A theory of events and situations

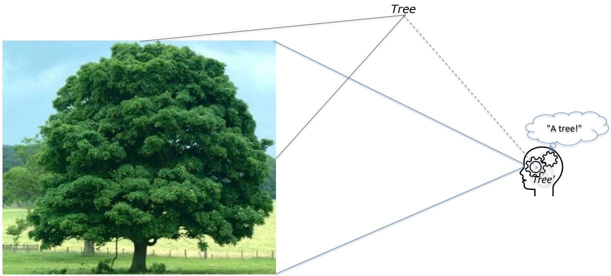
Robin Cooper
University of Gothenburg
Jonathan Ginzburg
Université Paris-Diderot, Sorbonne Paris-Cité

An Introduction to Semantics using Type Theory with Records
ESSLLI 2012
Lecture 1, part 2

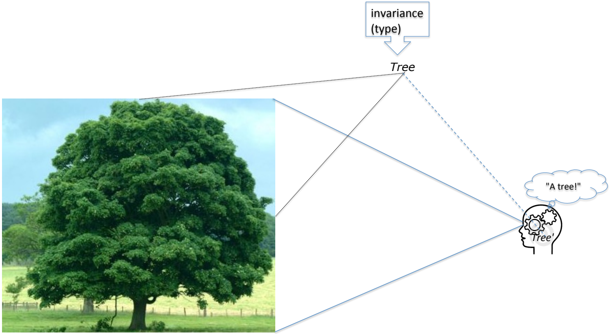
Outline

Outline

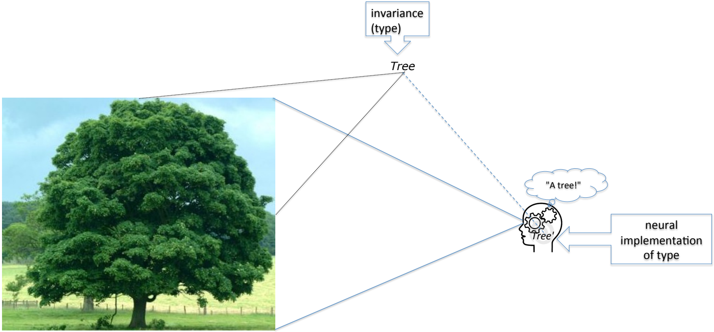
Seeing a tree (a simulation view)



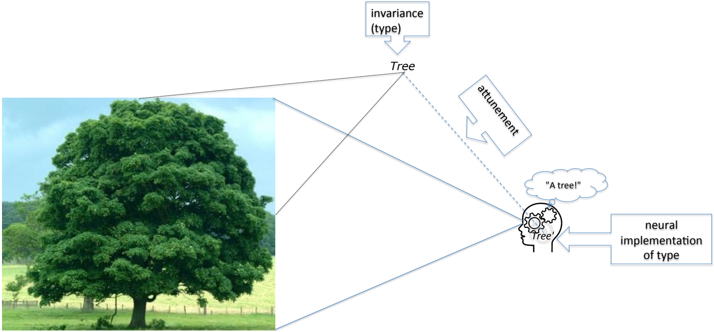
Seeing a tree (a simulation view)



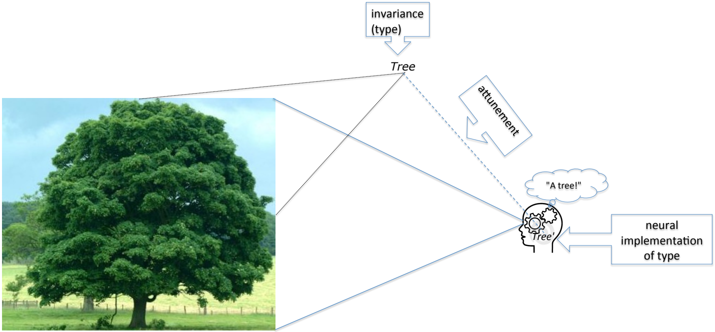
Seeing a tree (a simulation view)



Seeing a tree (a simulation view)



Seeing a tree (a simulation view)



Gibson (1986); Barwise and Perry (1983)

Judgement

- (An agent judges that) object a is of type T.
- $a : T$

Outline

TTR: Type theory with records

- 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/ch1-draft111114.pdf` for some current work in progress we will discuss here

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) provide a more general 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*

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) provide a more general 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*
- two fundamental questions when characterizing a type theory:
  - what types are there?
  - for any type, what are the objects of that type?

Basic types

- basic types (as opposed to complex ones) are types which are not constructed from smaller parts

Basic types

- basic types (as opposed to complex ones) are types which are not constructed from smaller parts
- a system of basic types contains a set of such types, **Type**

Basic types

- basic types (as opposed to complex ones) are types which are not constructed from smaller parts
- a system of basic types contains a set of such types, **Type**
- examples: *Ind* ("individual"), *Time* ("time point")

Basic types

- basic types (as opposed to complex ones) are types which are not constructed from smaller parts
- a system of basic types contains a set of such types, **Type**
- examples: *Ind* ("individual"), *Time* ("time point")
- a system of basic types also contains a function, *A*, which assigns a set of objects to each of the types in **Type**

Basic types

- basic types (as opposed to complex ones) are types which are not constructed from smaller parts
- a system of basic types contains a set of such types, **Type**
- examples: *Ind* ("individual"), *Time* ("time point")
- a system of basic types also contains a function, A, which assigns a set of objects to each of the types in **Type**
- $a : T$ iff $a \in A(T)$

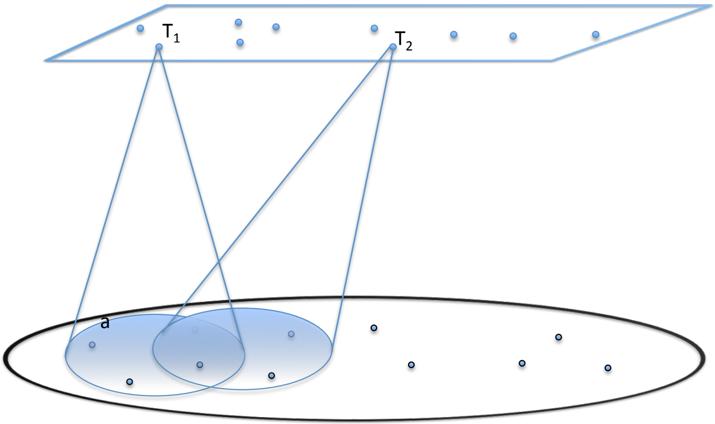
Intensionality

- 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.
- Consequence: you can have two distinct types which have the same set of objects associated with them, i.e. $A(T_1) = A(T_2)$, $a : T_1$ just in case $a : T_2$. The types are *intensional*.

$a : T_1$



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?

Why is structure important?

- increases intensionality (e.g. same "content", different structure)

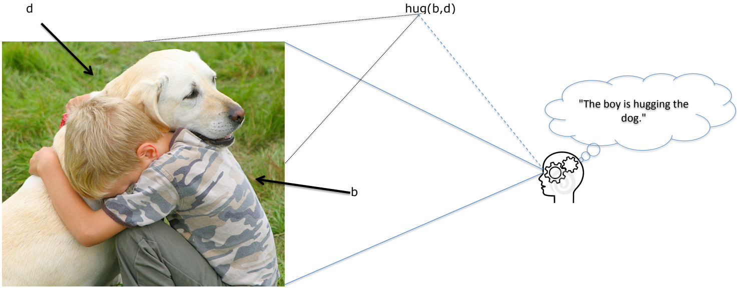
Why is structure important?

- 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



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:
  - Arity(run) $= \langle \mathit{Ind} \rangle$
  - Arity(hug) $= \langle \mathit{Ind}, \mathit{Ind} \rangle$
- We might also want to include time intervals and locations as part of the arities of these predicates

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, **PType**, in addition to basic types
- **PType** will be based on a set of predicates with their arities
- **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: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
- **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.

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

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 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”

$$\begin{bmatrix} \mathrm{x} & : & \mathit{Ind} \\ \mathrm{c}_{\mathrm{boy}} & : & \mathrm{boy(x)} \\ \mathrm{y} & : & \mathit{Ind} \\ \mathrm{c}_{\mathrm{dog}} & : & \mathrm{dog(y)} \\ \mathrm{e} & : & \mathrm{hug(x,y)} \end{bmatrix}$$

A boy hugs a dog

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

$$\begin{bmatrix} \mathrm{x} & : & \mathit{Ind} \\ \mathrm{c}_{\text{boy}} & : & \mathrm{boy(x)} \\ \mathrm{y} & : & \mathit{Ind} \\ \mathrm{c}_{\text{dog}} & : & \mathrm{dog(y)} \\ \mathrm{e} & : & \mathrm{hug(x,y)} \end{bmatrix}$$

The official notation

$$\left[\begin{array}{lll} \mathrm{x} & : & \mathit{Ind} \\ \mathrm{c}_{\text{boy}} & : & \langle \lambda v : \mathit{Ind}(\text{boy}(v)), \langle \mathrm{x} \rangle \rangle, \\ \mathrm{y} & : & \mathit{Ind} \\ \mathrm{c}_{\text{dog}} & : & \langle \lambda v : \mathit{Ind}(\text{dog}(v)), \langle \mathrm{y} \rangle \rangle \\ \mathrm{e} & : & \langle \lambda v_1 : \mathit{Ind}(\lambda v_2 : \mathit{Ind}(\text{hug}(v_1, v_2))), \\ & & \quad \langle \mathrm{x}, \mathrm{y} \rangle \rangle \end{array}\right]$$

A record of type *a boy hugs a dog*

$$\begin{bmatrix} \mathsf{x} & = & a \\ \mathsf{c}_{\mathrm{boy}} & = & s_1 \\ \mathsf{y} & = & b \\ \mathsf{c}_{\mathrm{dog}} & = & s_2 \\ \mathsf{e} & = & s_3 \end{bmatrix}$$

where:

$$\begin{array}{l} a : \mathit{Ind} \\ s_1 : \mathrm{boy}(a) \\ b : \mathit{Ind} \\ s_2 : \mathrm{dog}(b) \\ s_3 : \mathrm{hug}(a,b) \end{array}$$

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

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
- 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 are record types interesting for linguists?

- they allow us to model discourse representation structures

Why are record types interesting for linguists?

- they allow us to model discourse representation structures
- they allow us to model feature structures

Why are record types 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 are record types 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)

Outline

Summary

- Type theory as a formal theory related to perception
- A basic introduction to TTR:
  - basic types (e.g. *Ind*)
  - ptypes (e.g. hug(Sam,Dinah))
  - models which supply objects of basic types and ptypes
  - record types
  - mentioned some of their linguistic applications

Bibliography I

Barwise, Jon and John Perry (1983) *Situations and Attitudes*, Bradford Books, MIT Press, Cambridge, Mass.

Cooper, Robin (2005a) Austinian truth, attitudes and type theory, *Research on Language and Computation*, Vol. 3, pp. 333–362.

Cooper, Robin (2005b) Records and Record Types in Semantic Theory, *Journal of Logic and Computation*, Vol. 15, No. 2, pp. 99–112.

Cooper, Robin (2012) Type Theory and Semantics in Flux, in R. Kempson, N. Asher and T. Fernando (eds.), *Handbook of the Philosophy of Science*, Vol. 14: Philosophy of Linguistics, pp. 271–323, Elsevier BV. General editors: Dov M. Gabbay, Paul Thagard and John Woods.

Bibliography II

Gibson, James J. (1986) *The Ecological Approach to Visual Perception*, Lawrence Erlbaum Associates.

Martin-Löf, Per (1984) *Intuitionistic Type Theory*, Bibliopolis, Naples.

Montague, Richard (1973) The Proper Treatment of Quantification in Ordinary English, in J. Hintikka, J. Moravcsik and P. Suppes (eds.), *Approaches to Natural Language: Proceedings of the 1970 Stanford Workshop on Grammar and Semantics*, pp. 247–270, D. Reidel Publishing Company, Dordrecht.

Montague, Richard (1974) *Formal Philosophy: Selected Papers of Richard Montague*, Yale University Press, New Haven, ed. and with an introduction by Richmond H. Thomason.