Semantics, action and perception - an overview of TTR

Robin Cooper
University of Gothenburg

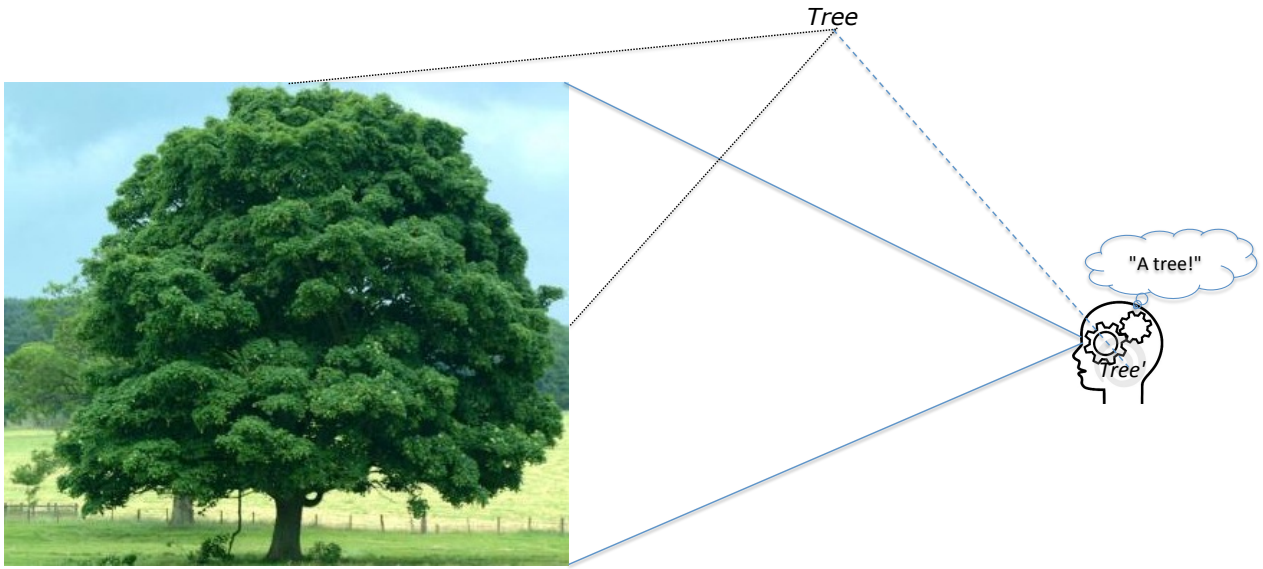
LT2318: Artificial Intelligence: Cognitive Systems, HT2018

https://sites.google.com/site/typetheorywithrecords/slides

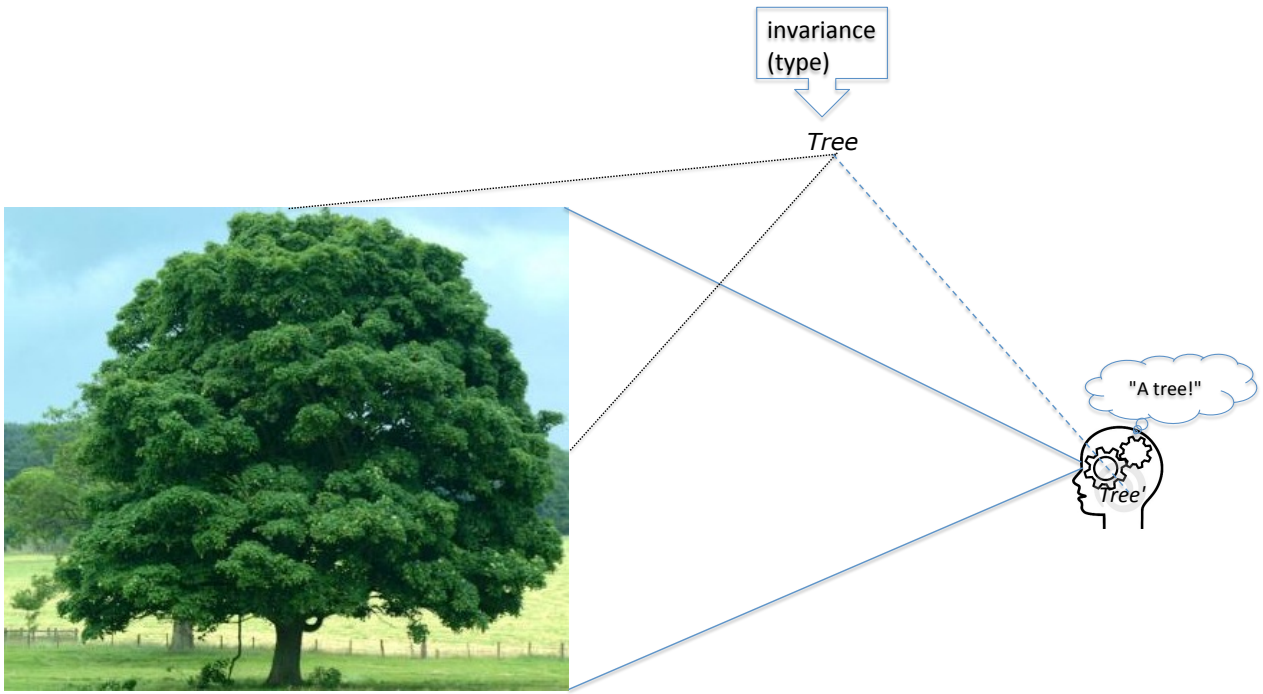
Outline

Outline

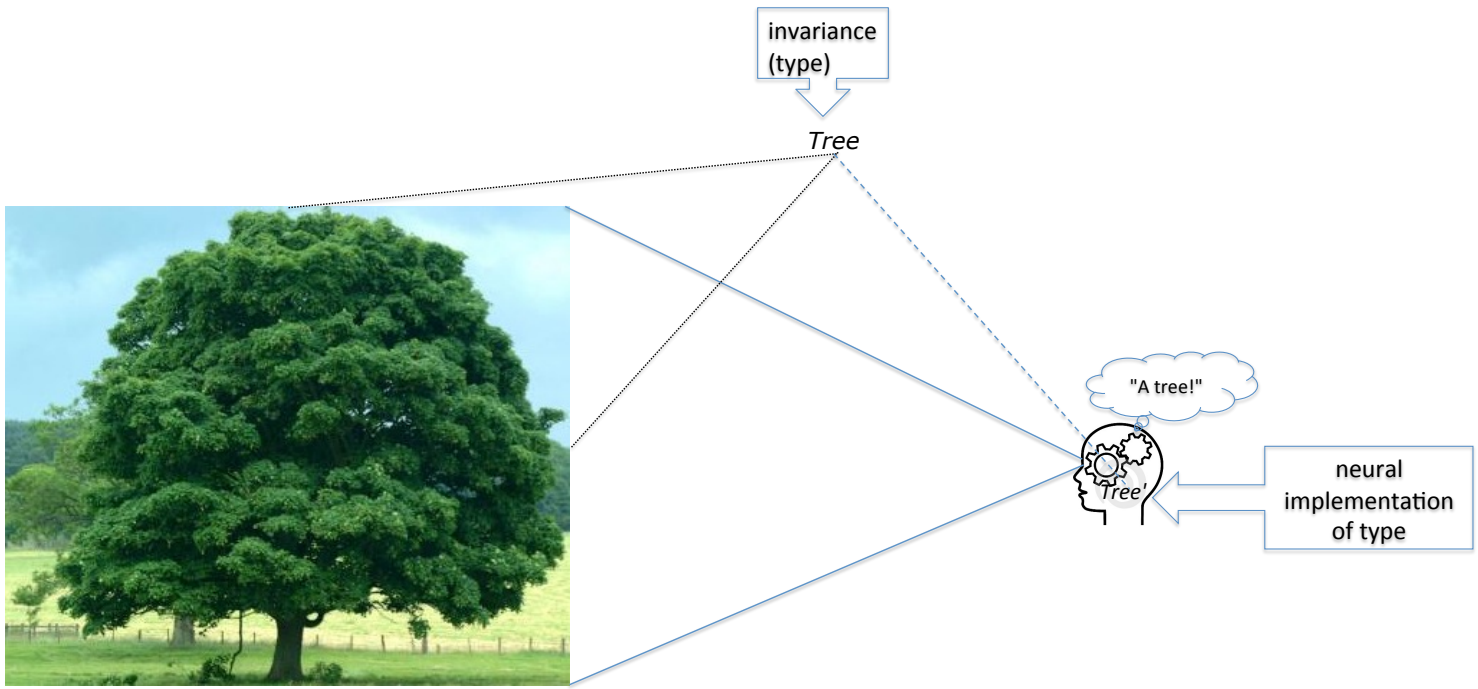
Seeing a tree (a simulation view)



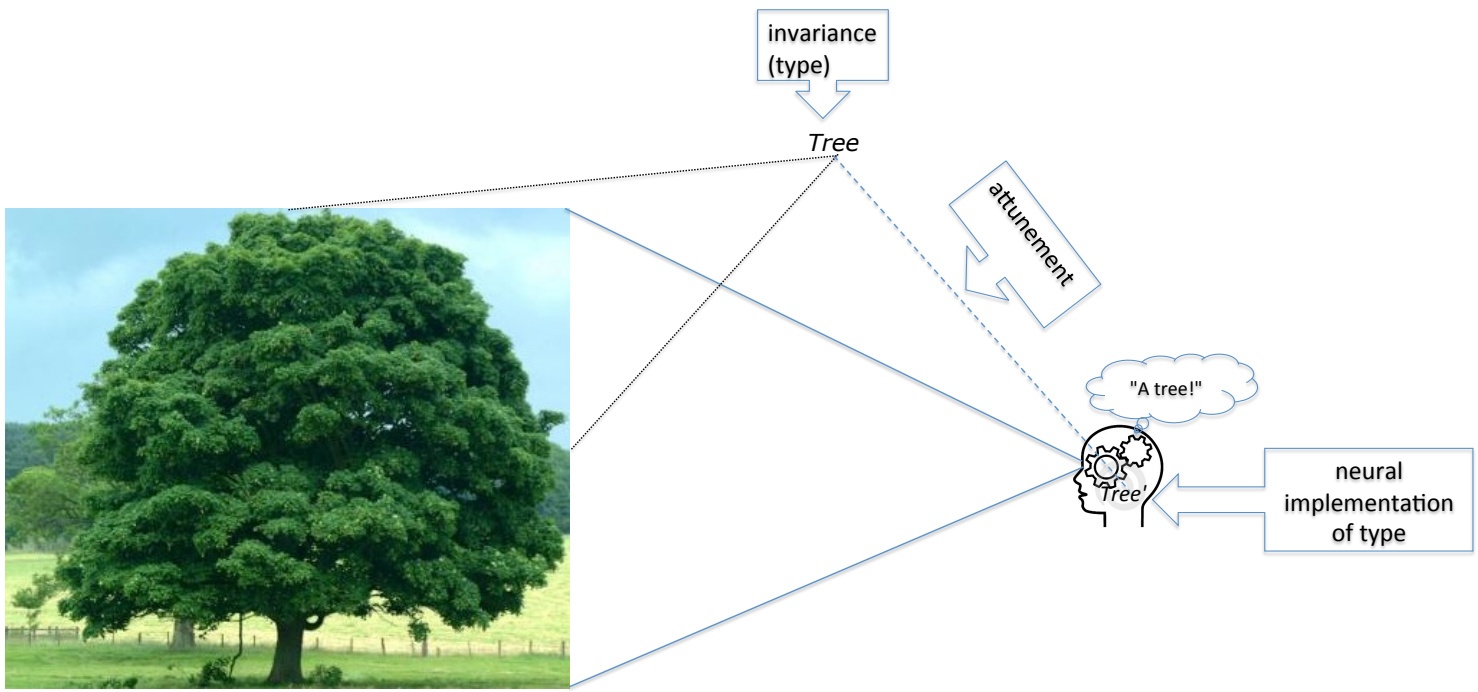
Seeing a tree (a simulation view)



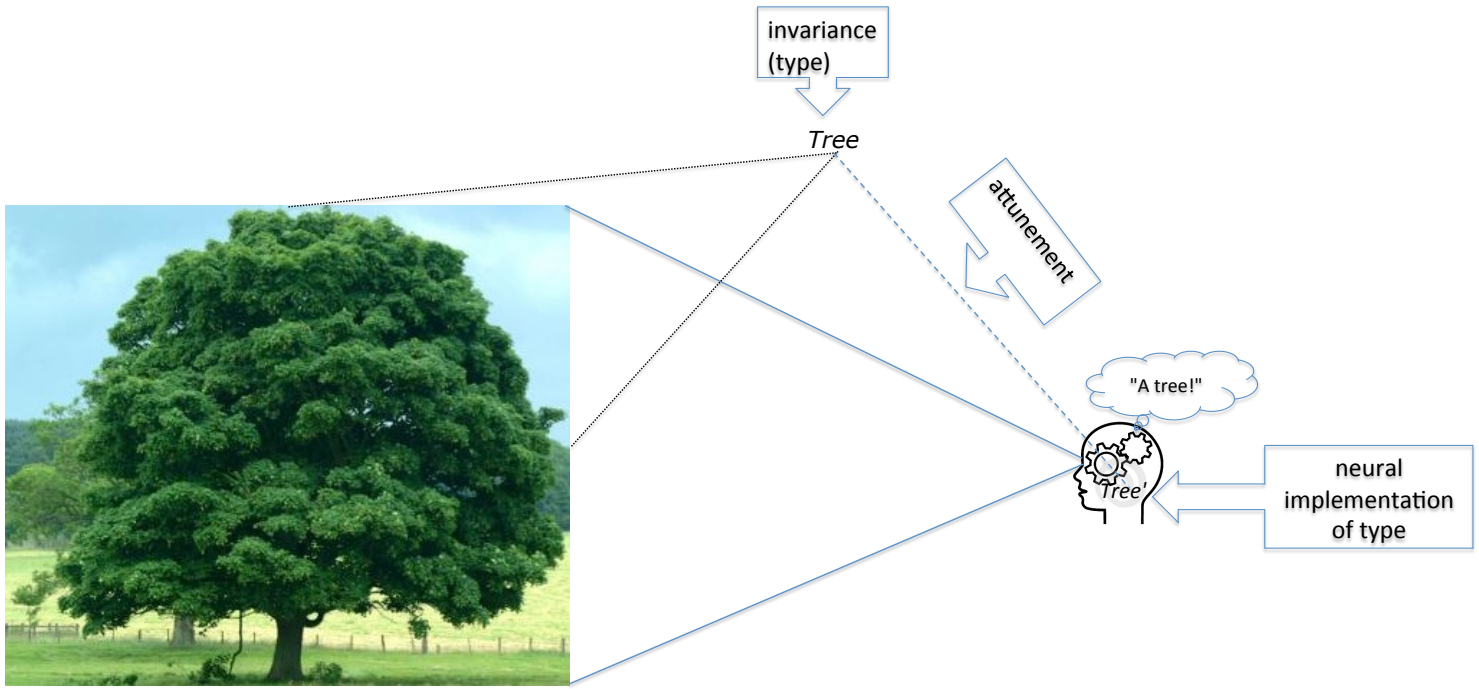
Seeing a tree (a simulation view)



Seeing a tree (a simulation view)



Seeing a tree (a simulation view)



Gibson (1986); Barwise and Perry (1983)

Outline

TTR: Type theory with records I

- The most recent published reference for the details is Cooper (2012)
- an overview with some details Cooper and Ginzburg (2015), see also `https://sites.google.com/site/semttrgbg/schedule` for lectures on youtube based on this material
- Also Cooper (2005a) for an earlier detailed treatment, Cooper (2005b) for relation to various semantic theories and Cooper (2017) for some linguistic motivation.
- `https://sites.google.com/site/typetheorywithrecords/drafts` for a book manuscript in (slow) progress, daily dump on `https://github.com/robincooper/ttl/blob/master/ttl.pdf`

TTR: Type theory with records II

- In general TTR references can be found at `https://sites.google.com/site/typetheorywithrecords/publications`
- implementation `PyTTR` on `https://github.com/GU-CLASP/pyttr`

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 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*

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

Judgement

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

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**
- example: *Ind* ("individual")

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**
- example: *Ind* ("individual")
- 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**
- example: *Ind* ("individual")
- 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 $a \in A(T)$

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**
- example: *Ind* ("individual")
- 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 $a \in A(T)$
- in `pyttr`

Witness conditions for types

- witness conditions for types can be external to type theory, for example, classifiers
- Arild Matsson's MLT thesis (`https://github.com/arildm/imagettr-thesis`)
- in `pyttr`

Intensionality

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

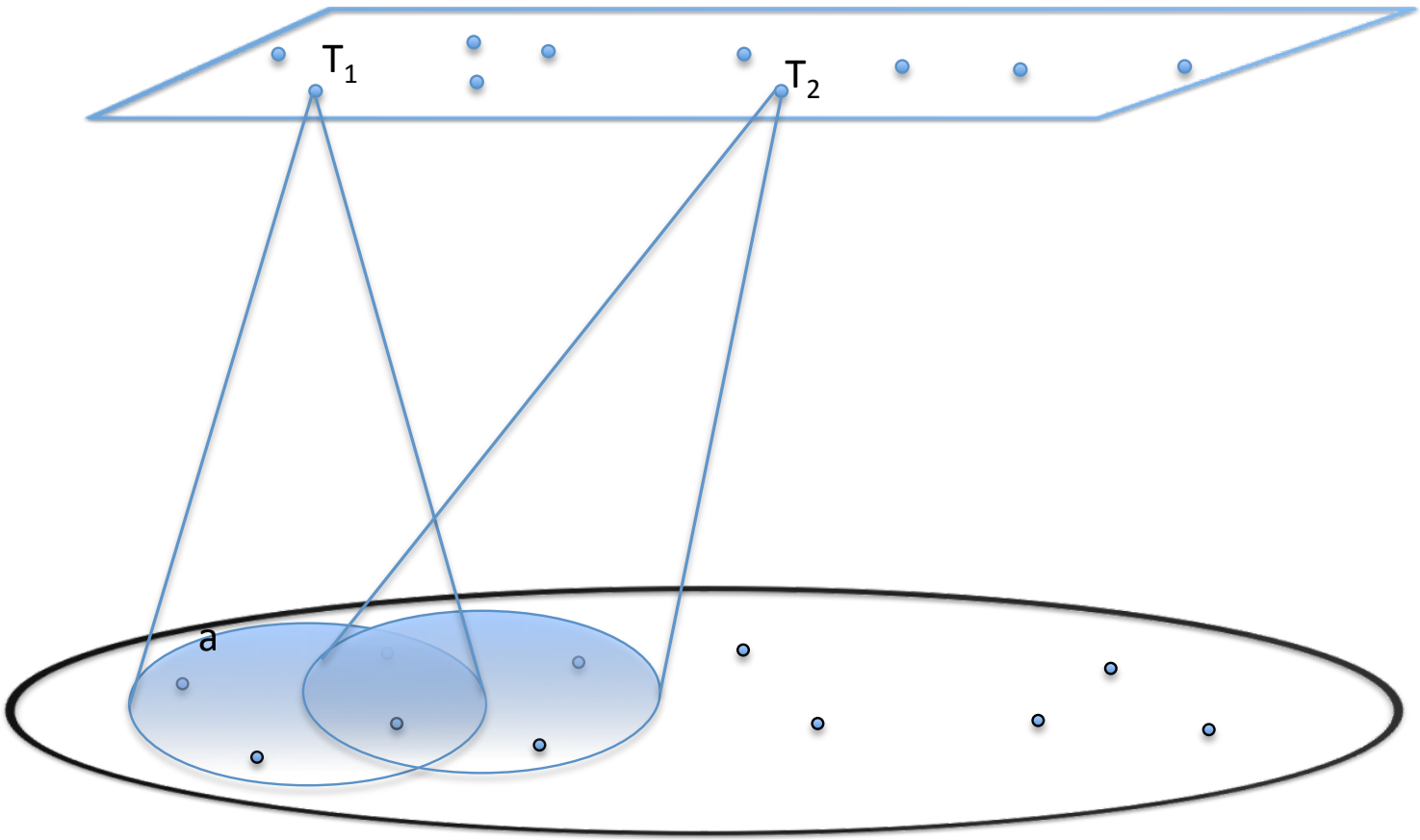
Intensionality

- 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*.

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*.
- in `pyttr`

$a : T_1$



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)

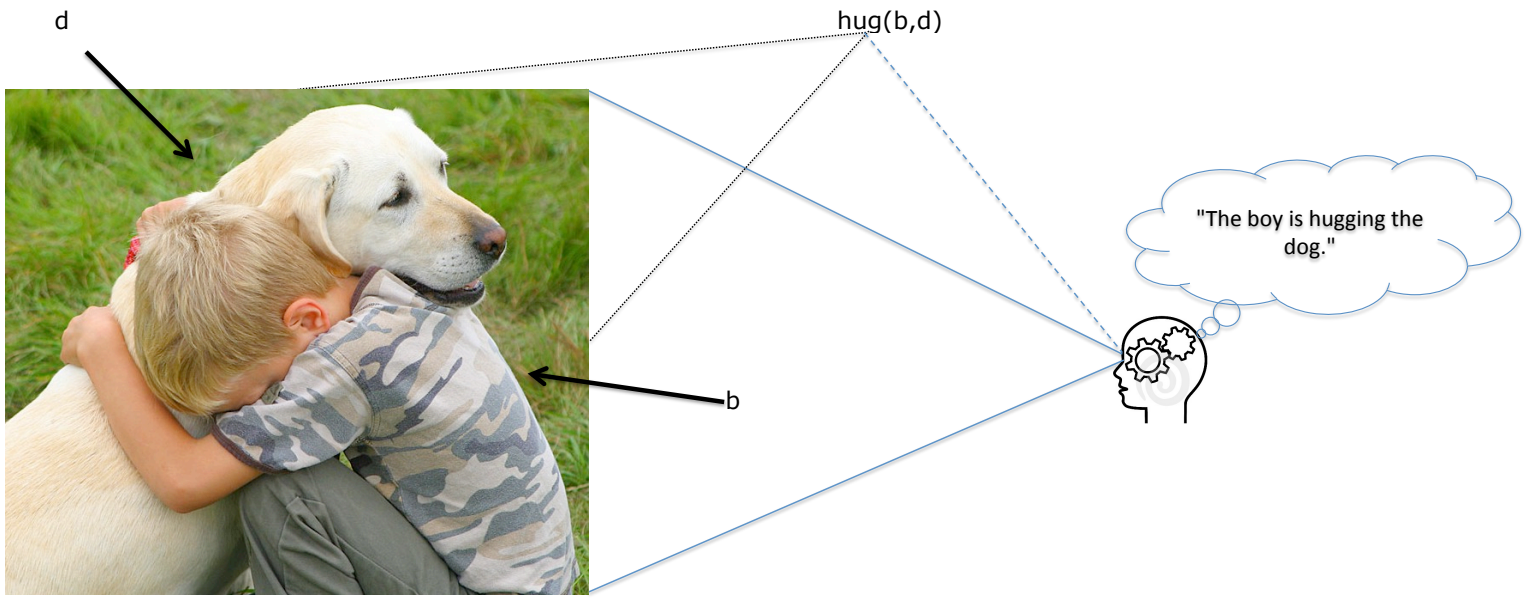
Why is structure important?

- 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:
  - $\text{Arity}(\text{run}) = \langle Ind \rangle$
  - $\text{Arity}(\text{hug}) = \langle Ind, Ind \rangle$

Predicates

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

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.
- in `pyttr`

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

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
- *possibilities* as different assignments to types (models)

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
- *possibilities* as different assignments to types (models)
- in `pyttr`

Witness conditions associated with ptypes

- in any situation, a is to the left of b iff b is to the right of a
- $s : \text{left}(a, b) \Leftrightarrow s : \text{right}(b, a)$
- in `pyttr`

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"

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

A boy hugs a dog

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

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

The official notation

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

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: $a : \mathit{Ind}$
$s_1 : \mathsf{boy}(a)$
$b : \mathit{Ind}$
$s_2 : \mathsf{dog}(b)$
$s_3 : \mathsf{hug}(a,b)$

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

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
- feature structures

Why are record types interesting for linguists?

They allow us to model

- discourse representation structures
- feature structures
- dialogue game boards (information states)

Why are record types interesting for linguists?

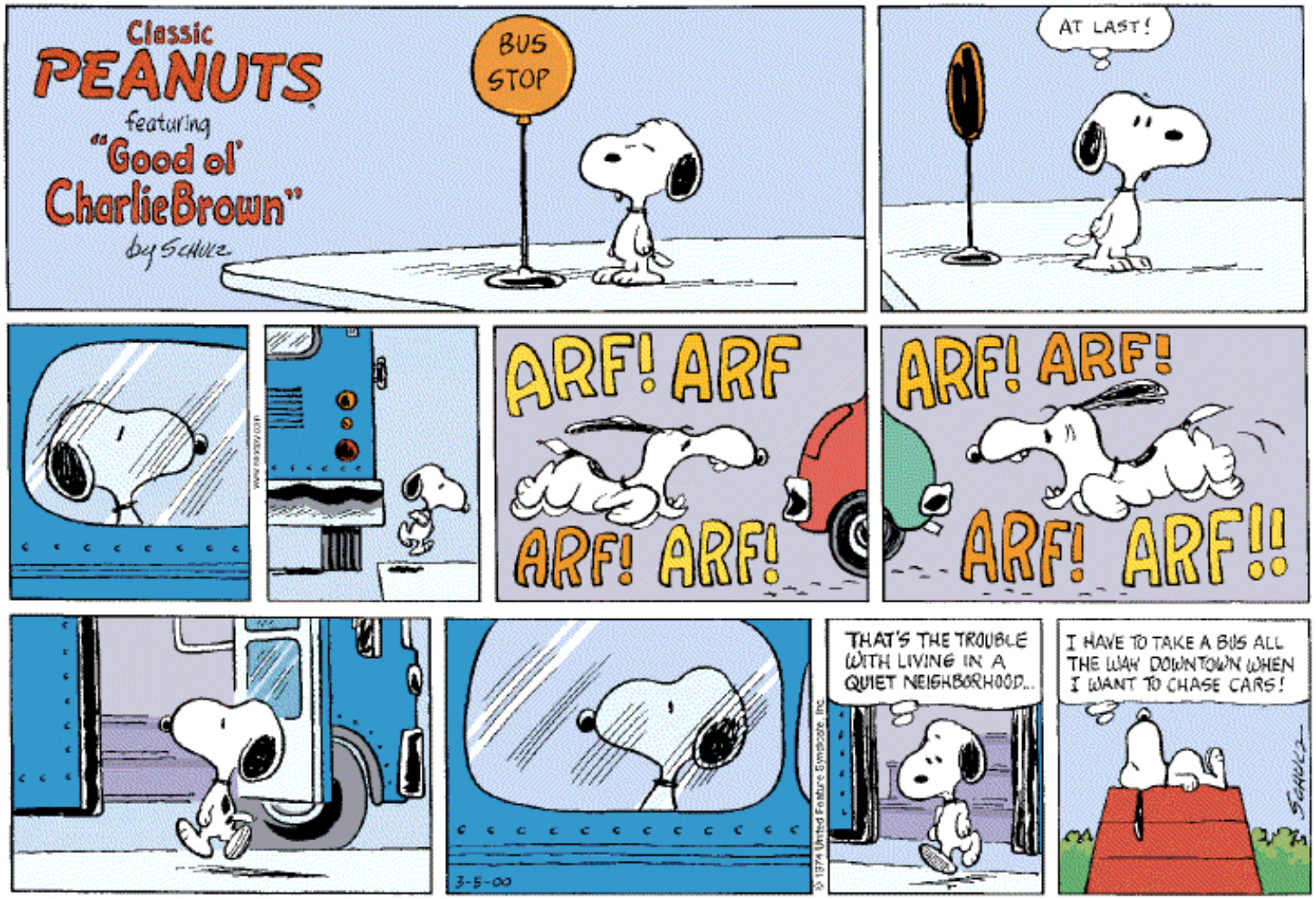
They allow us to model

- discourse representation structures
- feature structures
- dialogue game boards (information states)
- frames (as in frame semantics and FrameNet)

in `pyttr`

Outline

Fernando's string theory



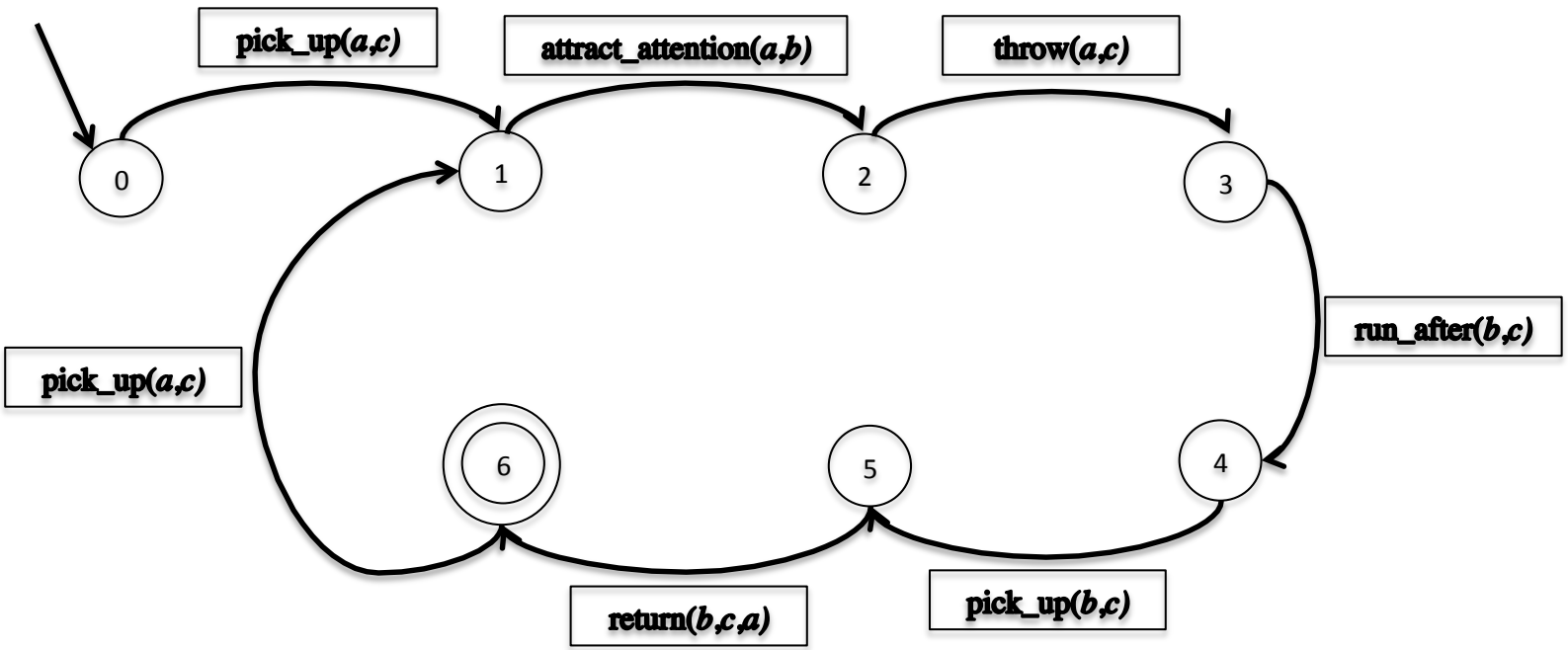
Some references to Fernando's work

Fernando (2004, 2006, 2008, 2009, 2015)

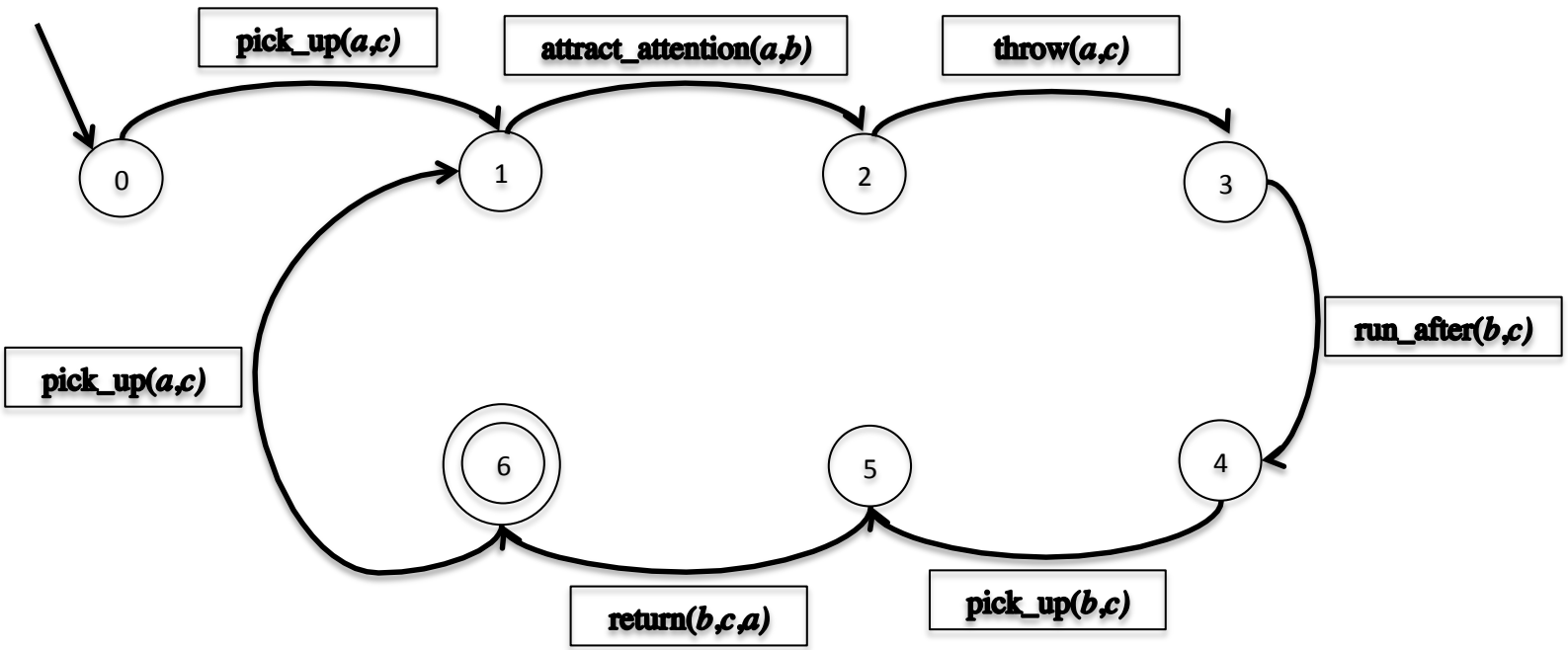
Regular types

1. if T_1, $T_2 \in$ **Type**, then $T_1 ^\frown T_2 \in$ **Type**
   $a : T_1 ^\frown T_2$ iff $a = x ^\frown y$, $x : T_1$ and $y : T_2$
2. if $T \in$ **Type** then $T^+ \in$ **Type**.
   $a : T^+$ iff $a = x_1 ^\frown \ldots ^\frown x_n$, $n > 0$ and for i, $1 \leq i \leq n$, $x_i : T$
   ...

A game of fetch



A game of fetch



$$(\text{pick\_up}(a,c)^\frown\text{attract\_attention}(a,b)^\frown\text{throw}(a,c)^\frown\text{run\_after}(b,c)^\frown \text{pick\_up}(b,c)^\frown\text{return}(b,c,a))^+$$

Outline

Partiality of event perception

- Do not need to observe all the frames in an event
- Suffices to observe enough to uniquely identify event types agent has in its resources

Inferring an event type from a partial observation

$$\lambda r{:}\left[\begin{array}{l} \mathsf{x}{:}\mathit{Ind} \\ \mathsf{c}_{\mathrm{human}}{:}\mathsf{human}(\mathsf{x}) \\ \mathsf{y}{:}\mathit{Ind} \\ \mathsf{c}_{\mathrm{dog}}{:}\mathsf{dog}(\mathsf{y}) \\ \mathsf{z}{:}\mathit{Ind} \\ \mathsf{c}_{\mathrm{stick}}{:}\mathsf{stick}(\mathsf{z}) \\ \mathsf{e}{:}\mathsf{pick\_up}(\mathsf{x},\mathsf{z})^\frown\mathsf{attract\_attention}(\mathsf{x},\mathsf{y}) \end{array}\right]\left(\left[\mathsf{e}{:}\mathsf{play\_fetch}(r.\mathsf{x},r.\mathsf{y})\right]\right)$$

Three views of this inference

Three views of this inference

- function from objects (events) to a *type*

Three views of this inference

- function from objects (events) to a *type* – $\lambda a : T_1(T_2[a])$

Three views of this inference

- function from objects (events) to a *type* – $\lambda a : T_1(T_2[a])$
- a *dependent type*

Three views of this inference

- function from objects (events) to a *type* – $\lambda a : T_1(T_2[a])$
- a *dependent type*
- perceiving something and inferring the type of something not (yet) perceived from that perception

Three views of this inference

- function from objects (events) to a *type* – $\lambda a : T_1(T_2[a])$
- a *dependent type*
- perceiving something and inferring the type of something not (yet) perceived from that perception
- we will see a number of other uses of dependent types, for example as the interpretation of verb phrases

in `pyttr`

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
  - string types
  - strings of events rather than strings of symbols

Bibliography I

References to work on TTR are available on
`https://sites.google.com/site/typetheorywithrecords`

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.

Bibliography II

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.

Cooper, Robin (2017) Adapting Type Theory with Records for Natural Language Semantics, in S. Chatzikyriakidis and Z. Luo (eds.), *Modern Perspectives in Type-Theoretical Semantics*, Studies in Linguistics and Philosophy *98*, pp. 71–94, Springer.

Cooper, Robin and Jonathan Ginzburg (2015) Type Theory with Records for Natural Language Semantics, in Lappin and Fox (2015), pp. 375–407.

Bibliography III

Fernando, Tim (2004) A finite-state approach to events in natural language semantics, *Journal of Logic and Computation*, Vol. 14, No. 1, pp. 79–92.

Fernando, Tim (2006) Situations as Strings, *Electronic Notes in Theoretical Computer Science*, Vol. 165, pp. 23–36.

Fernando, Tim (2008) Finite-state descriptions for temporal semantics, in H. Bunt and R. Muskens (eds.), *Computing Meaning, Volume 3* (*Studies in Linguistics and Philosophy* 83), pp. 347–368, Springer.

Fernando, Tim (2009) Situations in LTL as strings, *Information and Computation*, Vol. 207, No. 10, pp. 980–999.

Fernando, Tim (2015) The Semantics of Tense and Aspect: A Finite-State Perspective, in Lappin and Fox (2015).

Bibliography IV

Gibson, James J. (1986) *The Ecological Approach to Visual Perception*, Lawrence Erlbaum Associates, first published 1979.

Lappin, Shalom and Chris Fox, eds. (2015) *The Handbook of Contemporary Semantic Theory*, second edition, Wiley-Blackwell.

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.

Bibliography V

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