Adapting type theory with records for natural language semantics

Robin Cooper
University of Gothenburg

This research was supported in part by VR project 2009-1569, Semantic analysis of interaction and coordination in dialogue (SAICD).
Outline

Rich type theory, cognition and the formal semantics tradition

Type theory without records

Type theory with dependent record types

Dependent types
Outline

Rich type theory, cognition and the formal semantics tradition

Type theory without records

Type theory with dependent record types

Dependent types
Rich type theory

▷ *traditional type theories* (e.g. Montague, 1973, 1974) provide types for basic ontological classes (e.g., for Montague, entities, truth values, time points, possible worlds and total functions between these objects)
Rich type theory

- **traditional type theories** (e.g. Montague, 1973, 1974) provide types for basic ontological classes (e.g., for Montague, entities, truth values, time points, possible worlds and total functions between these objects)

- **rich type theories** (e.g. Martin-Löf, 1984) (“modern type theory”, Luo, 2010, 2011) provide a more general collection of types, e.g. in our type theory, categories of objects such as Tree, types of situations such as *Hugging of a dog by a boy*
Judgement

- (An agent judges that) object $a$ is of type $T$.
- $a : T$
A cognitive spin on type theory
A cognitive spin on type theory
A cognitive spin on type theory

- Adapting type theory with records for natural language semantics
- Rich type theory, cognition and the formal semantics tradition

Tree

invariance
(type)

Tree'

neural
implementation
of type

"A tree!"
A cognitive spin on type theory
A cognitive spin on type theory

Gibson (1986); Barwise and Perry (1983)
Perception and semantics

- Relate this simple minded view of perception to the kind of natural language interpretation which main stream semantics has taught us about
Perception and semantics

- Relate this simple minded view of perception to the kind of natural language interpretation which mainstream semantics has taught us about
- A view of linguistic evolution which roots linguistic ability in basic cognitive ability
Perception and semantics

- Relate this simple minded view of perception to the kind of natural language interpretation which main stream semantics has taught us about
- A view of linguistic evolution which roots linguistic ability in basic cognitive ability
- Do this in a way that incorporates results we have obtained from mainstream formal semantics
Perception and semantics

- Relate this simple minded view of perception to the kind of natural language interpretation which main stream semantics has taught us about
- A view of linguistic evolution which roots linguistic ability in basic cognitive ability
- Do this in a way that incorporates results we have obtained from mainstream formal semantics
- But also in a way that can provide useful applications in robotic systems, including learning theories
Some tensions between type theory and classical formal semantics

- By “classical formal semantics” we understand work in the Montague tradition
Some tensions between type theory and classical formal semantics

> By “classical formal semantics” we understand work in the Montague tradition

> Classical formal semantics is *model theoretic*. Rich type theory is *proof theoretic*.
Some tensions between type theory and classical formal semantics

- By “classical formal semantics” we understand work in the Montague tradition
- Classical formal semantics is *model theoretic*. Rich type theory is *proof theoretic*.
- Perhaps Martin-Löf’s *proof objects* narrow the gap?
Some tensions between type theory and classical formal semantics

- By “classical formal semantics” we understand work in the Montague tradition.
- Classical formal semantics is *model theoretic*.
- Rich type theory is *proof theoretic*.
- Perhaps Martin-Löf’s *proof objects* narrow the gap?
- Formal semantics assumes variants of *classical* logic. Type theory is *intuitionistic* or at least *constructivist*.
Some tensions between type theory and classical formal semantics

- By “classical formal semantics” we understand work in the Montague tradition.
- Classical formal semantics is *model theoretic*. Rich type theory is *proof theoretic*.
- Perhaps Martin-Löf’s *proof objects* narrow the gap?
- Formal semantics assumes variants of *classical* logic. Type theory is *intuitionistic* or at least *constructivist*.
- Perhaps surprising that linguists, interested in cognition, should be resistant to intuitionism. Humans do have a tendency to talk about things they can’t prove...
Some tensions, *contd*.

- Formal semantics is founded on *set theory* (ZF with urelements)
  Type theory normally regarded as an alternative to set theory
How to live with tension

- algebraic approach to structured semantic objects – proof theory in model theory
How to live with tension

- algebraic approach to structured semantic objects – proof theory in model theory
- transport constructive techniques to a classical setting, experiment with hybrids
How to live with tension

- algebraic approach to structured semantic objects – proof theory in model theory
- transport constructive techniques to a classical setting, experiment with hybrids
- build type theory on set theory
How to live with tension

- algebraic approach to structured semantic objects – proof theory in model theory
- transport constructive techniques to a classical setting, experiment with hybrids
- build type theory on set theory
- in general, a good way to make yourself unpopular

Perhaps Homotopy Type Theory offers a path to a resolution (Univalent Foundations Program, 2013)
How to live with tension

- algebraic approach to structured semantic objects – proof theory in model theory
- transport constructive techniques to a classical setting, experiment with hybrids
- build type theory on set theory
- in general, a good way to make yourself unpopular
- perhaps Homotopy Type Theory offers a path to a resolution (Univalent Foundations Program, 2013)
Outline

Rich type theory, cognition and the formal semantics tradition

Type theory without records

Type theory with dependent record types

Dependent types
TTR (type theory with records) 2012

- The most recent published reference for the details is Cooper (2012)
- Also Cooper (2005a) for an earlier detailed treatment
- Cooper (2005b) for relation to various semantic theories
- https://sites.google.com/site/typetheorywithrecords/drafts/ for some current work in progress
Basic types

A *system of basic types* is a pair:

\[ \text{TYPE}_B = \langle \text{Type}, A \rangle \]

where:

1. **Type** is a non-empty set
2. \( A \) is a function whose domain is **Type**
3. For any \( T \in \text{Type} \), \( A(T) \) is a set disjoint from **Type**
4. For any \( T \in \text{Type} \), \( a : \text{TYPE}_B \ T \) iff \( a \in A(T) \)
Adapting type theory with records for natural language semantics

Type theory without records

\[ a : T_1 \]
Intensionality

- Important: types are mathematical objects in their own right, they are not just sets of objects.
Intensionality

- Important: types are mathematical objects in their own right, they are not just sets of objects.
- $\forall a[a : T_1 \leftrightarrow a : T_2] \nRightarrow T_1 = T_2$
Separating modality and intensionality

- the traditional linguistic semantic approach uses possible worlds for both modality and intensionality
- several proposals for decoupling the two: cf. Thomason (1980); Muskens (2005), property theory (Chierchia and Turner, 1988; Fox and Lappin, 2005; Lappin, 2012)
- type theory offers a similar possibility to these proposals
Modal type systems

A modal system of basic types\(^1\) is a family of pairs:

\[
\text{TYPE}_{MB} = \langle \text{Type}, A \rangle_{A \in A}
\]

where:

1. \(A\) is a set of functions with domain \(\text{Type}\)
2. for each \(A \in A\), \(\langle \text{Type}, A \rangle\) is a system of basic types

\(^1\)This definition was not present in Cooper (2012).
A modal basic type system
Some simple modal notions I

If $\text{TYPE}_{MB} = \langle \text{Type}, A \rangle_{A \in \mathcal{A}}$ is a modal system of basic types, we shall use the notation $\text{TYPE}_{MB_A}$ (where $A \in \mathcal{A}$) to refer to that system of basic types in $\text{TYPE}_{MB}$ whose type assignment is $A$. Then:

1. for any $T_1, T_2 \in \text{Type}$, $T_1$ is (necessarily) equivalent to $T_2$ in $\text{TYPE}_{MB}$, $T_1 \cong_{\text{TYPE}_{MB}} T_2$, iff for all $A \in \mathcal{A}$,
   $$\{ a \mid a : \text{TYPE}_{MB_A} T_1 \} = \{ a \mid a : \text{TYPE}_{MB_A} T_2 \}$$

2. for any $T_1, T_2 \in \text{Type}$, $T_1$ is a subtype of $T_2$ in $\text{TYPE}_{MB}$, $T_1 \subseteq_{\text{TYPE}_{MB}} T_2$, iff for all $A \in \mathcal{A}$,
   $$\{ a \mid a : \text{TYPE}_{MB_A} T_1 \} \subseteq \{ a \mid a : \text{TYPE}_{MB_A} T_2 \}$$

3. for any $T \in \text{Type}$, $T$ is necessary in $\text{TYPE}_{MB}$ iff for all $A \in \mathcal{A}$,
   $$\{ a \mid a : \text{TYPE}_{MB_A} T \} \neq \emptyset$$
Some simple modal notions II

4. for any $T \in \text{Type}$, $T$ is possible in $\text{TYPE}_{MB}$ iff for some $A \in \mathcal{A}$, 
$$\{a \mid a : \text{TYPE}_{MB_A} T\} \neq \emptyset$$
Types may be structured mathematical objects

- types may be constructed from other mathematical objects
- that is, they are complex types (non-basic types)
Types may be structured mathematical objects

- types may be constructed from other mathematical objects
- that is, they are complex types (non-basic types)
- one kind of complex type is ptype, types which are constructed from predicates and objects used as arguments to the predicate
Types may be structured mathematical objects

- types may be constructed from other mathematical objects
- that is, they are complex types (*non-basic* types)
- one kind of complex type is *ptype*, types which are constructed from predicates and objects used as arguments to the predicate
- another kind of complex type is *record type*, types which consist of a collection of types indexed by labels
Why is structure important?

- Increases intensionality (e.g. same “content”, different structure)
- Allows us to find parts within a whole (e.g. in clarification)
- Allows us to modify by adding or removing a part (e.g. in learning new meanings or coordinating meaning with your dialogue partner)
Why is structure important?

- increases intensionality (e.g. same “content”, different structure)
Why is structure important?

- increases intensionality (e.g. same “content”, different structure)
- allows us to find parts within a whole (e.g. in clarification)
Why is structure important?

- increases intensionality (e.g. same “content”, different structure)
- allows us to find parts within a whole (e.g. in clarification)
- allows us to modify by adding or removing a part (e.g. in learning new meanings or coordinating meaning with your dialogue partner)
Seeing a hugging event

"The boy is hugging the dog."
Predicates

- predicates such as ‘run’, ‘hug’
Predicates

- predicates such as ‘run’, ‘hug’
- predicates come along with an *arity* which tells you what kind of *arguments* the predicates have:
  - Arity(run) = \langle Ind \rangle
  - Arity(hug) = \langle Ind, Ind \rangle
Predicates

- predicates such as ‘run’, ‘hug’
- predicates come along with an *arity* which tells you what kind of *arguments* the predicates have:
  - \( \text{Arity}(\text{run}) = \langle \text{Ind} \rangle \)
  - \( \text{Arity}(\text{hug}) = \langle \text{Ind}, \text{Ind} \rangle \)
- We might also want to include time intervals and locations as part of the arities of these predicates
Predicates

A *predicate signature* is a triple

\[ \langle \text{Pred}, \text{ArgIndices}, \text{Arity} \rangle \]

where:

1. **Pred** is a set (of predicates)
2. **ArgIndices** is a set (of indices for predicate arguments, normally types)
3. **Arity** is a function with domain **Pred** and range included in the set of finite sequences of members of **ArgIndices**.
Polymorphic predicates

A _polymorphic predicate signature_ is a triple

\[ \langle \text{Pred, ArgIndices, Arity} \rangle \]

where:

1. **Pred** is a set (of predicates)
2. **ArgIndices** is a set (of indices for predicate arguments, normally types)
3. **Arity** is a function with domain **Pred** and range included in the powerset of the set of finite sequences of members of **ArgIndices**.
Ptypes

- ptypes are constructed from predicates and arguments corresponding to their arities
Ptypes

- ptypes are constructed from predicates and arguments corresponding to their arities
- examples: run(d), hug(b,d) (where b,d:Ind)
Ptypes

- Ptypes are constructed from predicates and arguments corresponding to their arities.
- Examples: run(d), hug(b,d) (where b,d:Ind)
- Ptypes are types of situations (events)
Ptypes

- ptypes are constructed from predicates and arguments corresponding to their arities
- examples: run(d), hug(b,d) (where b,d:Ind)
- ptypes are types of situations (events)
- a system of complex types will contain a set of ptypes, PType, in addition to basic types
Ptypes

- ptypes are constructed from predicates and arguments corresponding to their arities
- examples: run(d), hug(b,d) (where b,d:Ind)
- ptypes are types of situations (events)
- a system of complex types will contain a set of ptypes, PType, in addition to basic types
- PType will be based on a set of predicates with their arities
Ptypes

- ptypes are constructed from predicates and arguments corresponding to their arities
- examples: run(d), hug(b,d) (where b,d:Ind)
- ptypes are types of situations (events)
- a system of complex types will contain a set of ptypes, \( \text{PType} \), in addition to basic types
- \( \text{PType} \) will be based on a set of predicates with their arities
- \( \text{PType} \) will contain all the possible ptypes for a given predicate given what is assigned to the arity for the predicate elsewhere in the system
Ptypes

- ptypes are constructed from predicates and arguments corresponding to their arities
- examples: run\((d)\), hug\((b,d)\) (where \(b,d:\text{Ind}\) )
- ptypes are types of situations (events)
- a system of complex types will contain a set of ptypes, \(\text{PType}\), in addition to basic types
- \(\text{PType}\) will be based on a set of predicates with their arities
- \(\text{PType}\) will contain all the possible ptypes for a given predicate given what is assigned to the arity for the predicate elsewhere in the system
- a system of complex types will also contain a function, \(F\), which assigns a set, possibly empty, (of situations) to each ptype.
System of complex types I

A system of complex types is a quadruple:

\[ \text{TYPE}_C = \langle \text{Type}, \text{BType}, \langle \text{PType}, \text{Pred}, \text{ArgIndices}, \text{Arity} \rangle, \langle A, F \rangle \rangle \]

where:

1. \( \langle \text{BType}, A \rangle \) is a system of basic types
2. \( \text{BType} \subseteq \text{Type} \)
3. for any \( T \in \text{Type} \), if \( a : \langle \text{BType}, A \rangle \ T \) then \( a : \text{TYPE}_C \ T \)
4. \( \langle \text{Pred}, \text{ArgIndices}, \text{Arity} \rangle \) is a (polymorphic) predicate signature
System of complex types II

5. $^2 P(a_1, \ldots, a_n) \in \text{PType}$ iff $P \in \text{Pred}$,
   $T_1 \in \text{Type}, \ldots, T_n \in \text{Type}$, $\text{Arity}(P) = \langle T_1, \ldots, T_n \rangle$
   ($\langle T_1, \ldots, T_n \rangle \in \text{Arity}(P)$) and $a_1 : \text{TYPE}_C T_1, \ldots, a_n : \text{TYPE}_C T_n$

6. $\text{PType} \subseteq \text{Type}$

7. for any $T \in \text{PType}$, $F(T)$ is a set disjoint from $\text{Type}$

8. for any $T \in \text{PType}$, $a : \text{TYPE}_C T$ iff $a \in F(T)$

---

$^2$This clause has been modified since Cooper (2012) where it was a conditional rather than a biconditional.
Models

- $A$ and $F$ together, that is, $\langle A, F \rangle$, is a *model*
- a model consists of an assignment to the basic types and an assignment to the ptypes
Models

- $A$ and $F$ together, that is, $\langle A, F \rangle$, is a *model*
- a model consists of an assignment to the basic types and an assignment to the ptypes
- note that a model in this sense is *part* of the type system (not an external interpretation of it)
- this is an important difference between rich type theories and traditional model theory
- the model will affect what ptypes there are
- modal type systems will no longer hold the set of types constant across different possibilities
Modal systems of complex types

A modal system of complex types based on $\mathcal{M}$ is a family of quadruples:

$$\text{TYPE}_{MC} = \langle \text{Type}_M, \text{BType}, \langle \text{PType}_M, \text{Pred}, \text{ArgIndices}, \text{Arity} \rangle, M \rangle_{M \in \mathcal{M}}$$

where for each $M \in \mathcal{M}$, $\langle \text{Type}_M, \text{BType}, \langle \text{PType}_M, \text{Pred}, \text{ArgIndices}, \text{Arity} \rangle, M \rangle$ is a system of complex types.

---

3 This definition has been modified since Cooper (2012) to make $\text{PType}$ and $\text{Type}$ be relativized to the model $M$. 

---
Restrictive and inclusive modal notions

If $\text{TYPE}_{MC} = \langle \text{Type}_M, \text{BType}, \langle \text{PType}_M, \text{Pred}, \text{ArgIndices}, \text{Arity} \rangle, M \rangle_{M \in \mathcal{M}}$ is a modal system of complex types based on $\mathcal{M}$, we shall use the notation $\text{TYPE}_{MC_M}$ (where $M \in \mathcal{M}$) to refer to that system of complex types in $\text{TYPE}_{MC}$ whose model is $M$.

Let $\text{Type}_{MC_{restr}}$ be $\bigcap_{M \in \mathcal{M}} \text{Type}_M$, the “restrictive” set of types which occur in all possibilities, and $\text{Type}_{MC_{incl}}$ be $\bigcup_{M \in \mathcal{M}} \text{Type}_M$, the “inclusive” set of types which occur in at least one possibility.
Restrictive modal notions

1. for any $T_1, T_2 \in \text{Type}_{MC_{\text{restr}}}$, $T_1$ is (necessarily) equivalent, to $T_2$ in $\text{TYPE}_{MC}$, $T_1 \approx_{\text{TYPE}_{MC}} T_2$, iff for all $M \in \mathcal{M}$,
   \[
   \{ a \mid a : \text{TYPE}_{MC_M} T_1 \} = \{ a \mid a : \text{TYPE}_{MC_M} T_2 \}
   \]

2. for any $T_1, T_2 \in \text{Type}_{MC_{\text{restr}}}$, $T_1$ is a subtype, of $T_2$ in $\text{TYPE}_{MC}$, $T_1 \sqsubseteq_{\text{TYPE}_{MC}} T_2$, iff for all $M \in \mathcal{M}$,
   \[
   \{ a \mid a : \text{TYPE}_{MC_M} T_1 \} \subseteq \{ a \mid a : \text{TYPE}_{MC_M} T_2 \}
   \]

3. for any $T \in \text{Type}_{MC_{\text{restr}}}$, $T$ is necessary, in $\text{TYPE}_{MC}$ iff for all $M \in \mathcal{M}$,
   \[
   \{ a \mid a : \text{TYPE}_{MC_M} T \} \neq \emptyset
   \]

4. for any $T \in \text{Type}_{MC_{\text{restr}}}$, $T$ is possible, in $\text{TYPE}_{MC}$ iff for some $M \in \mathcal{M}$,
   \[
   \{ a \mid a : \text{TYPE}_{MC_M} T \} \neq \emptyset
   \]
Inclusive modal notions

1. for any $T_1, T_2 \in \text{Type}_{MC_{incl}}$, $T_1$ is (necessarily) equivalent to $T_2$ in $\text{TYPE}_{MC}$, $T_1 \approx \text{TYPE}_{MC} T_2$, iff for all $M \in \mathcal{M}$, if $T_1$ and $T_2$ are members of $\text{Type}_M$, then
   \[
   \{ a \mid a : \text{TYPE}_{MC_M} T_1 \} = \{ a \mid a : \text{TYPE}_{MC_M} T_2 \}
   \]

2. for any $T_1, T_2 \in \text{Type}_{MC_{incl}}$, $T_1$ is a subtype of $T_2$ in $\text{TYPE}_{MC}$, $T_1 \sqsubseteq \text{TYPE}_{MC} T_2$, iff for all $M \in \mathcal{M}$, if $T_1$ and $T_2$ are members of $\text{Type}_M$, then
   \[
   \{ a \mid a : \text{TYPE}_{MC_M} T_1 \} \subseteq \{ a \mid a : \text{TYPE}_{MC_M} T_2 \}
   \]

3. for any $T \in \text{Type}_{MC_{incl}}$, $T$ is necessary in $\text{TYPE}_{MC}$ iff for all $M \in \mathcal{M}$, if $T \in \text{Type}_M$, then
   \[
   \{ a \mid a : \text{TYPE}_{MC_M} T \} \neq \emptyset
   \]

4. for any $T \in \text{Type}_{MC_{incl}}$, $T$ is possible in $\text{TYPE}_{MC}$ iff for some $M \in \mathcal{M}$, if $T \in \text{Type}_M$, then
   \[
   \{ a \mid a : \text{TYPE}_{MC_M} T \} \neq \emptyset
   \]
Relating restrictive and inclusive modal notions

It is easy to see that if any of the restrictive definitions holds for given types in a particular system then the corresponding inclusive definition will also hold for those types in that system.
Adding function types

A system of complex types \( \text{TYPE}_C = \langle \text{Type}, \text{BType}, \langle \text{PType}, \text{Pred}, \text{ArgIndices}, \text{Arity} \rangle, \langle A, F \rangle \rangle \) has function types if

1. for any \( T_1, T_2 \in \text{Type} \), \( (T_1 \to T_2) \in \text{Type} \)

2. for any \( T_1, T_2 \in \text{Type} \), \( f : \text{TYPE}_C (T_1 \to T_2) \) iff \( f \) is a function whose domain is \( \{ a \mid a : \text{TYPE}_C T_1 \} \) and whose range is included in \( \{ a \mid a : \text{TYPE}_C T_2 \} \)
Adding join (disjunctive) types

A system of complex types $\text{TYPE}_C = \langle \text{Type}, \text{BType}, \langle \text{PType}, \text{Pred}, \text{ArgIndices}, \text{Arity} \rangle, \langle A, F \rangle \rangle$ has join types if

1. for any $T_1, T_2 \in \text{Type}$, $(T_1 \lor T_2) \in \text{Type}$
2. for any $T_1, T_2 \in \text{Type}$, $a : \text{TYPE}_C (T_1 \lor T_2)$ iff $a : \text{TYPE}_C T_1$ or $a : \text{TYPE}_C T_2$
Adding meet (intersection, conjunctive) types

A system of complex types $\text{TYPE}_C = \langle \text{Type}, \text{BType}, \langle \text{PType}, \text{Pred}, \text{ArgIndices}, \text{Arity} \rangle, \langle A, F \rangle \rangle$ has meet types if

1. for any $T_1, T_2 \in \text{Type}$, $(T_1 \land T_2) \in \text{Type}$
2. for any $T_1, T_2 \in \text{Type}$, $a : \text{TYPE}_C (T_1 \land T_2)$ iff $a : \text{TYPE}_C T_1$ and $a : \text{TYPE}_C T_2$
Making types first class citizens

- allowing types to be arguments to predicates
- important for treating attitude predicates such as *believe*, *know*
- simplest treatment is to allow ptypes like ‘believe(*a*, *T*)’ where *a* is an individual and *T* is a type
- more sophisticated is to allow Austinian propositions (pairs of objects and types) as arguments. (Austin, 1961; Barwise and Perry, 1983; Ginzburg, 2012)
The type Type and stratification

An intensional system of complex types is a family of quadruples indexed by the natural numbers:

\[\text{TYPE}_{IC} = \left< \text{Type}^n, \text{BType}, \left< \text{PType}^n, \text{Pred}, \text{ArgIndices}, \text{Arity} \right>, \left< A, F^n \right> \right>_{n \in \text{Nat}}\]

where (using \(\text{TYPE}_{IC,n}\) to refer to the quadruple indexed by \(n\)):

1. for each \(n\), \(\left< \text{Type}^n, \text{BType}, \left< \text{PType}^n, \text{Pred}, \text{ArgIndices}, \text{Arity} \right>, \left< A, F^n \right> \right>\) is a system of complex types
2. for each \(n\), \(\text{Type}^n \subseteq \text{Type}^{n+1}\) and \(\text{PType}^n \subseteq \text{PType}^{n+1}\)
3. for each \(n\), if \(T \in \text{PType}^n\) and \(p \in F^n(T)\) then \(p \in F^{n+1}(T)\)
4. for each \(n > 0\), \(\text{Type}^n \in \text{Type}^n\)
5. for each \(n > 0\), \(T : \text{TYPE}_{IC,n}\) Type\(^n\) iff \(T \in \text{Type}^{n-1}\)
Dependent function types

An intensional system of complex types $\textbf{TYPE}_{IC}$,

$$\textbf{TYPE}_{IC} = \langle \text{Type}^n, \text{BType}, \langle \text{PType}^n, \text{Pred}, \text{ArgIndices}, \text{Arity} \rangle, \langle A, F^n \rangle \rangle_{n \in \text{Nat}}$$

has dependent function types if

1. for any $n > 0$, $T \in \text{Type}^n$ and $F : \textbf{TYPE}_{ICn} (T \rightarrow \text{Type}^n)$, $((a : T) \rightarrow F(a)) \in \text{Type}^n$

2. for each $n > 0$, $f : \textbf{TYPE}_{ICn} ((a : T) \rightarrow F(a))$ iff $f$ is a function whose domain is $\{ a \mid a : \textbf{TYPE}_{ICn} T \}$ and such that for any $a$ in the domain of $f$, $f(a) : \textbf{TYPE}_{ICn} F(a)$.

We might say that on this view dependent function types are “semi-intensional” in that they depend on there being a type of types for their definition but they do not introduce types as arguments to predicates and do not involve the definition of orders of types in terms of the types of the next lower order.
Intensional modal systems of complex types

An *intensional modal system of complex types based on* \( \mathcal{M} \) is a family, indexed by the natural numbers, of families of quadruples indexed by members of \( \mathcal{M} \):

\[
\text{TYPE}_{IMC} = \langle \text{Type}^n, \text{BType}, \langle \text{PType}^n, \text{Pred}, \text{ArgIndices}, \text{Arity} \rangle, \mathcal{M}_n \rangle_{\mathcal{M} \in \mathcal{M}, n \in \text{Nat}}
\]

where:

1. for each \( n \), \( \langle \text{Type}^n, \text{BType}, \langle \text{PType}^n, \text{Pred}, \text{ArgIndices}, \text{Arity} \rangle, \mathcal{M}_n \rangle_{\mathcal{M} \in \mathcal{M}} \) is a modal system of complex types based on \( \{ \mathcal{M}_n \mid \mathcal{M} \in \mathcal{M} \} \)

2. for each \( \mathcal{M} \in \mathcal{M}, \langle \text{Type}^n, \text{BType}, \langle \text{PType}^n, \text{Pred}, \text{ArgIndices}, \text{Arity} \rangle, \mathcal{M}_n \rangle_{n \in \text{Nat}} \) is an intensional system of complex types
Outline

Rich type theory, cognition and the formal semantics tradition

Type theory without records

Type theory with dependent record types

Dependent types
Are ptypes the only types of situations?

- suppose $b$ is Bill, a boy and $d$ is Dinah, a dog
- we have allowed ourselves the ptype $\text{hug}(b,d)$, the type of situation where Bill hugs Dinah
- but we have not allowed ourselves the type of “boy hugs dog” situations corresponding to a boy hugs a dog
- there are a number of ways to construct such types in rich type theories – we use record types
A boy hugs a dog

Record type – “a collection of labelled types”

\[
\begin{bmatrix}
x & : & \text{Ind} \\
\text{c}_{\text{boy}} & : & \text{boy}(x) \\
y & : & \text{Ind} \\
\text{c}_{\text{dog}} & : & \text{dog}(y) \\
e & : & \text{hug}(x,y)
\end{bmatrix}
\]
A boy hugs a dog

Record type – “a collection of labelled types”
...not quite because of dependencies

\[
\begin{bmatrix}
  x & : & Ind \\
  c_{\text{boy}} & : & \text{boy}(x) \\
  y & : & Ind \\
  c_{\text{dog}} & : & \text{dog}(y) \\
  e & : & \text{hug}(x,y)
\end{bmatrix}
\]
The official notation

\[
\begin{aligned}
    x & : \text{Ind} \\
    c_{\text{boy}} & : \langle \lambda v : \text{Ind}(\text{boy}(v)), \langle x \rangle \rangle, \\
    y & : \text{Ind} \\
    c_{\text{dog}} & : \langle \lambda v : \text{Ind}(\text{dog}(v)), \langle y \rangle \rangle \\
    e & : \langle \lambda v_1 : \text{Ind}(\lambda v_2 : \text{Ind}(\text{hug}(v_1, v_2))), \langle x, y \rangle \rangle
\end{aligned}
\]
A record of type *a boy hugs a dog*

\[
\begin{cases}
  x = a \\
  c_{\text{boy}} = s_1 \\
  y = b \\
  c_{\text{dog}} = s_2 \\
  e = s_3
\end{cases}
\]

where:
- \( a : \text{Ind} \)
- \( s_1 : \text{boy}(a) \)
- \( b : \text{Ind} \)
- \( s_2 : \text{dog}(b) \)
- \( s_3 : \text{hug}(a,b) \)
Two important facts about records

- You can construct a record of a given type just in case there are objects of the types required by its fields – i.e. the labelling is arbitrary (records as indexed (labelled) multisets)
Two important facts about records

- You can construct a record of a given type just in case there are objects of the types required by its fields – i.e. the labelling is arbitrary (records as indexed (labelled) multisets)
- A record of a given type may contain more fields than required by the type – this record also belongs to a subtype of the type where the extra fields are added
Why should record types be interesting for linguists?

▶ they allow us to model discourse representation structures
Why should record types be interesting for linguists?

- they allow us to model discourse representation structures
- they allow us to model feature structures
Why should record types be interesting for linguists?

▶ they allow us to model discourse representation structures
▶ they allow us to model feature structures
▶ they allow us to model dialogue game boards (information states)
Why should record types be interesting for linguists?

- they allow us to model discourse representation structures
- they allow us to model feature structures
- they allow us to model dialogue game boards (information states)
- they allow us to model frames (as in frame semantics and FrameNet)
Records

A *record* is a finite set of ordered pairs (called *fields*) which is the graph of a function. If \( r \) is a record and \( \langle \ell, v \rangle \) is a field in \( r \) we call \( \ell \) a *label* and \( v \) a *value* in \( r \) and we use \( r.\ell \) to denote \( v \). \( r.\ell \) is called a *path* in \( r \).

A record \( r \) is *well-typed* with respect to a system of types \( \text{TYPE} \) with set of types \( \text{Type} \) and a set of labels \( L \) iff for each field \( \langle \ell, a \rangle \in r, \ell \in L \) and either \( a :_{\text{TYPE}} T \) for some \( T \in \text{Type} \) or \( a \) is itself a record which is well-typed with respect to \( \text{TYPE} \) and \( L \).
Adding (non-depent) record types to a system of complex types

A system of complex types $\text{TYPE}_C = \langle \text{Type}, \text{BType}, \langle \text{PType}, \text{Pred}, \text{ArgIndices}, \text{Arity}\rangle, \langle \text{A}, \text{F}\rangle \rangle$ has record types based on $\langle L, \text{RType}\rangle$, where $L$ is a countably infinite set (of labels) and $\text{RType} \subseteq \text{Type}$, where $\text{RType}$ is defined by:

1. $\text{Rec} \in \text{RType}$
2. $r : \text{TYPE}_C \text{Rec}$ iff $r$ is a well-typed record with respect to $\text{TYPE}_C$ and $L$.
3. if $\ell \in L$ and $T \in \text{Type}$, then $\{\langle \ell, T\rangle\} \in \text{RType}$.
4. $r : \text{TYPE}_C \{\langle \ell, T\rangle\}$ iff $r : \text{TYPE}_C \text{Rec}$, $\langle \ell, a\rangle \in r$ and $a : \text{TYPE}_C T$.
5. if $R \in \text{RType}$, $\ell \in L$, $\ell$ does not occur as a label in $R$ (i.e. there is no field $\langle \ell', T'\rangle$ in $R$ such that $\ell' = \ell$), then $R \cup \{\langle \ell, T\rangle\} \in \text{RType}$.
6. $r : \text{TYPE}_C R \cup \{\langle \ell, T\rangle\}$ iff $r : \text{TYPE}_C R$, $\langle \ell, a\rangle \in r$ and $a : \text{TYPE}_C T$. 
Record types in intensional type systems

An intensional system of complex types \( \text{TYPE}_{IC} = \langle \text{Type}^n, \text{BType}, \langle \text{PType}^n, \text{Pred}, \text{ArgIndices}, \text{Arity} \rangle, \langle A, F^n \rangle \rangle_{n \in \text{Nat}} \) has record types based on \( \langle L, \text{RType}^n \rangle \) for each \( n \), \( \langle \text{Type}^n, \text{BType}, \langle \text{PType}^n, \text{Pred}, \text{ArgIndices}, \text{Arity} \rangle, \langle A, F^n \rangle \rangle \) has record types based on \( \langle L, \text{RType}^n \rangle \) and

1. for each \( n \), \( \text{RType}^n \subseteq \text{RType}^{n+1} \)
2. for each \( n > 0 \), \( \text{RecType}^n \in \text{RType}^n \)
3. for each \( n > 0 \), \( T : \text{TYPE}_{ICn} \) \( \text{RecType}^n \) iff \( T \in \text{RType}^{n-1} \)
Dependent record types in intensional type systems

An intensional system of complex types $\text{TYPE}_{IC} = \langle \text{Type}^n, \text{BType}, \langle \text{PType}^n, \text{Pred}, \text{ArgIndices}, \text{Arity} \rangle, \langle A, F^n \rangle \rangle_{n \in \text{Nat}}$ has dependent record types based on $\langle L, \text{RType}^n \rangle_{n \in \text{Nat}}$, if it has records types based on $\langle L, \text{RType}^n \rangle_{n \in \text{Nat}}$ and for each $n > 0$

1. if $R$ is a member of $\text{RType}^n$, $\ell \in L$ not occurring as a label in $R$, $T_1, \ldots, T_m \in \text{Type}^n$, $R.\pi_1, \ldots, R.\pi_m$ are paths in $R$ and $F$ is a function of type $(T_1 \to \ldots \to (T_m \to \text{Type}^n)\ldots)$, then $R \cup \{\langle \ell, \langle F, \langle \pi_1, \ldots, \pi_m \rangle \rangle \rangle \} \in \text{RType}^n$.

2. $r : \text{TYPE}_{IC_n} R \cup \{\langle \ell, \langle F, \langle \pi_1, \ldots, \pi_m \rangle \rangle \rangle \} \iff r : \text{TYPE}_{IC_n} R$, $\langle \ell, a \rangle$ is a field in $r$, $r.\pi_1 : \text{TYPE}_{IC_n} T_1, \ldots$, $r.\pi_m : \text{TYPE}_{IC_n} T_m$ and $a : \text{TYPE}_{IC_n} F(r.\pi_1, \ldots, r.\pi_m)$. 
Outline

Rich type theory, cognition and the formal semantics tradition

Type theory without records

Type theory with dependent record types

Dependent types
Functions which return types

- intensional type systems with function types include the type of functions from objects of arbitrary type $T$ to types: $(T \rightarrow Type)$
Adapting type theory with records for natural language semantics

Dependent types

Functions which return types

- Intensional type systems with function types include the type of functions from objects of arbitrary type $T$ to types: $(T \to Type)$

- A function of this type:
  \[ \lambda x : T_1 (T_2) \]
Functions which return types

- Intensional type systems with function types include the type of functions from objects of arbitrary type $T$ to types:
  $$(T \to Type)$$
- A function of this type:
  $$\lambda x : T_1 (T_2)$$
- More generally
  $$\lambda x_1 : T_1 (\lambda x_2 : T_2 \ldots \lambda x_n : T_n (T_{n+1}) \ldots) : (T_1 \to (T_2 \to \ldots (T_n \to Type)) \ldots)$$
Functions which return types

- Intensional type systems with function types include the type of functions from objects of arbitrary type \( T \) to types:
  \[( T \rightarrow Type)\]
- A function of this type:
  \( \lambda x : T_1 \ (T_2) \)
- More generally
  \( \lambda x_1 : T_1(\lambda x_2 : T_2 \ldots \lambda x_n : T_n(T_{n+1})\ldots) : (T_1 \rightarrow (T_2 \rightarrow \ldots (T_n \rightarrow Type))\ldots) \)
- Such functions can be thought of as dependent types
Record types as “propositions”

Record type:

\[
\begin{bmatrix}
\begin{aligned}
\times & : & \text{Ind} \\
\text{c}_1 & : & \text{boy}(\times) \\
y & : & \text{Ind} \\
\text{c}_2 & : & \text{dog}(y) \\
\text{c}_3 & : & \text{hug}(\times, y)
\end{aligned}
\end{bmatrix}
\]
Record types as “propositions”

Record type:

\[
\begin{aligned}
  x & : Ind \\
  c_1 & : \text{boy}(x) \\
  y & : Ind \\
  c_2 & : \text{dog}(y) \\
  c_3 & : \text{hug}(x,y)
\end{aligned}
\]

Record:

\[
\begin{aligned}
  x & = a \\
  c_1 & = p_1 \\
  y & = b \\
  c_2 & = p_2 \\
  c_3 & = p_3
\end{aligned}
\]

where $a$, $b$ are of type $Ind$, individuals $p_1$ is a witness for $\text{boy}(a)$ $p_2$ is a witness for $\text{dog}(b)$ $p_3$ is a witness for $\text{hug}(a, b)$
Contents of constituents in compositional semantics

- **hugged**

\[ \lambda r: [x: \text{Ind}] ( \begin{array}{l} c_{\text{tns}} : \text{e<s-event} \\ e : \text{hug}(r.x) \end{array} ) \]

- Montague’s \( \langle e, t \rangle \) becomes \( [x: \text{Ind}] \rightarrow \text{RecType} \)
Frames as arguments

- (the temperature) rose
- $\lambda r: [x: \text{Ind}] \begin{bmatrix} c_{\text{tns}} & \text{e$<$s-event} \\ c_{\text{rise}} & \text{rise}(r, \text{e-time}) \end{bmatrix}$
Questions

- **wh-questions**: who left – $\lambda r: [x: \text{Ind}] \ (e: \text{leave}(x))$
- **yes/no-question**: did Bo leave – $\lambda r: \text{Rec} \ ([e: \text{leave}(bo)])$
Montaguesque meanings/Kaplanian characters

- functions from contexts to types

\[
\lambda r: \left[ \begin{array}{l}
  x : \text{Ind} \\
  c_1 : \text{named}(x, "Kim") \\
\end{array} \right] \left( \begin{array}{l}
  y : \text{Ind} \\
  c_2 : \text{dog}(y) \\
  c_3 : \text{hug}(r.x, y) \\
\end{array} \right)
\]
Montaguesque meanings/Kaplanian characters

- functions from contexts to types

\[ \lambda r: \left[ \begin{array}{l}
  x : Ind \\
  c_1 : \text{named}(x, "Kim")
\end{array} \right] \left( \begin{array}{l}
  y : Ind \\
  c_2 : \text{dog}(y) \\
  c_3 : \text{hug}(r.x,y)
\end{array} \right) \]

- incremental specification of context types

\[ \lambda r: \left[ \begin{array}{l}
  x=b : Ind \\
  c_1 : \text{named}(x, "Kim")
\end{array} \right] \left( \begin{array}{l}
  y : Ind \\
  \quad \left( \begin{array}{l}
    c_2 : \text{dog}(y) \\
    c_3 : \text{hug}(r.x,y)
  \end{array} \right)
\end{array} \right) \]
Grammar rules

- a function from constituents of certain types to a type for their combination
Grammar rules

- a function from constituents of certain types to a type for their combination
- observations of linguistic acts of certain types lead you to suppose that there is a composite object of another type
Grammar rules

- a function from constituents of certain types to a type for their combination
- observations of linguistic acts of certain types lead you to suppose that there is a composite object of another type
- \texttt{s_np_vp}
  \[
  \lambda r_1: [\text{cat}=\text{np}: \text{Cat}] \\
  (\lambda r_2: [\text{cat}=\text{vp}: \text{Cat}] \\
  [\text{cat}=\text{s}: \text{Cat} \\
  (\text{daughters:} [\text{fst}=r_1: [\text{cat}=\text{np}: \text{Cat}]] \\
  \text{rst:} [\text{fst}=r_2: [\text{cat}=\text{vp}: \text{Cat}]] \\
  \text{rst}=\text{nil:} [\text{Sign}]))
  \]
\]
Grammar rules II
Grammar rules II

- **binary_sign**
  \[
  \lambda r_1: \text{Sign} \\
  (\lambda r_2: \text{Sign} \\
  (\text{Sign} \land \text{daughters: } \left[\begin{array}{l}
  \text{fst} = r_1: \text{Sign} \\
  \text{rst}: \left[\begin{array}{l}
  \text{rst} = r_2: \text{Sign} \\
  \text{rst} = \text{nil}: [\text{Sign}]\end{array}\right]\end{array}\right]) ))
  \]
Grammar rules II

- **binary_sign**
  \[ \lambda r_1 : \text{Sign} \]
  \[ (\lambda r_2 : \text{Sign} \]
  \[ (\text{Sign} \land \left[ \begin{array}{c}
  \text{daughters:} \\
  \text{rst:} \\
  \text{rst=\text{nil}:[Sign]}
  \end{array} \right] ) ] ) \]

- \[ [f : T_1] \land [g : T_2] = [f : T_1] \]
  \[ [f : T_1] \land [f : T_2] = [f : T_1 \land T_2] \]
Grammar rules II

- **binary_sign**
  \[ \lambda r_1 : Sign \]
  \[(\lambda r_2 : Sign \]
  \[\text{(Sign} \land \left[ \text{daughters:} \begin{cases} \text{fst=}r_1 : Sign \\ \text{rst:} \begin{cases} \text{fst=}r_2 : Sign \\ \text{rst=}\text{nil:}[\text{Sign}] \end{cases} \end{cases} \right] \right) \]

- \[ [f : T_1] \land [g : T_2] = [f : T_1] \]
  \[ [f : T_1] \land [f : T_2] = [f : T_1 \land T_2] \]

- **binary_sign \land s_np_vp**
Grammar rules II

- **binary_sign**
  \[ \lambda r_1 : \text{Sign} \]
  \[ (\lambda r_2 : \text{Sign} \]
  \[ (\text{Sign} \land \text{daughters:} \begin{cases} \text{fst} = r_1 : \text{Sign} \\ \text{rst} : \begin{cases} \text{fst} = r_2 : \text{Sign} \\ \text{rst} = \text{nil} : [\text{Sign}] \end{cases} \end{cases} ) ) \]

- \([f : T_1] \land [g : T_2] = [f : T_1] \]
  \([f : T_1] \land [f : T_2] = [f : T_1 \land T_2] \]

- **binary_sign \land s \_ np \_ vp**

- \[ \lambda r_1 : T_1 \ldots \lambda r_n : T_n(T_{n+1}) \land \lambda r_1 : T'_1 \ldots \lambda r_n : T'_n(T'_{n+1}) = \]
  \[ \lambda r_1 : T_1 \land T'_1 \ldots \lambda r_n : T_n \land T'_n(T_{n+1} \land T'_{n+1}) \]
Adapting type theory with records for natural language semantics

- Dependent types

Update rules in dialogue

- a function from an information state and a new dialogue contribution to a type for a new information state

- **downdate_agenda**

  \[
  \lambda r: \left[ \text{private: } \left[ \text{agenda: } [\text{RecType}] \right] \right] \\
  \begin{align*}
  &\text{moves: } \{ \text{Move} \} \\
  &\text{mtype: } \text{RecType}
  \end{align*}

  \lambda e: \\
  \begin{align*}
  &c_{\text{fst}}: \text{fst}(\text{mtype, } r.\text{private.agenda}) \\
  &m: \text{mtype} \land \left[ \text{actor=} \text{Self: } \text{Ind} \right] \\
  &c_{\in}: m \in \text{moves} \\
  \left( \begin{array}{l}
  \text{private: } \left[ \text{agenda=} \text{rst}(r.\text{private.agenda}): [\text{RecType}] \right] \\
  \text{shared: } \left[ \text{latest_utterance: } \left[ \text{moves=} e.\text{moves: } \{ \text{Move} \} \right] \right]
  \end{array} \right)
  \end{align*}
Dependent types and inference

- both grammar rules and update rules can be considered to be functions which enable inference
Dependent types and inference

- both grammar rules and update rules can be considered to be functions which enable inference
- “observe something of type $T_1$ – conclude that type $T_2$ is non-empty”
Dependent types and inference

- both grammar rules and update rules can be considered to be functions which enable inference
- “observe something of type $T_1$ – conclude that type $T_2$ is non-empty”
- Aristotelian-Breitholtzian enthymemes are similar Breitholtz (2010); Breitholtz and Cooper (2011); Breitholtz (2011)
Dependent types and inference

- both grammar rules and update rules can be considered to be functions which enable inference
- “observe something of type $T_1$ – conclude that type $T_2$ is non-empty”
- Aristotelian-Breitholtzian enthymemes are similar Breitholtz (2010); Breitholtz and Cooper (2011); Breitholtz (2011)

$$\lambda r:\left[ x:Ind_{\text{has fever}}:\text{has fever}(x) \right] \left[ c_{\text{sick}}:\text{sick}(r.x) \right]$$
Dependent types and inference

- both grammar rules and update rules can be considered to be functions which enable inference
- “observe something of type $T_1$ – conclude that type $T_2$ is non-empty”
- Aristotelian-Breitholtzian enthymemes are similar Breitholtz (2010); Breitholtz and Cooper (2011); Breitholtz (2011)

\[ \lambda r: \begin{bmatrix} x: \text{Ind} \\ c_{\text{has fever}}: \text{has fever}(x) \end{bmatrix} \left( \left[ c_{\text{sick}}: \text{sick}(r.x) \right] \right) \]

\[ \lambda r: \begin{bmatrix} x: \text{Ind} \\ c_{\text{breathe rapidly}}: \text{breathe rapidly}(x) \end{bmatrix} \left( \left[ c_{\text{has fever}}: \text{has fever}(r.x) \right] \right) \]
Conclusion

- Not covered here
  - regular (string) types – relating to Fernando’s string theory of events (Fernando, 2004, 2011)
  - merging of record types – corresponding to graph unification in computational linguistics and feature based theories
Conclusion

- Not covered here
  - regular (string) types – relating to Fernando’s string theory of events (Fernando, 2004, 2011)
  - merging of record types – corresponding to graph unification in computational linguistics and feature based theories
- type theory with a cognitive spin
Conclusion

- Not covered here
  - regular (string) types – relating to Fernando’s string theory of events (Fernando, 2004, 2011)
  - merging of record types – corresponding to graph unification in computational linguistics and feature based theories
- type theory with a cognitive spin
- offers the hope of a type theoretical foundation for linguistics
Conclusion

- Not covered here
  - regular (string) types – relating to Fernando’s string theory of events (Fernando, 2004, 2011)
  - merging of record types – corresponding to graph unification in computational linguistics and feature based theories
- type theory with a cognitive spin
- offers the hope of a type theoretical foundation for linguistics
- but perhaps it’s not quite like anybody else’s type theory...
Bibliography I


Bibliography III


Bibliography IV


Bibliography V


Bibliography VI


