that automated selection can be done according to typings.
3. Linguistic Coercions

- Basic linguistic coercions can be represented by means of coercions in coercive subtyping:
  - (*) Julie enjoyed a book.
  - (**) \( \exists x \): Book. enjoy(j, x)
  - enjoy : Human \( \rightarrow \) Event \( \rightarrow \) Prop
  - Book \( \leq_{\text{reading}} \) Event
  - (*) Julie enjoyed reading a book.

- Local coercions to disambiguate multiple coercions:
  - coercion Book \( \leq_{\text{reading}} \) Event in (**)  
  - coercion Book \( \leq_{\text{writing}} \) Event in (**)
Dependent typing

- What about (example by Asher in [Asher & Luo]):
  
  (♯) Jill just started War and Peace, which Tolstoy finished after many years of hard work. But that won’t last because she never gets through long novels.

- Overlapping scopes of “reading” and “writing”.

- A solution with dependent typing
  
  - Evt : Human → Type
    
    Evt(h) is the type of events conducted by h : Human.
  
  - start, finish, last : ∏h: Human. (Evt(h) → Prop)
  
  - read, write : ∏h: Human. Book → Evt(h)
  
  - Book ⪯_{c(h)} Evt(h), where c(h,b)=writing if “h wrote b” & c(h,b)=reading if otherwise (parameterised coercion over h)
Then, (\#) is formalised as

\[
\begin{align*}
&\text{start}(j,wp) \\
&\quad \land \text{finish}(t,wp) \\
&\quad \land \neg \text{last}(j,wp) \\
&\quad \land \forall lb : LBook. \text{finish}(j, \pi_1(lb))
\end{align*}
\]

which is (equal to)

\[
\begin{align*}
&\text{start}(j,\text{reading}(j,wp)) \\
&\quad \land \text{finish}(t,\text{writing}(t,wp)) \\
&\quad \land \neg \text{last}(j,\text{reading}(j,wp)) \\
&\quad \land \forall lb : LBook. \text{finish}(j, c(j,\pi_1(lb)))
\end{align*}
\]

as intended.
Plan of Lecture IV

- Logic in an MTT
  - Propositions-as-types, consistency, and HOL in UTT
- Brief recap of coercive subtyping
  - Explain the inadequacy of subsumptive subtyping for MTTs
- Two applications of coercive subtyping
  - Copredication via dot-types
    - Dot-types in MTTs for copredication
  - Disjoint union types (A+B)
    - Modelling privative adjective modifications (e.g., fake gun)
IV.1. Logics in MTTs – propositions as types

- Curry-Howard correspondence (1958, 1969):
  - Formulae as types
  - Proofs as objects

<table>
<thead>
<tr>
<th>formula</th>
<th>type</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>P ⊨ Q</td>
<td>P → Q</td>
<td>If ... then ...</td>
</tr>
<tr>
<td>∀x:A.P(x)</td>
<td>Πx:A.P(x)</td>
<td>Every man is handsome.</td>
</tr>
</tbody>
</table>

Eg: λx:P.x : P→P
Curry-Howard correspondence: basic example

- Theorem.
  \[ \vdash^{\mathcal{L}} \text{ for the implicational intuitionistic logic and} \]
  \[ \vdash \text{ for the simply typed } \lambda \text{-calculus.} \]

  Then,
  - if \( \Gamma \vdash M : A \), then \( e(\Gamma) \vdash^{\mathcal{L}} A \), where \( e(\Gamma) \) maps \( x:A \) to \( A \);
  - if \( \Delta \vdash^{\mathcal{L}} A \), then \( \Gamma \vdash M : A \) for some \( \Gamma \) & \( M \) such that \( e(\Gamma) \equiv \Delta \).
Implicational propositional logic

\[(Ax)\] \quad \frac{}{\Gamma, \ A \vdash A}

\[(\rightarrow I)\] \quad \frac{}{\Gamma, \ A \vdash B} \quad \frac{}{\Gamma \vdash A \rightarrow B}

\[(\rightarrow E)\] \quad \frac{\Gamma \vdash A \rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}

where \( \Gamma \) is a set of formulas \( A \).
Simply-typed $\lambda$-calculus (rules as before)

(Var)  
\[ \Gamma, \ x : A \vdash x : A \]

(Abs)  
\[ \Gamma, \ x : A \vdash b : B \]
\[ \Gamma \vdash \lambda x : A. b : A \to B \]

(App)  
\[ \Gamma \vdash f : A \to B \quad \Gamma \vdash a : A \]
\[ \Gamma \vdash f(a) : B \]

where $\Gamma$ is a set of assumptions of the form $x : A$. 
Logic in impredicative type theories

- Prop – universe of logical propositions

\[
\begin{align*}
\Gamma \vdash valid \\
\hline \\
\Gamma \vdash Prop : Type & \quad \Gamma \vdash A : Prop \\
\hline \\
\Gamma \vdash A : Type
\end{align*}
\]

Notational notes:
In these three slides, “A : Type” stands for “A type”. 
Π-types/universal quantification with Prop

(Πₜ)

\(\Gamma \vdash A : Type \quad \Gamma, x : A \vdash B : Type\)

\(\Gamma \vdash \Pi x : A.B : Type\)

(Πₚ)

\(\Gamma \vdash A : Type \quad \Gamma, x : A \vdash P : Prop\)

\(\Gamma \vdash \Pi x : A.P : Prop\)

(λ)

\(\Gamma, x : A \vdash b : B\)

\(\Gamma \vdash \lambda x : A.b : \Pi x : A.B\)

(app)

\(\Gamma \vdash f : \Pi x : A.B \quad \Gamma \vdash a : A\)

\(\Gamma \vdash f(a) : [a/x]B\)

Πₜ for Π-types and Πₚ for universal quantification
Logical operators in, e.g., UTT

\[ \forall x:A.P[x] \quad =_{df} \quad \Pi x:A.P[x] \]
\[ P_1 \supset P_2 \quad =_{df} \quad \forall x:P_1.P_2 \]
\[ \text{true} \quad =_{df} \quad \forall X:\text{Prop}. \quad X \supset X \]
\[ \text{false} \quad =_{df} \quad \forall X:\text{Prop}. \quad X \]
\[ P_1 \& P_2 \quad =_{df} \quad \forall X:\text{Prop}. \quad (P_1 \supset P_2 \supset X) \supset X \]
\[ P_1 \lor P_2 \quad =_{df} \quad \forall X:\text{Prop}. \quad (P_1 \supset X) \supset (P_2 \supset X) \supset X \]
\[ \neg P_1 \quad =_{df} \quad P_1 \supset \text{false} \]
\[ \exists x:A.P[x] \quad =_{df} \quad \forall X:\text{Prop}. \quad (\forall x:A.(P[x] \supset X)) \supset X. \]
Why are these definitions reasonable?
- Usual introduction/elimination rules are all derivable.

Examples
- Conjunction
  - If P and Q are provable, so is P & Q.
  - If P & Q is provable, so are P and Q.
- Falsity
  - false has no proof in the empty context (logical consistency).
  - false implies any proposition.
An episode: logic-enriched type theories

❖ Curry-Howard naturally leads to *intuitionistic* logics.
  ❖ What about, say, *classical* logics?
❖ But:
  ❖ Type-checking and logical inference are orthogonal.
  ❖ They can be independent with each other.
  ❖ In particular, the embedded logic of a type theory is not necessarily intuitionistic.
  ❖ Type theories are not just for constructive mathematics.
❖ A possible answer to the above question:
  ❖ Logic-enriched type theories (LTTs)
IV.2. Subtyping: recap and the adequacy issue

Let’s start with three slides seen yesterday – the basic concepts in subsumptive subtyping and coercive subtyping.
Subsumptive subtyping: traditional notion

Subsumptive subtyping:

\[
\begin{align*}
\text{a : A} & \quad A \leq B \\
\text{------------------------------} \\
\text{a : B}
\end{align*}
\]

This is called the “subsumption rule”.

Fundamental principle of subtyping

If $A \leq B$ and, wherever a term of type $B$ is required, we can use a term of type $A$ instead.

For example, the subsumption rule realises this.
Coercive subtyping: basic idea

- $A \leq B$ if there is a coercion $c$ from $A$ to $B$:

  Eg. Even $\leq$ Nat; Man $\leq$ Human; $\Sigma$(Man, handsome) $\leq$ Man; ...

- Subtyping as abbreviations:
  
  $a : A \leq_c B$
  
  “a” can be regarded as an object of type $B$
  
  $C_B[a] = C_B[c(a)]$, ie, “a” stands for “c(a)”

- This is more general than subsumptive subtyping and adequate for MTTs as well.
Adequacy of subtyping

Question:

Is subsumptive subtyping adequate for MTTs (or type theories with canonical objects)?

Answer:

No (canonicity fails)!

(Hence coercive subtyping.)
Canonicity

Example:
- $A = \text{Nat}, a = 3+4, v = 7.$
Definition

Any closed object of an inductive type is computationally equal to a canonical object of that type.

This is a basis of MTTs – type theories with canonical objects.

- This is why the elimination rule is adequate.
- For $\Sigma$-types, for example, its elimination rules say that any closed object in a $\Sigma$-type is a pair.
Canonicity for subsumptive subtyping?

Q: If $A \leq B$ and $a : A$ is canonical in $A$, is it canonical in $B$?
Canonicity is lost in subsumptive subtyping.

Eg,

\[
\frac{A \leq B}{\text{List}(A) \leq \text{List}(B)}
\]

- nil(A) : List(B), by subsumption;
- But nil(A) \neq \text{any canonical B-list nil(B) or cons(B,b,l)}.
- The elim rule for List(B) is inadequate: it does not cover nil(A) ... ...
Coercive subtyping: summary

- Inadequacy of subsumptive subtyping
  - Canonical objects
  - Canonicity: key for MTTs (TTs with canonical objects)
  - Subsumptive subtyping violates canonicity.

- Adequacy of coercive subtyping for MTTs
  - Coercive subtyping preserves canonicity & other properties.
  - Conservativity (Soloviev & Luo 2002, Luo, Soloviev & Xue 2012)

- Historical development and applications in CS
  - Implementations in proof assistants: Coq, Lego, Plastic, Matita
IV.3. Dot-types and copredication

- **Copredication** (Asher, Pustejovský, ...)
  - John picked up and mastered the book.
  - The lunch was delicious but took forever.
  - The newspaper you are reading is being sued by Mia.
  - ...
  - ...

- **How to deal with this in formal semantics**
  - Dot-objects (eg, Asher 2011, in the Montagovian setting)
  - It has a problem: subtyping and CNs-as-predicates strategy do not fit with each other ...
Subtyping problem in the Montagovian setting

- Problematic example (in Montague semantics)
  - [heavy] : (Phy → t) → (Phy → t)
  - [book] : Phy • Info → t
  - [heavy book] = [heavy][[book]]?
  - In order for the above to be well-typed, we need
    Phy • Info → t ≤ Phy → t
    By contravariance, we need
    Phy ≤ Phy • Info
    But, this is not the case (the opposite is)!
- In MTT-semantics, because CNs are interpreted as types, things work as intended (see next slide).
In MTT-semantics, CNs are types – we have:

“John picked up and mastered the book.”

\[
\text{[pick up]}: \text{Human} \rightarrow \text{PHY} \rightarrow \text{Prop}
\]

\[
\leq \text{Human} \rightarrow \text{PHY} \cdot \text{INFO} \rightarrow \text{Prop}
\]

\[
\leq \text{Human} \rightarrow [\text{book}] \rightarrow \text{Prop}
\]

\[
\text{[master]}: \text{Human} \rightarrow \text{INFO} \rightarrow \text{Prop}
\]

\[
\leq \text{Human} \rightarrow \text{PHY} \cdot \text{INFO} \rightarrow \text{Prop}
\]

\[
\leq \text{Human} \rightarrow [\text{book}] \rightarrow \text{Prop}
\]

Hence, both have the same type (in LType) and therefore can be coordinated by “and” to form “picked up and mastered” in the above sentence.

**Remark:** CNs as types in MTT-semantics – so things work.

**Question:** How to introduce dot-types like PHY • INFO in an MTT?
Dot-types in MTTs

- What is $A \bullet B$?
  - Inadequate accounts (as summarised in (Asher 08)):
    - Intersection type
    - Product type

- Proposal (SALT20, 2010)
  - $A \bullet B$ as type of pairs that do not share components
  - Both projections as coercions

- Implementations
  - Coq implementations (Luo/LACL11,
    - Implemented in proof assistant Plastic by Xue (2012).
Key points of a dot-type

- A dot-type is not an ordinary type (e.g., not an inductive type).
- To form $A \bullet B$, $A$ and $B$ cannot share components:
  - E.g., “Phy$\bullet$Phy” and “(Phy$\bullet$Info)$\bullet$Phy” are not dot-types.
  - This is in line with Pustejovsky’s view that dot-objects “appear in selectional contexts that are contradictory in type specification.” (2005)
- $A \bullet B$ is like $A \times B$ but both projections are coercions:
  - $A \bullet B \leq_{\pi_1} A$ and $A \bullet B \leq_{\pi_2} B$
  - This is OK because of the non-sharing requirement. (Note: to have both projections as coercions would not be OK for product types $A \times B$ since coherence would fail.)
\[
\begin{align*}
A : \text{Type} & \quad B : \text{Type} \quad \mathcal{C}(A) \cap \mathcal{C}(B) = \emptyset \\
\hline
A \bullet B : \text{Type} \\
\end{align*}
\]

\[
\begin{align*}
a : A & \quad b : B \\
\langle a, b \rangle : A \bullet B \\
p_1(c) : A & \quad p_2(c) : B \\
c : A \bullet B \\
\hline
p_1(\langle a, b \rangle) = a : A & \quad p_2(\langle a, b \rangle) = b : B \\
\end{align*}
\]

\[
\begin{align*}
A \bullet B : \text{Type} \\
A \bullet B <_{p_1} A : \text{Type} \\
\end{align*}
\]

\[
\begin{align*}
A \bullet B : \text{Type} \\
A \bullet B <_{p_2} B : \text{Type} \\
\end{align*}
\]
Another example

❖ “heavy book”
   ❖ [heavy] : Phy → Prop
   ≤ Phy•Info → Prop
   ≤ Book → Prop
   ❖ So, the following is well-formed:
     [heavy book] = Σ(Book, [heavy])
IV.4. Disjoint union types

- Disjoint union types
  - \( A+B \) with two injections \( \text{inl} : A \rightarrow A+B \) and \( \text{inr} : B \rightarrow A+B \)
  - Rules for \( A+B \) – formation/introduction/elimination/computation rule(s)
Recall the following slide on adjectives:

- **adjective : CNs → CNs**
  - In MG, predicates to predicates.
  - In MTT-semantics, types to types.
- **Proposals in MTT-sem** *(Chatzikyriakidis & Luo, FG13 & JoLLI17)*

<table>
<thead>
<tr>
<th>classification</th>
<th>example</th>
<th>types employed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intersective</td>
<td>handsome man</td>
<td>$\Sigma$-types (of pairs)</td>
</tr>
<tr>
<td>Subsective</td>
<td>large mouse</td>
<td>$\Pi$-types (polymorphism)</td>
</tr>
<tr>
<td>Privative</td>
<td>fake gun</td>
<td>disjoint union types</td>
</tr>
<tr>
<td>Non-committal</td>
<td>alleged criminal</td>
<td>belief contexts</td>
</tr>
</tbody>
</table>
Privative adjectives

- “fake gun”
  - $G_R$ – type of real guns
  - $G_F$ – type of fake guns
  - $G = G_R + G_F$ – type of all guns
  - Declare inl and inr both as coercions: $G_R \leq_{\text{inl}} G$ and $G_F \leq_{\text{inr}} G$

- Now, eg,
  - Can define “real gun” or “fake gun” inductively as predicates of type $G \rightarrow \text{Prop}$ so that $\neg [\text{real gun}](g)$ iff $[\text{fake gun}](g)$.
  - We can interpret, for $f : G_F$, “f is not a real gun” as $\neg [\text{real gun}](f)$, which is logically equivalent to $[\text{fake gun}](f)$, which is True.
  - Note that, in the above, $[\text{real gun}](f)$ and $[\text{fake gun}](f)$ are only well-typed because $G_R \leq_{\text{inr}} G$ and $G_F \leq_{\text{inr}} G$. 
V. Advanced Topics

- Advanced topics in MTT-semantics
  - Dependent types in event semantics
  - MTT-semantics is both model-theoretic & proof-theoretic
  - Dependent Categorial Grammars
    - Syntactic analysis corresponding to MTT-semantics
    - Two papers: Lambek dependent types (Luo 2015) and Linear dependent types (Luo and Zhang 2016)
  - ... ...

We shall consider the first two in this lecture.

(BTW, references for all lectures are available – see the last several slides of this lecture.)
V.1. Dependent Event Types

- This part is based on the slides for my last week’s presentation of the following paper:

I. Dependent event types

- $C_e$: DETs in simple type theory (Montague’s setting)
- UTT[E]: DETs in modern type theories (MTT-semantics)
- Adequacy of $C_e$: embedding into UTT[E]
- Comparison of traditional event semantics, $C_e$ and UTT[E]

II. Event quantification problem: an example

- EQP in traditional event sem. and solutions in $C_e$ and UTT[E]
Davidson’s event semantics

- Consider:
  - (*) John buttered the toast.
    \[ [(*)] = \text{butter}(j,t), \text{where butter : } e^2 \rightarrow t. \]
  - (**) John buttered the toast with the knife at midnight.
    \[ (?) [(**)] = \text{butter}(j,t,k,m), \text{where butter : } e^4 \rightarrow t \]
    \[ (?) [(**)] = m(k(\text{butter}(j)))(t), \text{where butter : } e \rightarrow e \rightarrow t, m/k : (e \rightarrow t) \rightarrow (e \rightarrow t) \]

- Davidson’s original motivation (1967): better treatment of adverbial modifications – e.g., butter : \( e^2 \rightarrow \text{Event} \rightarrow t \), and
  - \[ [(*)] = \exists e: \text{Event}. \text{butter}(j,t,e) \]
  - \[ [(**)] = \exists e: \text{Event}. \text{butter}(j,t,e) \& \text{with}(e,k) \& \text{at}(e,m) \]
  - Note: \[ [(**)] \supset [(*)] \], among many other desirable inferences.
    (No need for meaning postulates, needed in both (?)-approaches.)

- Neo-Davidson semantics (1980s): eg, butter : \( \text{Event} \rightarrow t \) and
  - \[ [(*)] = \exists e: \text{Event}. \text{butter}(e) \& \text{agent}(e)=j \& \text{patient}(e)=t. \]
I. Dependent event types

- Refined types of events: Event $\Rightarrow$ Evt(...)
- Event types dependent on agents/patients
  - For $a$:Agent and $p$:Patient, consider dependent event types $\text{Event}$, $\text{Evt}_A(a)$, $\text{Evt}_P(p)$, $\text{Evt}_{AP}(a,p)$
  - Note: the subscripts $A$, $P$ and $AP$ are just symbols.
- Subtyping ($a:A$ and $A \leq B \Rightarrow a:B$) between DETs:

```
\text{Evt}_A(a) \leq \text{Evt}_{AP}(a,p) \leq \text{Evt}_P(p) \leq \text{Event}
```

WoLLIC 2017
Dependent event types in Montagovian setting

- Eg. John talked loudly.
  - talk, loud : Event→t
  - agent : Event→e→t

- (neo-)Davidsonian event semantics
  $\exists e : Event. \text{talk}(e) \& \text{loud}(e) \& \text{agent}(e, j)$

- Dependent event types in Montagovian setting:
  $\exists e : Evt_A(j). \text{talk}(e) \& \text{loud}(e)$
  which is well-typed because $Evt_A(j) \leq \text{Event}$. 
$C_e$: Underlying formal system

- $C_e$ extends Church's simple type theory (1940) (as used by Montague in MG), by dependent event types
- Church's STT

\[
\begin{align*}
\text{e type} & \quad \text{t type} & \quad x : A \ [x : A] & \quad P \ true \ [P \ true] \\
\frac{A \ type \ B \ type}{A \ \rightarrow \ B \ type} & \quad \frac{b : B \ [x : A] \ x \ \notin \ FV(B)}{\lambda x : A. b : A \ \rightarrow \ B} & \quad \frac{f : A \ \rightarrow \ B \ \ a : A}{f(a) : B} \\
\frac{P : t \quad Q : t}{P \ \supset \ Q : t} & \quad \frac{Q \ true \ [P \ true]}{P \ \supset \ Q \ true} & \quad \frac{P \ \supset \ Q \ true \ \ P \ true}{Q \ true} \\
\frac{A \ type \ P : t \ [x : A]}{\forall(A, x. P) : t} & \quad \frac{P \ true \ [x : A]}{\forall(A, x. P) \ true} & \quad \frac{\forall(A, x. P[x]) \ true \ \ a : A}{P[a] \ true}
\end{align*}
\]
Dependent event types in $C_e$
UTT[E]: Dependent event types in MTT-sem

- UTT[E]: UTT with coercions in E
  - UTT: a modern type theory (Luo 1994)
  - E characterising subtyping for DETs
- Dependent event types in MTT-semantics

\[
\begin{align*}
\text{John talked loudly.} \\
\text{talk} : \Pi h : \text{Human. } \text{Evt}_A(h) \to \text{Prop.} \\
\text{loud} : \text{Event} \to \text{Prop.} \\
[\text{John talked loudly}] = \exists e : \text{Evt}_A(j). \text{talk}(j, e) \& \text{loud}(e). 
\end{align*}
\]
UTT[E]: formal presentation in LF

- **Constant types/families:**
  - \( \text{Entity}: \text{Type} \)
  - \( \text{Agent}, \text{Patient}: \text{Type} \).
  - \( \text{Event}: \text{Type} \),
    \( \text{Evt}_A: (\text{Agent})\text{Type} \),
    \( \text{Evt}_P: (\text{Patient})\text{Type} \), and
    \( \text{Evt}_{AP}: (\text{Agent})(\text{Patient})\text{Type} \).

- **Coercive subtyping in E for DETs:**
  \[
  \text{Evt}_{AP}(a, p) \leq_{c_1[a,p]} \text{Evt}_A(a), \quad \text{Evt}_{AP}(a, p) \leq_{c_2[a,p]} \text{Evt}_P(p), \\
  \text{Evt}_A(a) \leq_{c_3[a]} \text{Event}, \quad \text{Evt}_P(p) \leq_{c_4[p]} \text{Event},
  \]

  where \( c_3[a] \circ c_1[a, p] = c_4[p] \circ c_2[a, p] \).

- **UTT[E]** has nice properties such as normalisation and consistency (Luo, Soloviev & Xue 2012).
Faithful embedding of $C_e$ into $UTT[E]$

- **Definition (embedding of $C_e$ into $UTT[E]$)**
  - $[x] = x$; $[e] = \text{Entity}$; $[t] = \text{Prop}$
  - $[A \rightarrow B] = [A] \rightarrow [B]$;
    $[\lambda x:A.b] = \lambda ([A], T, [x:A].[b])$, if $[b] : T$
  - $[f(a)] = \text{app}(S, T, [f],[a])$, if $[f] : S \rightarrow T$ and $[a] : S_0 \leq S$
  - $[P \supseteq Q] = [P] \supseteq [Q]$; $[\forall (A,x.P)] = \forall ([A],[x:A].[P])$

- **Theorem (embedding is “faithful”)**
  - $\Gamma \vdash A \text{ type} \Rightarrow [\Gamma] \vdash [A] : \text{Type}$.
  - $\Gamma \vdash a : A \Rightarrow [\Gamma] \vdash [a] : A_0$ for some $A_0$ s.t. $[\Gamma] \vdash A_0 \leq_d [A]$ for some $d$.
  - $\Gamma \vdash P \text{ true} \Rightarrow [\Gamma] \vdash p : [P]$, for some $p$.
  - $\Gamma \vdash A \leq B \Rightarrow [\Gamma] \vdash [A] \leq_c [B] : \text{Type}$, for some unique $c$.

- **Corollary:** $C_e$ inherits nice properties from $UTT[E]$ including, e.g., normalisation and logical consistency.
Comparison (John talked loudly)

- (neo-)Davidsonian event semantics
  - talk, loud : Event→t and agent : Event→e→t.
    \[ \exists e : Event. \text{talk}(e) \& \text{loud}(e) \& \text{agent}(e, j) \]

- Dependent event types in Montagovian setting:
  - talk, loud : Event→t and agent : Event→e→t.
    \[ \exists e : \text{Evt}_A(j). \text{talk}(e) \& \text{loud}(e) \]
    which is well-typed because \( \text{Evt}_A(j) \leq \text{Event} \).

- Dependent event types in MTT-semantics:
  
  \[ \text{talk} : \prod h : \text{Human}. \text{Evt}_A(h) \rightarrow \text{Prop}. \]
  \[ \text{loud} : \text{Event} \rightarrow \text{Prop}. \]
  \[ [\text{John talked loudly}] = \exists e : \text{Evt}_A(j). \text{talk}(j, e) \& \text{loud}(e). \]

  Note: talk’s type requires that e have a dependent event type.
II. Event quantification problem

- A form of incompatibility between event semantics and MG (Champollion, Winter-Zwarts, de Groote-Winter).

- No man talked.

\[(\text{neo-})\text{Davidson (even the incorrect (\#) is legal)\]

\[\begin{align*}
(1) & \quad \neg \exists x : e. \man(x) & \lor \exists e: \text{Event. talk}(e) & \land \text{agent}(e, x) \\
(2) & \quad (\#) \exists e : \text{Event.} \quad \neg \exists x : e. \man(x) & \land \text{talk}(e) & \land \text{agent}(e, x)
\end{align*}\]

**DETs in Montague** (the incorrect (*) is illegal)

\[\begin{align*}
(3) & \quad \neg \exists x : e. \man(x) & \lor \exists e : Evt_A(x). \text{talk}(e) \\
(4) & \quad (*) \exists e : Evt_A(x). \quad \neg \exists x : e. \man(x) & \land \text{talk}(e)
\end{align*}\]

But, we still have a problem, albeit a small one ...
What if one changes $\text{Evt}_A(x)$ into Event?
That still would not prevent the following incorrect semantics:

$$\exists e: \text{Event}. \neg \exists x : e. \text{man}(x) \land \text{talk}(e)$$

MTT-semantics helps:

$$\text{DETs in MTT-sem}$$

(5) $\neg \exists x : \text{Man} \exists e : \text{Evt}_A(x). \text{talk}(x, e)$
(6) (*) $\exists e : \text{Evt}_A(x). \neg \exists x : \text{Man}. \text{talk}(x, e)$

Note: talk’s type “dictates” the use of $\text{Evt}_A(x)$: talk$(x,e)$ would not be well-typed if $e : \text{Event}$ only (and not of type $\text{Evt}_A(x)$). So, something like (#) would not be available.
Future work related to DETs: questions

- Why thematic roles as indexes of DEPs?
  - Conceptual precedency/dependency of existence?
    - Evt\textsubscript{A}(a) for a:Agent
    - “a exists” in order for an event in Evt\textsubscript{A}(a) to exist ...

- Several questions on DETs
  - Dependency on other kinds of parameters than thematic roles?
    (eg, Evt(h) where h:Human in (Asher & Luo 12))
  - Potential applications of DETs (not just event quantification problem.)
  - Other forms of dependent event types
V.2. MTT-sem is both model-/proof-theoretic

- The above claim was first made in the following talk/paper:

- Since then, further discussions and developments have been made, although the basic theme and arguments have remained the same.

Let’s start by revisiting two slides in Lecture 1.
Formal semantics

- **Model-theoretic semantics**
  - Meaning is given by denotation.
  - c.f., Tarski, ..., Montague.
  - e.g., Montague grammar (MG)
    - NL $\rightarrow$ simple type theory $\rightarrow$ set theory

- **Proof-theoretic semantics**
  - In logics, meaning is inferential use (proof/consequence).
  - c.f., Gentzen, Prawitz, ..., Martin-Löf.
  - e.g., Martin-Löf’s meaning theory
Simple example for MTS and PTS

- **Model-theoretic semantics**
  - John is happy. \(\Rightarrow\) happy(john)
  - John is a member of the set of entities that are happy.
  - Montague’s semantics is model-theoretic – it has a wide coverage (powerful).

- **Proof-theoretic semantics**
  - How to understand a proposition like happy(john)?
  - In logic, its meaning can be characterised by its uses – two respects:
    - How it can be arrived at (proved)?
    - How it can be used to lead to other consequences?

(*)
Example argument for traditional set-theoretic sem.

- Or, an argument against non-set-theoretic semantics
- “Meanings are out in the world”
  - Portner’s 2005 book on “What is Meaning” – typical view
  - Assumption that set theory represents (or even is) the world

Comments:

- This is illusion! Set theory is just a theory in FOL, not “the world”.
- A good/reasonable formal system can be as good as set theory. (For example, if set theory is good enough, then so is an MTT.)
Claim:

Formal semantics in Modern Type Theories is both model-theoretic and proof-theoretic.

- **NL → MTT** (representational, model-theoretic)
  - MTT as meaning-carrying language with its types representing collections (or “sets”) and signatures representing situations

- **MTT → Meaning theory** (inferential roles, proof-theoretic)
  - MTT-judgements, which are semantic representations, can be understood proof-theoretically by means of their inferential roles (c.f., Martin-Löf’s meaning theory)
Traditional model-theoretic semantics:
Logics/NL $\rightarrow$ Set-theoretic representations

Traditional proof-theoretic semantics of logics:
Logics $\rightarrow$ Inferences

Formal semantics in Modern Type Theories:
NL $\rightarrow$ MTT-representations $\rightarrow$ Inferences

Remark: This was not possible without a language like MTTs; in other words, MTTs offer a new possibility for NL semantics!
Justifications of the claim

- Model-theoretic characteristics of MTT-semantics
  - Signatures – context-like but more powerful mechanism to represent situations (“incomplete worlds”)
- Proof-theoretic characteristics of MTT-semantics
  - Meaning theory of MTTs – inferential role semantics of MTT-judgements

Remark: The proof-theoretic characteristics is easier to justify; what about the model-theoretic ones? A focus of some recent work such as those on signatures.
In MTT-semantics, MTT is a representational language.

- Types represent collections (c.f., sets in set theory) – see earlier slides on using rich types in MTTs to give semantics.
- Signatures represent situations (or incomplete possible worlds).
Signatures

- Types and signatures/contexts are embodied in judgements:
  \[ \Gamma \vdash_{\Sigma} a : A \]
  where \( A \) is a type, \( \Gamma \) is a context and \( \Sigma \) is a signature.

- New: Signatures, similar to contexts, are finite sequences of entries, but
  - their entries are introducing constants (not variables; i.e., cannot be abstracted – c.f, Edinburgh LF (Harper, Honsell & Plotkin 1993)), and
  - besides membership entries, allows more advanced ones such as manifest entries and subtyping entries (see later).
Situations represented as signatures

- **Beatles’ rehearsal: simple example**
  - **Domain:** \(\Sigma_1 \equiv D : Type,\)
    \[
    \begin{align*}
    & John : D, \quad Paul : D, \quad George : D, \quad Ringo : D, \\
    & Brian : D, \quad Bob : D
    \end{align*}
    \]
  - **Assignment:** \(\Sigma_2 \equiv B : D \rightarrow Prop, \quad b_J : B(John), \ldots, \quad b_B : \neg B(Brian), \quad b'_B : \neg B(Bob), \)
    \[
    \begin{align*}
    & G : D \rightarrow Prop, \quad g_J : G(John), \ldots, \quad g_G : \neg G(Ringo), \ldots
    \end{align*}
    \]
  - **Signature representing the situation of Beatles’ rehearsal:**
    \[\Sigma \equiv \Sigma_1, \quad \Sigma_2, \ldots, \quad \Sigma_n\]
  - **We have, for example,**
    \[\Gamma \vdash_{\Sigma} G(John) \text{ true and } \Gamma \vdash_{\Sigma} \neg B(Bob) \text{ true.}\]
    “John played guitar” and “Bob was not a Beatle”.

*Remark: the same as a slide in Lecture 2, except that we now use signatures, rather than contexts.*
This shows that, by means of membership entries, we already can do things we would usually do in models (in set theory):

- Declaring types (say, D is a type, representing a collection)
- Declaring objects of a type (say John : D)
- Remark: In a many-sorted FOL, one may declare a FOL-language with sorts and constants, not different sorts/constants in the same language.

However, we need to further increase the representational power – manifest fields and subtyping assumptions in signatures.
Manifest entries

- More sophisticated situations
  - E.g., infinite domains
- In signatures, we can have a manifest entry:
  
  \[ x \sim a : A \]

where \( a : A \).

- Informally, it assumes \( x \) that behaves the same as \( a \).
Manifest entries: formal treatment

- Manifest entries are just abbreviations of special membership entries:
  - \(x \sim a : A\) abbreviates \(x : 1_A(a)\) where \(1_A(a)\) is the unit type with only object \(*_A(a)\).
  - with the following coercion:
    \[
    \frac{\Gamma \vdash \Sigma A : Type \quad \Gamma \vdash \Sigma a : A}{\Gamma \vdash \Sigma 1_A(a) \leq_{\xi_{A,a}} A : Type}
    \]
    where \(\xi_{A,a}(z) = a\) for every \(z : 1_A(a)\).
  - So, in any hole that requires an object of type \(A\), we can use \(x\) which, under the above coercion, will be coerced into \(a\), as intended.
Manifest entries: examples

\[\Sigma_1 \equiv D : Type,\]
\[\text{John} : D, \text{Paul} : D, \text{George} : D, \text{Ringo} : D, \text{Brian} : D, \text{Bob} : D\]
\[\Sigma_2 \equiv B : D \rightarrow Prop, b_J : B(\text{John}), ..., b_B : \neg B(\text{Brian}), b'_B : \neg B(\text{Bob}),\]
\[G : D \rightarrow Prop, g_J : G(\text{John}), ..., g_G : \neg G(\text{Ringo}), ...\]

\[D \sim a_D : Type, B \sim a_B : D \rightarrow Prop, G \sim a_G : D \rightarrow Prop,\]

where

\[a_D = \{\text{John, Paul, George, Ringo, Brian, Bob}\}\]
\[a_B : D \rightarrow Prop, \text{the predicate ‘was a Beatle’},\]
\[a_G : D \rightarrow Prop, \text{the predicate ‘played guitar’},\]

with \(a_D\) being a finite type and \(a_B\) and \(a_G\) inductively defined.
(Note: Formally, “Type” should be a type universe.)
Infinity:

- Infinite domain $D$ represented by infinite type $\text{Inf}$
  \[ D \sim \text{Inf} : \text{Type} \]
- Infinite predicate with domain $D$:
  \[ f \sim f\text{-defn} : D \rightarrow \text{Prop} \]
  with $f\text{-defn}$ being inductively defined.
- "Animals in a snake exhibition":
  \[ \text{Animal}_1 \sim \text{Snake} : \text{CN} \]
Subtyping entries in signatures

- Subtyping entries in a signature:
  \[ c : A \leq B \]
  This is to declare \( A \leq c B \), where \( c \) is a functional operation from \( A \) to \( B \).
- Eg, we may have
  \[ D \sim \{ \text{John, ... } \} : \text{Type}, c : D \leq \text{Human} \]
- Note that, formally, for signatures,
  - we only need “coercion contexts” but do not need “local coercions” [Luo 2009, Luo & Part 2013];
  - this is meta-theoretically simpler (Lungu 2017)
Concluding Remarks

- Using contexts to represent situations: historical notes
  - Ranta 1994 (even earlier?)
  - Further references [Bodini 2000, Cooper 2009, Dapoigny/Barlatier 2010]

- We introduce “signatures” with new forms of entries: manifest/subtyping entries
  - Manifest/subtyping entries in signatures are simpler than manifest fields (Luo 2009) and local coercions (Luo & Part 2013).

- Preserving TT’s meta-theoretic properties is important (eg, consistency of the embedded logic).

- Summary
  - NL $\rightarrow$ MTT (model-theoretic)
  - MTT $\rightarrow$ meaning theory (proof-theoretic)
References (1)

- J. Belo. Dependentely Sorted Logic. LNCS 4941.
- P. Bodini. Formalizing Contexts in Intuitionistic Type Theory. Fundamenta Informaticae 4(2).
References (2)

References (3)

- S. Chatzikyriakidis and Z. Luo. Formal Semantics in Modern Type Theories. ISTE/Wiley Science Publishing Ltd. (to appear)
- Dapoigny and Barlatier. Modelling Contexts with Dependent Types. Fundamenta Informaticae 104. 2010.
References (4)

References (5)

References (6)
