Journal of Logic, Language and Information (2019) 28:131–155
https://doi.org/10.1007/s10849-019-09285-4

Representing Types as Neural Events

Robin Cooper[1]

Published online: 14 March 2019

Abstract
One of the claims of Type Theory with Records is that it can be used to model types learned by agents in order to classify objects and events in the world, including speech events. That is, the types can be represented by patterns of neural activation in the brain. This claim would be empty if it turns out that the types are in principle impossible to represent on a finite network of neurons. We will discuss how to represent types in terms of neural events on a network and present a preliminary computational implementation that maps types to events on a network. The kind of networks we will use are closely related to the transparent neural networks discussed by Strannegård.

Keywords Type theory · Record types · Neural modelling · Compositionality · Linguistic recursion

1 Types and Cognition

Work on TTR, Type Theory with Records, Cooper and Ginzburg (2015) and Cooper (2017a, in prep) claims that it can be used to model types learned by agents in order to classify objects and events in the world. If this is true, types must be represented in some way in brains. In this paper we will make some preliminary suggestions concerning the nature of such representations. The question of neural representations of types arises in connection with the kind of type theory proposed by TTR in a way that it does not in connection with more traditional type theories. The reason is that TTR aims to provide the kind of types that agents use in the perception of objects and events and which they use in interaction to communicate with each other. If it were to turn out that the kind of types used are in principle impossible to represent on arrays of neurons then this would call this project into question. In this paper we explore in an abstract way how types could be represented by events in a network of neurons. We

✉ Robin Cooper
cooper@ling.gu.se

[1] Department of Philosophy, Linguistics and Theory of Science, Centre for Linguistic Theory and Studies in Probability, University of Gothenburg, Gothenburg, Sweden

cannot claim at this stage that this is a theory of what actually happens in the brain. But we do think that what we have suggests that it feasible to represent these types in a neural array.

In contrast to *traditional type theories* used in classical approaches to formal semantics (Montague 1973, 1974), TTR is a *rich type theory* inspired by work developing from Martin-Löf (1984), called "modern type theory" by Luo (2010, 2011). Traditional type theories provide types for basic ontological classes (e.g., for Montague: entities, truth values, time points, possible worlds and total functions between these objects) whereas rich type theories provide a more general collection of types, e.g. in our type theory, categories of objects such as *Tree*, types of situations such as *boy-hugs-dog*, that is, the type of situation where a boy hugs a dog.

Central to a rich type theory is the notion of *judgement* as in

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

in symbols, $a : T$. We say that a is a *witness* for T. We build on this notion to put a cognitive spin on type theory, and say that perception involves a judgement that an object (possibly an event or, more generally, a situation) belongs to a certain type. Perception is constrained by the types to which an agent is attuned. This relates to ideas about visual perception proposed by Gibson (1979) which were influential in the development of situation semantics (Barwise and Perry 1983).

If an agent perceives a particular object, a, as a tree it must be because they are attuned to a type, *Tree*, which can be thought of as an invariant (in Gibson's sense) across situations containing a tree or across objects which would be classified as a tree. In TTR and other similar type theories, types can be complex objects constructed from other objects in the theory. For example, in TTR, in order to construct the type of situation in which Kim hugs Fido we use a predicate 'hug' as a type constructor which requires two individuals (like 'kim' and 'fido') to return a type of situation represented as 'hug(kim,fido)'. In TTR we call this a *ptype* (a type constructed using a predicate). Note that while this is the notation which is used to represent a formula or proposition with a truth value in standard logic, for us it represents a type of situation. There is, however, a correspondence between this type and a proposition, following the "propositions as types" dictum which Martin-Löf took over from intuitionistic logic. The type can be said to be "true" just in case it has a witness, that is, there is a situation of the type and "false" otherwise, that is, if the type is empty (has no witnesses).

This gives us a way of talking about situations in which particular objects stand in a relation. However, we also want to have more general types such as *boy-hugs-dog* where the type is not constructed on the basis of particular individuals but just requires that *some* boy hugs *some dog*. In TTR we use *record types* for such a type. In (1) we show a record type corresponding to *a boy hugs a dog*.

$$(1) \quad \begin{bmatrix} x & : & Ind \\ c_{boy} & : & boy(x) \\ y & : & Ind \\ c_{dog} & : & dog(y) \\ e & : & hug(x,y) \end{bmatrix}$$

(1) is a graphic display of a set of ordered pairs (*fields*) each of which contains a *label* (such as 'x', 'c_{boy}') and a type (such as *Ind*, 'boy(x)'[1]) A record type must be the graph of a function, that is, the labels may only occur once.

Witnesses of record types are called *records*. An example of a record is given in (2).

$$(2)\quad \begin{bmatrix} x & = & sam \\ c_{boy} & = & s_1 \\ y & = & fido \\ c_{dog} & = & s_2 \\ e & = & s_3 \end{bmatrix}$$

As before (2) is a graphic display of a set of ordered pairs (fields). In the record the fields consist of a label followed by a witness for a type. The record (2) will be of the type (1) just in case the conditions in (3) hold.

(3) sam : *Ind*
s_1 : boy(sam)
fido : *Ind*
s_2 : dog(fido)
s_3 : hug(sam, fido)

The types in the fields with the labels 'c_{boy}', 'c_{dog}' and 'e' in (1) depend on the objects in 'x' and 'y'-fields in the record which is being judged as being of the type. Thus, for example, the relevant type for 'c_{boy}' with respect to the record in (2) is 'boy(sam)' as indicated in (3) since 'sam' is the value in the 'x'-field.

The representation of this kind of dependence illustrated in (1) is convenient for everyday use but it becomes inexact when we consider that record types may be embedded within larger record types and that the same label may occur at different levels in a record type. Consider that (4) is a well-formed record.

$$(4)\quad \begin{bmatrix} x & = & \begin{bmatrix} x & = & sam \\ c_{boy} & = & s_1 \end{bmatrix} \\ y & = & \begin{bmatrix} y & = & fido \\ c_{dog} & = & s_2 \end{bmatrix} \\ e & = & s_3 \end{bmatrix}$$

We might think that we could represent a type for this record as (5).

$$(5)\quad \begin{bmatrix} x & : & \begin{bmatrix} x & : & Ind \\ c_{boy} & : & boy(x) \end{bmatrix} \\ y & : & \begin{bmatrix} y & : & Ind \\ c_{dog} & : & dog(y) \end{bmatrix} \\ e & : & hug(x,y) \end{bmatrix}$$

[1] To call 'boy(x)' a type is a bit of a simplification, as we will see shortly.

But this leads to unclarity as to which 'x' and 'y' are being referred to in 'boy(x)', 'dog(y)' and 'hug(x,y)'. The strategy for dealing with this in TTR involves modelling *dependent fields* in record types (such as $\left[\mathrm{c_{boy}}:\mathrm{boy(x)}\right]$) as ordered pairs whose first member is a label and whose second member is an ordered pair of a dependent type and a tuple of paths showing where to find the arguments to the dependent type.

A *dependent type* is a function from objects of a given type to a type. (6) provides an example.

(6) $\lambda v{:}\mathit{Ind}\ .\ \mathrm{boy}(v)$

Thus a more explicit rendering of the type (1) is (7).

(7)
$$\left[\begin{array}{lcl} \mathrm{x} & : & \mathit{Ind} \\ \mathrm{c_{boy}} & : & \langle \lambda v{:}\mathit{Ind}\ .\ \mathrm{boy}(v), \langle \mathrm{x} \rangle \rangle \\ \mathrm{y} & : & \mathit{Ind} \\ \mathrm{c_{dog}} & : & \langle \lambda v{:}\mathit{Ind}\ .\ \mathrm{dog}(v), \langle \mathrm{y} \rangle \rangle \\ \mathrm{e} & : & \langle \lambda v_1{:}\mathit{Ind}\ .\ \lambda v_2{:}\mathit{Ind}\ .\ \mathrm{hug}(v_1,v_2), \langle \mathrm{x,y} \rangle \rangle \end{array}\right]$$

This is not the easiest representation to read but it does more nearly represent the structure of the type that we seek to represent as a neural event.

Dependent types, that is, functions that return types, have a number of uses in the theory of language which has been developed using TTR. One of these which we will talk about the neural representation of below is the treatment of generalized quantifiers as relations between dependent types. What we discuss here is a slightly simplified version of that discussed in Cooper (in prep) and elsewhere. Let us call functions of type (*Ind* → *Type*) *properties (of individuals)*. Examples of properties are given in (8).

(8) a. $\lambda v{:}\mathit{Ind}\ .\ \mathrm{dog}(v)$
b. $\lambda v{:}\mathit{Ind}\ .\ \mathrm{run}(v)$

We will use 'dog′' and 'run′' to represent (8a) and (8b) respectively. We can now introduce predicates corresponding to quantifier relations which hold between properties. Thus, for example, we can form the ptype in (9).

(9) $\mathrm{every}(\mathrm{dog}', \mathrm{run}')$

Intuitively this represents the type of situations in which every dog runs. We can relate this treatment of generalized quantifiers to classical generalized quantifier theory (Barwise and Cooper 1981). We use the notational conventions in (10).

(10) a. if T is a type, we use $[\check{}T]$ to represent $\{a \mid a : T\}$
b. We say that a type, T, is "true" just in case $[\check{}T] \neq \emptyset$ (*propositions as types*)
c. if P is a property, we use $[\downarrow P]$ to represent the *extension of property* P:

$\{a \mid [\check{}P(a)] \neq \emptyset\}$

We can now express familiar statements from generalized quantifier theory as, for example, (11)

(11) If P and Q are properties, every(P, Q) is true iff $[\downarrow P] \subseteq [\downarrow Q]$

2 Neuroscience Fiction

If we adopt the cognitive view of type theory discussed in the previous section, it seems that we should accept that the types can be represented by patterns of neural activation in the brain. The claim of the cognitive view would appear to be empty if it turns out that the types are in principle impossible to represent on a finite network of neurons. The aim of this paper is to suggest a way in which types could be represented neurally in principle rather than to make precise claims about the nature of their representation in agents' brains. Nevertheless it is our hope that the kind of techniques we are sketching are ultimately neurally plausible in the sense that they could be made consistent with what we know about biological brains.

The project we are engaged in here can be called *neuroscience fiction*. We cannot yet hope to observe brain activity corresponding to single types as conceived of in TTR. Available techniques such as FMRI do not have fine enough resolution and there is too much noise of other brain activity to easily identify exactly which neural activity corresponds to the perception of a situation where, for example, a boy hugs a dog. We see this work as an attempt to consider what a top-down approach to the neuroscience of perception and classification would be as opposed to the bottom-up approach which is available in current neuroscience. Basically the idea is this: in the bottom-up approach you might show a subject a picture of a boy hugging a dog and see what is common in brain activity over a large number of trials; on the top-down approach you create a theory which makes a prediction of brain activity corresponding to a boy hugging a dog and you then test the prediction in subjects shown a picture of a boy hugging a dog.

We agree with Pulvermüller (2002) that it is valuable to explore neural possibilities and disagree with Chomsky (2000) who argues that the relation of linguistic analysis to neuroscience should not be pursued because relevant elements and principles of brain structure may yet have to be discovered. As in physics, theoretical predictions may be made which can be empirically tested subsequently when experimental techniques become available and the theoretical predictions may serve to guide experimental work.

It will be central to our discussion here that what is involved in representing types neurally is a neural event rather than a piece of neural architecture. We will present a preliminary Python implementation (see `nu.ipynb` on https://github.com/GU-CLASP/pyttr) that maps a subset of the TTR types characterized in the TTR system `pyttr` to types of events on a network.[2] The types of neural events are also charac-

[2] The TTR types covered in the current implementation are: basic types, ptypes, dependent types (construed as usual in TTR as functions), meet and join types, and record types. The TTR types are not modified in any way to accommodate the mapping to types of neural events.

terized as TTR types. The program implements a mapping from TTR types to (TTR) types of neural events. As such it can be seen as a hand-crafted compiler. The fact that it deals with neural structures might create the incorrect impression that some kind of machine learning is involved involving techniques like the adjustment of weights on connections between neurons such as back propagation. Here we are concerned only with the problem of how types might be represented on a network of neurons. The issue of how types can be learned from some kind of input is a separate problem. (For some discussion see Matsson 2018).

The kind of networks we will use are closely related to the transparent neural networks (TNN) discussed by Strannegård and Nizamani (2016). Transparent networks differ from standard neural networks in that the architecture and events on the network are associated with interpretations. In standard neural networks interpretations are associated with the input layer and the output layer but it is unclear how what happens between these two layers should be interpreted. Transparent nets on the other hand are constructed in a way so that all layers are interpretable. Patterns of activation on collections of neurons are given an interpretation in terms of the world external to the network. In this paper, the external interpretation will be in terms of the type theoretic objects of TTR.

It may be helpful to emphasize some of the things we are *not* doing with this particular implementation: we are not engaging in a machine learning exercise but rather addressing the theoretical question of how types could be represented in a neurologically plausible network; we are not addressing the question of recognizing witnesses for types but just the representation of the types themselves. The question of learning to make judgements that certain situations in the world are of these types *is* something where models of machine learning might be helpful and we will have some suggestions for how this could be approached later. For some relevant suggestions see Larsson (2013).

We will make some basic assumptions about neurons which seem to correspond to basic facts about typical neurons. A neuron consists of a body which carries out a computation on the basis of several inputs received on a number of dendrites connected to the body. A neuron has a single axon on which signals can be sent on the basis of the computation performed on the input received on the neuron's dendrites. While the neuron only has a single axon this axon may be connected to a large number of dendrites of other neurons by means of a number of axon terminals branching from the axon. The connection between an axon terminal and a dendrite is known as a synapse and the synapse itself may have some computational power. The input on a dendrite can correspond to a real number whereas the output on an axon (based on a computation of dendritic input) is boolean: either the neuron fires or it does not. We can think of the computation carried out by a synapse as converting a boolean to a real number. A simplified representation of a neural state is a characterization of which neurons have active axons, that is, which neurons have an output of 1. For a neuroscientist this description of what is going on in the brain may seem like oversimplification to the point of falsity. However, it will enable us to address some of the basic formal problems associated with representing types as neural activation.

3 The Binding Problem

Consider a TTR ptype like 'hug(a, b)'. Intuitively it is a type of situation in which a hugs b. The binding problem refers to making sure that one can distinguish between the type of events where a hugs b, the ptype 'hug(a, b)' in TTR, and the type of events where b hugs a, 'hug(b, a)' (Shastri 1999; Kiela 2011). The binding problem is, of course, more general than finding a way of representing TTR types, or even just neural representations of semantic content, as Jackendoff (2002, p. 58ff) points out. A minimal solution is to designate an event involving the activation of a single neuron to represent each of 'hug', a and b. A more realistic encoding would most likely be events involving several neurons for each of these but the activation of a single neuron will be sufficient for the purposes of this discussion. It would not be a solution, for example, to take a simple "architectural" approach where we say that 'hug(a,b)' is represented by the simultaneous firing of these three neurons. That would mean that 'hug(b,a)' would have the same neural representation. An initial proposal for the neural representation of the ptype might rather be a neural event in which each of the neural events associated with the predicate and the arguments occurs in turn. In the neural TTR implementation this is displayed as the history of activation on a network as in (12).

(12)

a	0	0	1	0	0
b	0	0	0	1	0
hug_n	0	1	0	0	0

Here each row indicates a neuron given a convenient label in the first column. (The label `hug_n` indicates the neuron corresponding to the predicate 'hug'.) The labels are there for convenience in the display so that we can have some idea of what is going on. We have no particular theory of what it means to say that a neuron is labelled, for example, by `a` or `hug_n`. We might think of it as indicating the neuron that is active just in case the agent thinks of individual a or a hugging event but we have at present nothing substantial to say about how this association of neural activity with aspects of the external world comes about. Subsequent columns indicate whether the neuron is firing at successive time-steps. However, while it may be possible to make the relevant distinctions in simple examples simply by requiring that neurons fire in a certain order, this would not give us what we need to represent more complex structures and we do not wish to rely on neurons' order of firing but rather on the phasing of neurons with other neurons in a way similar to that originally suggested by Shastri and Ajjanagadde (1993) and Shastri (1999).

This means that we add neurons that will correspond to predicate and argument roles and also a neuron that will be active throughout the neural event coding that the three separate neural events group together. The phasing in the output of our implementation may give the impression of being rather different to that proposed by Shastri and Ajjanagadde since our program produces a representation of one cycle of neural activity, and thus many neurons only fire once as opposed to the repetitive spiking represented in many of Shastri and Ajjanagadde's diagrams. However, this is only a matter of what is being represented in our output. The idea is that a brain

representing 'hug(a, b)' would cycle repeatedly through the patterns we represent until attention is shifted to something else in the way that Shastri and Ajjanagadde suggest in their discussion of phasing. A possible difference, though, is that we envision the possibility as clearly marking the end of a cycle by shutting down all the relevant neurons for one time step before progressing to the next cycle. The pattern of activity on the network is given in (13).

(13)

a	0	0	1	0	0
b	0	0	0	1	0
hug_n	0	1	0	0	0
ptype2	*	1	1	1	0
rel	*	1	0	0	0
arg0	*	0	1	0	0
arg1	*	0	0	1	0

Here the activation of the neuron labelled 'ptype2' encodes that a two-argument ptype is represented from time-steps 2–4 with the relation at time-step two and the two arguments at the subsequent time-steps. We use 'rel' as the label neuron for relations, 'arg0' for first arguments and 'arg1' for second arguments. This assumes that the notions of first and second argument are the relevant notion (that is, these neurons could be used to label arguments to any relation) as opposed to theta-roles like *agent* and *theme* or relation specific roles such as *hugger* and *hugee*. In terms of what is presented in this paper nothing depends on this choice. The fact that we are labelling the various roles in the ptype means that while we are exploiting the fact that the ptype is encoded as an event over several time-steps in order to solve the binding problem, it is no longer important exactly which order the events occur in. The 4th–7th rows in this display correspond to what we might call "book-keeping" neurons which are used to indicate the structure of represented types, as opposed to the "content" neurons represented in the first three rows. Just as we do not have anything to say about how content neurons get their content, we do not have anything to say about how book-keeping neurons get their labelling function.

Note that the final column in this display, where all the neurons are inactive, is important. It is important that the neuron 'ptype2' ceases to be active at the end of the event representing the ptype and that none of the events which are part of the representation of the ptype continue after that neuron has stopped being active. If this were not the case we would not have successfully delineated the boundaries of the neural event representing the ptype.

If the system discovers during the course of a computation that not enough book-keeping neurons are available it will create those needed in order to carry out the computation. In the implementation this represents an expansion of the number of neurons in the network and it is indicated in this display by the occurrences of '*' in the first column indicating that these neurons did not exist at the first time-step. While neurogenesis (structural plasticity) is a known phenomenon—see Maguire et al. (2000, 2006) for a discussion of the relative sizes of the hippocampus in London taxi drivers as compared with London bus drivers—it does not seem reasonable to assume that human brains actually grow during the course of a computation in this way, but we might take

this expansion of the network to model a use of previously unused neurons in order to carry out a novel computation. Note that this recruitment of neurons on the fly could be quite efficient since it need not involve creating either new neurons or new connections between neurons. Note also that while we are duplicating book-keeping neurons we are not duplicating content neurons. Furthermore the book-keeping neurons are reused when representing new types. The only case in which new neurons are recruited is if there are not enough book-keeping neurons available for a particular type.

From this simple example, three potential basic principles of neural representation emerge:

- *neural events* (with phasing) are important for neural representation (rather than just neural architecture or snapshots of the network at a single time-step)
- *neural event types* can be *realized differently* on different networks, cf. Fedorenko and Kanwisher (2009) and Fedorenko and Kanwisher (2011). Which neurons are dedicated to a particular purpose can vary from network to network and depends in part on the order in which things are presented to the network.
- We can expect a kind of *compositionality* in neural representations. For example, whatever pattern of activation a network uses to represent 'hug' (firing of a single neuron or multiple neurons), that pattern of activation will occur in phase with a 'rel' pattern of activation in representing a ptype with 'hug'.

Note that the neural events we are talking about do not necessarily have architectural implications, for example, that the neurons involved in the event are connected. There could be factors external to the configuration (even non-neural factors such as perception) which cause neurons to fire together even though they are not connected. It is true that neurons which tend to fire together become wired together (Hebbian learning). But note that in the case of predicates and their arguments we do not normally want this to happen since predicates can occur with different arguments and the objects which are the arguments can occur with different predicates. So for example we do not in general want an agent to think about a hugging b whenever she thinks about hugging or to think about hugging whenever she thinks about a.

4 The Recursion Problem

As a simple illustration of the kind of recursion needed by linguistic representations we will show examples where ptypes can occur as arguments within ptypes as in (14).

(14) a. believe(c, hug(a,b))
b. know(d, believe(c, hug(a,b)))

One aspect of this recursion that can be challenging for neural representation is that there is in princple no upper limit on the depth of embedding that can be obtained.

Another challenge is that various components may be repeated at various points in the structure as in (15)

(15) believe(a, believe(b, hug(a,b)))

Thus the phasing of neurological events in a representation of this has to be such that a single object can play several distinct roles in the representation. This is an instance of what Jackendoff (2002, p. 61), refers to as *The Problem of 2*.

The technique we developed for coding ptypes as neurological events in order to solve the binding problem is in fact adequate to deal with the recursion problem as well. Example (16) is a trace of a network event representing believe (c, hug(a,b)).

(16)

a	0	0	0	0	1	0	0	0
b	0	0	0	0	0	1	0	0
hug_n	0	0	0	1	0	0	0	0
ptype2	0	1	1	1	1	1	1	0
rel	0	1	0	0	0	0	0	0
arg0	0	0	1	0	0	0	0	0
arg1	0	0	0	1	1	1	1	0
c	0	0	1	0	0	0	0	0
believe_n	0	1	0	0	0	0	0	0
ptype2	*	0	0	1	1	1	0	0
rel	*	0	0	1	0	0	0	0
arg0	*	0	0	0	1	0	0	0
arg1	*	0	0	0	0	1	0	0

This is the first time that this network has seen an embedding of a ptype within a ptype and it therefore adds an additional set of book-keeping neurons for a two-place ptype. Note that the ptype2 neuron represented in row 4 is active from time-step 2 to time-step 7 whereas the ptype2 neuron in row 10 is active from time-step 4 to time-step 6 within the period of activation of the arg1 neuron represented in row 7. Note also that we are creating extra book-keeping neurons but not extra copies of content neurons. If a content neuron is used twice it fires in synchrony with two distinct book-keeping neurons firing at different times.

What we have here is thus a rather straightforward encoding of structure in a two-dimensional binary matrix. Given that the network is capable of growing in order to accommodate greater depths of embedding there is in principle no limit on the depth of embedding that it can handle except for (in the case of the implementation) available memory in the computer or (in the case of a natural brain) availability of neurons that can be dedicated to book-keeping. This is in contrast to the kind of neural network representation of recursion provided by, for example, Christiansen and Chater (1999) which is limited to a finite number of embeddings. On the other hand we have only looked at representation and said nothing about learning. This makes it difficult to make any meaningful comparison with the literature on neural networks at this point. Pulvermüller (2002) also discusses recursion as something that distinguishes the kind of abstract grammars developed by linguists and his neuronal grammars which are

not able to model recursion directly. SHRUTI as described by Shastri (1999) also has a finite limited architecture, introducing, for example, a limited number set by a parameter of copies of neural representation of relations corresponding to verbs in order to deal with multiple facts where the relation has different arguments. In this connection our proposal is more like the blackboard architecture introduced by van der Velde and de Kamps (2006) where the blackboard may represent different bindings at different times. Where our proposal differs from all of these proposals, however, is that we take a dynamic view of neural architecture where unused neurons can be dedicated to a particular task on demand as required by the complexity of what needs to be processed. We use this, however, for the bookkeeping neurons and do not duplicate neural architecture associated with relations or their arguments. It is this that captures the unbounded nature (up to limits arising from the size of the neural network) of recursion and reuse of elements in a structure.

The importance of recursion and the compositional approach to neural representation is further illustrated by the treatment of dependent types as functions which return a type, for example a ptype. Such functions can be of arbitrary depth (e.g. functions which return a function which returns a type and so on). Also we treat generalized quantifiers in terms of ptypes whose arguments are dependent types. Thus we can have a situation where we have a ptype within which is a function and within the function is a ptype construction. This is the kind of recursion which is common in linguistic structure. To illustrate how this works consider how we can create a dependent type which returns a ptype in `pyttr`, illustrated by the code in (17).

(17)
```
T = DepType('v',Ind,
                PType(hug,['v','b']))
print(show(T))
```

This returns (18).

(18)
```
lambda v:Ind . hug(v, b)
```

Thus we have created a function from objects of type *Ind* (individual) to the ptype of situations where that individual hugs *b*. A neural event which represents this function has a neural event representing a ptype ('ptype2') temporally included in a neural event representing a function ('lambda') as shown in (19).

(19)

b	0	0	0	0	1	0	0
hug_n	0	0	1	0	0	0	0
ptype2	0	0	1	1	1	0	0
rel	0	0	1	0	0	0	0
arg0	0	0	0	1	0	0	0
arg1	0	0	0	0	1	0	0
Ind_n	0	1	0	0	0	0	0
lambda	*	1	1	1	1	1	0
dom	*	1	0	0	0	0	0
var	*	1	0	1	0	0	0
rng	*	0	1	1	1	1	0

The function is represented over the time span during which the neuron labelled 'lambda' is active. It has two components: the domain (labelled 'dom') which is represented in one time-step and the range (labelled 'rng') which is represented over the remaining time-steps of the time span. The domain introduces a variable (indicated by the label 'var') which is typed (indicated by the neuron labelled 'Ind_n'). Thus in addition to the neuron 'lambda' three neurons are active in the domain component of the function: 'dom', 'var' and 'Ind_n'. The range uses the representation for ptypes, but notice that 'var' is active in synchrony with 'arg0' indicating that the first argument of the ptype is the variable from the domain of the function. Notice that this is a representation of the structured TTR object which is a function. It does not, for example, represent the graph of the function (a set of ordered pairs as in the von Neumann set theoretic reconstruction of a function) or a procedure that would compute the value of the function for a given argument. One might well think that the representation of functions in the brain would be something more abstract or procedural than this representation of a structured object resembling rather closely the syntax of a logical expression representing a function. Nevertheless, we present this in order to show at least a way of presenting the kind of arbitrarily recursive structure that is needed to represent functions.

We can represent the type of situation in which every dog runs as the ptype (20).

(20)
```
every(lambda x:Ind . dog(x),
      lambda x:Ind . run(x))
```

This type will be correspond to an neural event as illustrated in (21).

(21)

every_n	0	1	0	0	0	0	0	0	0	0	0	0	0
dog_n	0	0	0	1	0	0	0	0	0	0	0	0	0
run_n	0	0	0	0	0	0	0	0	1	0	0	0	0
Ind_n	0	0	1	0	0	0	0	1	0	0	0	0	0
ptype2	*	1	1	1	1	1	1	1	1	1	1	1	0
rel	*	1	0	0	0	0	0	0	0	0	0	0	0
arg0	*	0	1	1	1	1	1	0	0	0	0	0	0
arg1	*	0	0	0	0	0	0	1	1	1	1	1	0
lambda	*	0	1	1	1	1	0	1	1	1	1	0	0
dom	*	0	1	0	0	0	0	1	0	0	0	0	0
var	*	0	1	0	1	0	0	1	0	1	0	0	0
rng	*	0	0	1	1	1	0	0	1	1	1	0	0
ptype1	*	0	0	1	1	0	0	0	1	1	0	0	0
rel	*	0	0	1	0	0	0	0	1	0	0	0	0
arg0	*	0	0	0	1	0	0	0	0	1	0	0	0

Note that as in the reuse of the logical variable 'x' in (20) in different binding domains so the 'var' neuron can be used separately in the two functions and we do not need to create an additional 'var' neuron. The scope of the variable is delineated by the activity of the 'lambda' neuron, which is also reused for the second function. We

would only have needed to create a new 'lambda' neuron if the second function had been embedded in the first. Note also that it is important that the 'lambda' neuron be inactive for at least one time-step between the two functions—otherwise the neural event would be indicating that only one function is represented.

Finally in this section we show how to represent a simple dependent record type using these techniques. The record type we are representing is that in (22).

(22) $\begin{bmatrix} \text{l\_x} & : & \textit{Ind} \\ \text{l\_e} & : & \text{dog(l\_x)} \end{bmatrix}$

(We use the labels 'l_x' and 'l_e' rather than 'x' and 'e' in the computational representation to make it easier to spot which symbols are being used to represent labels.) What is represented in the neural implementation of (22) is the version with the more explicit representation of the dependent field in (23).

(23) $\begin{bmatrix} \text{l\_x} & : & \textit{Ind} \\ \text{l\_e} & : & \langle \lambda v{:}\textit{Ind} \,.\, \text{dog}(v), \langle \text{l\_x} \rangle \rangle \end{bmatrix}$

The trace of neural activation corresponding to this type is given in (24).

(24)

l_x	0	0	0	0	0	0	0	1	0	0	1	0	0	0
l_e	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Ind_n	0	0	1	0	0	0	0	0	0	0	0	1	0	0
dog_n	0	0	0	1	0	0	0	0	0	0	0	0	0	0
field	0	1	1	1	1	1	1	1	1	0	0	0	0	0
label	0	1	0	0	0	0	0	0	0	0	0	0	0	0
value	0	0	1	1	1	1	1	1	1	0	0	0	0	0
field	0	0	0	0	0	0	0	0	0	0	1	1	0	0
label	0	0	0	0	0	0	0	0	0	0	1	0	0	0
value	0	0	0	0	0	0	0	0	0	0	0	1	0	0
rec	0	1	1	1	1	1	1	1	1	1	1	1	1	0
lambda	0	0	1	1	1	1	0	0	0	0	0	0	0	0
dom	0	0	1	0	0	0	0	0	0	0	0	0	0	0
var	0	0	1	0	1	0	0	0	0	0	0	0	0	0
rng	0	0	0	1	1	1	0	0	0	0	0	0	0	0
ptype1	0	0	0	1	1	0	0	0	0	0	0	0	0	0
rel	0	0	0	1	0	0	0	0	0	0	0	0	0	0
arg0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

As before the rows represent the activity pattern of a single neuron over a number of time steps. We can see that this record type is represented over a period of twelve time steps as shown by the period of continuous activity of the neuron labelled by `rec` in the leftmost column. The record type contains two fields as is indicated by two neurons labelled by `field`. One field is represented from time step 2 to time step 9 and the other between time steps 11 and 13. Notice that the order of the fields in the

representation is not significant (records and record types are modelled as *sets* of fields in the theory) and in (24) they have been ordered in the reverse order to that given in (23). In the same way the order of the elements in a field (the label and the value) is not important because they are identified respectively by activity of the `label` and `value` fields.

Just looking at this example, one may wonder why we have chosen to have two neurons representing a field, rather than just one which is activated twice. The reason is that records and record types are in general recursive structures. Thus a record type with its own fields may occur within a field of a superordinate record type. The depth of embedding, while it will always be finite within a single structure, can be arbitrarily deep. Thus we use the technique which we associated with recursion here. Each field consists of a label and a value and the period of activation of a `field`-neuron is divided between a period of activation for a `label`-neuron and a `value`-neuron. There are separate neurons to represent the labels 'l_x' and 'l_e'. Notice the 'l_x' is used twice in this record type, once as the label for a field and once as a reference to that field in the dependent field labelled 'l_e'. These two different uses are represented by the neuron 'l_x' being activated in the neural representation. Only one of those activations is accompanied by a simultaneous activation of a `label`-neuron indicating that this represents the occurrence of 'l_x' as the label in a field.

5 Types and Psychological Reality

It may seem that the kind of neural representation that we have proposed in this paper is overly "syntactic" in that it is essentially constructing a binary representation of the structure of the objects in our theory closely following the way they are represented on the page which in turn corresponds to the structure of the set theoretic objects we use to model them. However, if we are serious about proposing structured mathematical objects to represent types and also serious about those types having some kind of psychological reality in that agents with brains can use them not only to classify objects and events in the world but also to reason about the types themselves as, for example, in our treatment of predicates like *believe*, then it seems that this structure should be reflected in the neural representation. It seems also that something like this structure is necessary if we are to deal with recursion, which is, of course, involved in the treatment of intensional predicates like *believe* which seem to involve reasoning about types. If this kind of neural representation should turn out to be incorrect then we should seriously consider revising the type theory we use so that it has a different structure or finding a new way of making the mapping between types and neural events. Note that in the general case the types of TTR are not limited to the types to which linguistic agents are attuned. Thus while what we have been discussing in this paper relates to what is needed for natural language we want to say that the type system developed in linguistic agents has evolved from type systems developed by

non-linguistic organisms. While the types needed for non-linguistic agents may not be the same we would still want to see them encompassed by the general theory of TTR and have a way of neurally encoding them in the brains of those non-linguistic agents.

Nevertheless, in a way, what we are doing could be compared with a Fodorean language of thought (Fodor 1976, 2008). While nobody believes that there are syntactic expressions of a language of thought present in the brain one could imagine that some system of neural representation of such a language similar to what we have suggested for TTR types is. For many reasons, I think the kind of type based approach we are using here is preferable to the language of thought hypothesis, but this is not the place to discuss this. However, any neural representation relating to our conceptual abilities and the semantics of natural language must tackle the issues of binding and recursion that we have discussed here and the general points we have been making could be applied *mutatis mutandis* to other theories of cognitive structure.

The kind of system of neural representation we are suggesting could be used to make precise predictions about the complexity of concepts and propositions modelled as types. For example, according to the conversion for types that we have implemented we can say that the type in (23) needs a network with at least 18 neurons and at least 12 time steps are needed to represent it. We say "at least" here because it seems likely that the activation of single neurons here might correspond to patterns of activation of several neurons and this would lead to more neurons and more time steps involved in the representation of the type. We can, however, think of the theory as presented as placing a lower bound on space (number of neurons involved) and time required for the representation.

There is, however, an important issue to be tackled concerning the representation of types as neural events. This we take up in the next section.

6 Memory: A Simple Kind of Learning

We have argued above that a good way to deal with the binding problem and the recursion problem in representing types in neural networks is to let the representations be neural events rather than pieces of neural architecture. The simple-minded idea is that when an agent classifies an object or event as being of a certain type the corresponding neural event will occur in the agent's brain. While there seem to be good reasons to think of the representations as events rather than architecture, it seems initially puzzling how such an agent could store a type in memory. In the TTR literature we talk of agents as having types available as resources which can be used to make judgements about objects and situations. In particular Cooper et al. (2015) talk about estimating probabilistic judgements based on previous judgements. How could such judgements be stored in memory if they are just represented as neural events? We would not want to suggest, for example, that having a type in memory means that the neural event corresponding to the type is constantly being rehearsed in the brain. This makes it seem like we need something architectural in the brain corresponding to types after all.

Our proposed solution uses an idea from transparent neural networks (TNN) where a single neuron which is *top-active* in the sense of TNN (Strannegård and Nizamani 2016) can be regarded as encoding a concept since it is triggered by a complex activity corresponding to that concept. A node, n, is top active at a given timepoint if its excitation has been triggered by the excitation of other nodes in the network but n itself does not trigger activity in any other nodes. In this way, the activation of n can be seen as representing a complex concept characterized by the activation which triggered it. Here we will turn the idea around and create a single *memory* neuron which when excited will trigger a neural event representing a type. This is a simple way of "freezing" a neural event in a network in architectural terms. The memory neuron must be connected to other neurons in the network in a way so that its activation will occasion an orderly progression of neural events in sequence with the correct phasing. This is achieved by introducing *delay* neurons (Strannegård et al. 2015) which can be used to delay passing on a signal an arbitrary number of time-steps. Pulvermüller (2002) uses delay in neural circuitry in a similar way to account for the processing of linguistic strings. For an interesting account of delay circuitry in nature see Schöneich et al. (2015). As an illustration (25) shows the trace of a network with a memory neuron (labelled `every dog runs` in the display) for the type in (20).

Delay neurons, like other bookkeeping neurons, are added as required in the process of creating the memory. Notice that our treatment of quantification in terms of generalized quantifiers where the *every* is a predicate holding between two properties means that we can reuse our method for encoding ptypes in this more complex example involving quantification. Currently memories of judgements are implemented by activating a neuron represented by an object in phase with the representation of a type, though we suspect that something more like the method used for ptypes will ultimately be necessary. It should, however, be noted that representing the memory of a judgement, which can be thought of as representing an Austinian proposition in terms of TTR, is not the same as making the judgement itself which might involve a rather different pattern of activation. (26) shows an example where a particular event 'e' is judged to have the type in (20).

(25)

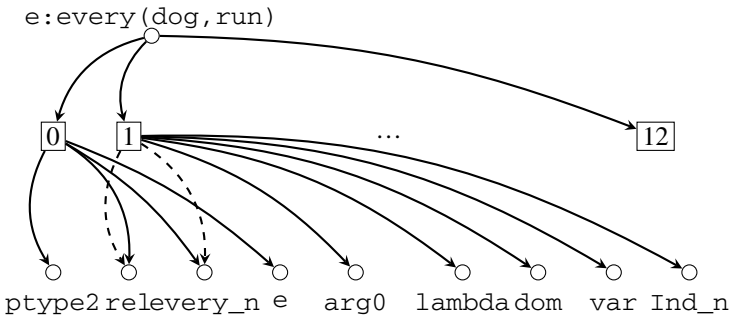
every_n	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
dog_n	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
run_n	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
Ind_n	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0
ptype2	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0
rel	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
arg0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0
arg1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0
lambda	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0
dom	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0
var	0	0	0	0	1	0	1	0	0	1	0	1	0	0	0
rng	0	0	0	0	0	1	1	1	0	0	1	1	1	0	0
ptype1	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0
rel	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0
arg0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0
every dog runs	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0
Delay	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0
Delay	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0
Delay	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0
Delay	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0
Delay	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0
Delay	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0
Delay	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0
Delay	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0
Delay	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0
Delay	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
Delay	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
Delay	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

(26)

every_n	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
dog_n	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
run_n	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
Ind_n	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0
e	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0
ptype2	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0
rel	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
arg0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0
arg1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0
lambda	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0
dom	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0
var	0	0	0	0	1	0	1	0	0	1	0	1	0	0	0	0
rng	0	0	0	0	0	1	1	1	0	0	1	1	1	0	0	0
ptype1	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0
rel	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0
arg0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0
e:every(dog,run)	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
Delay	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0
Delay	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0
Delay	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0
Delay	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0
Delay	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0
Delay	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0
Delay	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0
Delay	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0
Delay	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0
Delay	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0
Delay	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
Delay	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
Delay	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

A memory neuron, like 'e:every(dog,run)' in (26), is given an excitatory synaptic link to a number of delay neurons, one for each time step in the neural event that is to be engendered by exciting the memory neuron. In the case of (26) therefore there are 13 delay neurons. Each successive delay neuron is associated with an increased delay. Thus the first delay neuron will fire immediately it is excited, the second after one time step, the third after two time steps, and so on. Each of the delay neurons has excitatory synaptic links to the neurons which should be turned on at the corresponding time step and inhibitory synaptic links to those neurons which should be deactivated at that time step. Memorizing a type involves creating the memory neuron and the required delay neurons and creating the required synaptic links so that exciting the memory neuron will give rise to the neural event corresponding to the type. In (27) we give a partial representation of the network that achieves (26) representing just that part of the network that will achieve the first two time steps of the neural event corresponding to the type.

(27)



In (27) neurons are represented as small circles except for delay neurons which are represented as rectangles containing a number indicating the number of time steps delay between excitation and firing. Excitatory links between neurons are represented by solid arrows and inhibitory connections by dashed arrows. Thus in the first time step after the excitation of the memory neuron '`e:every(dog,run)`' the neurons '`ptype2`', '`rel`', '`every_n`' and 'e' are activated. In the second step the neurons '`rel`' and '`every_n`' are deactivated. The neurons '`ptype2`' and '`e`' will remain active and in addition the neurons '`arg0`', '`lambda`', '`dom`', '`var`' and '`Ind_n`' will be activated. This corresponds to the pattern of activation in (26).

What is the likelihood of such memory structures being encoded in natural brains? We know that delay circuitry exists in nature (Schöneich et al. 2015). Recent work suggests memories are transferred from the hippocampus to the neocortex during sleep (Draguhn 2018). It has been known since the 1970's that the firing of single neurons in the hippocampus, known as "place cells", can have referential significance, being activated when the animal enters a certain space in its evironment (O'Keefe and Dostrovsky 1971) [cited in Draguhn (2018) and Moser et al. (2017)]. Similarly there are "grid cells" corresponding to spatial coordinates (Moser et al. 2017). It therefore does not seem unlikely that single neurons could represent memories of types or judgements in the way that we have suggested even though we currently do not have experimental evidence of it.

It is perhaps important to bear in mind that the proposal here is not that the activation of these memory neurons is the only way to generate a neural event associated with a type. We assume that an agent confronted with a situation which she classifies as one in which every dog runs will have a neural event corresponding to the type 'every(dog,run)'. This same brain activity will be associated with other situations of this type when the agent classifies them. There may be several different kinds of external stimuli that give rise to this same pattern of brain activity, including hearing someone say the sentence *every dog is running* or reading the sentence in a book. None of this necessarily involves any memory neuron. It is a correlation between situations external to the agent and internal brain states of the agent. The memory neurons become involved when the agent remembers that it was true that every dog ran. In this case the agent is capable of running the neural event again even though an appropriate external stimulus is not present. The stimulus which activates the memory neuron may be of a very different nature to events of dogs running or events where somebody describes situations in which dogs run, but the activation of the memory neuron will engender a neural event of the kind that would have happened if the agent had perceived an event of the type.

7 Three Kinds of Neural Inference

The kind of mapping from types to neural events that we are suggesting in this paper suggests three distinct kinds of neural inference which we shall call: *structural inference (immediate inference)*, *associative inference (fast inference)* and *inference by rule (slow inference)*. We shall discuss each of these in turn.

Structural inference (immediate inference) TTR, like other type theories with records, uses record types to introduce a structural kind of subtyping. Thus, for example, (28a) is a subtype of (28b). Any record of type (28a) will also be of type (28b).

(28)

a. $\begin{bmatrix} x & : & Ind \\ c & : & dog(x) \\ e & : & run(x) \end{bmatrix}$

b. $\begin{bmatrix} x & : & Ind \\ c & : & dog(x) \end{bmatrix}$

Given the compositional nature of our mapping from types to neural events representing them, a neural event representing (28a) will have as a subevent a neural event representing (28b). The inference is immediate in the sense that any representation of (28a) will already include a representation of (28b).

Associative inference (fast inference) This is a kind of inference where the neural architecture is modified in a similar way that we have suggested for memory so that if one type is represented another type will be represented beginning at the next time step. This opening of connections between representations of the two types could be triggered by the agent experiencing events of the second type in association with events of the first type. Hence we call this associative inference. It is also fast inference in the sense that whenever the first type is represented, the second one is represented

immediately afterwards as a consequence of the neural architecture. We have implemented a simple learning agent which uses this kind of inference in a regime related to reinforcement learning. See discussion in Sect. 8.

Inference by rule (slow inference) Slow inference involves drawing a conclusion by applying a learned rule to an object representation. We do not yet have a neural implementation of this reasoning. However, such rules would be implemented as the kind of functions which we used above for properties and we have suggested neural representations for objects. This builds on the approach to reasoning using topoi and enthymemes developed by Breitholtz (2014a, b). An exact account of how this kind of neural reasoning would be implemented on a network remains for future work, however.

It is not obvious from looking at any inference rule in a logical representation which of these kinds of neural inference it should belong to. Consider, for example, the inference 'Give(x,y,z) → Own(y,z)' discussed by Shastri (1999). This could in principle be neurally implemented as any of the three types of inference. There could be a rule that we apply or it could be that whenever there is a neural event corresponding to 'Give(x,y,z)' it is always followed by another neural event corresponding to 'Own(y,z)'. Alternatively we could think of the type corresponding to 'Give(x,y,z)' as containing the type corresponding to 'Own(y,z)', that is, the first type is a subtype of the second: any situation in which x gives z to y is one in which y owns z. Our intuition tells us that for any mature human agent this aspect of the nature of giving is not a matter of slow inference. One could imagine that some inference patterns can become progressively habituated developing from inference by rule to associative inference and finally to structural inference. Equally one might conjecture that some associative inferences can be learned directly from experience based on what types of events are associated.

It is perhaps not obvious that the three kinds of neural inference sketched here exhaust all the kinds of inference that need to be captured. For example, given a well-formed ptype 'dog(a)' we can conclude in TTR that a is of type *Ind* assuming that the arity of the predicate 'dog' is given as '[*Ind*]', that is, the predicate is specified as taking one argument, of type *Ind*. Given the kind of neural representation of ptypes that we have proposed in this paper, this would not count as a structural inference since we have not specified the type of arguments in ptypes as part of the representation of ptypes. One could, however, imagine a slightly more complex neural representation of ptypes where the neural representation of the type of the arguments to a predicate as specified by the arity is activated simultaneously with the neural representation of the argument. In that way the inference would become a structural neural inference. At this stage we leave it open as to whether this is the best way to represent this inference neurally.

8 Relating Neural Representations of Types to the External World

The neuron labelled 'e:every(dog,run)' in (26) and (27) could be said to encode an *Austinian proposition* in memory in something like the sense discussed by Cooper

et al. (2015), that is, it encodes a pair consisting of an object and a type normally represented in TTR as a record with two fields as in (29).

$$(29)\quad \begin{bmatrix} \text{sit} & = & e \\ \text{type} & = & \text{every}(\lambda x{:}\mathit{Ind}\ .\ \text{dog}(x),\ \lambda x{:}\mathit{Ind}\ .\ \text{run}(x)) \end{bmatrix}$$

It says absolutely nothing, however, about what has to happen in the world (or in the agent's perceptual apparatus) in order for this memory to be formed. The problem we have tackled here is only that of how types could be represented by neural events, not how the types could be grounded in terms of witnesses in the real world. One way to do this is to relate the types to neurological implementations of classifiers used in machine learning. The Python implementation of TTR on which the implementation of neural TTR is built provides types with an attribute called `witness conditions` and the witness conditions associated with a type can call functions external to the TTR system itself. This facility could be transferred to the neural implementation of types discussed here. Thus a neural implementation of a classifier could in principle be used to provide a connection between the neural representation of types we have discussed here and the world external to the agent. Realizing this in an implementation is something that remains for future work, however. (See Matsson (2018)) for a promising beginning.) Somewhat similar suggestions have been made in the literature concerning the neural representation of word meaning or concepts, for example, Pulvermüller (2013).

The discussion in the preceding paragraph concerns how an agent might learn the witness conditions associated with a type. Another challenge to be considered is how an agent might modify its neural architecture by making the connections related to associative (fast) inference as discussed in Sect. 7. Here we do have a simple preliminary implementation[3] based on an example in Strannegård et al. (2017). It implements an extremely simple virtual sheep, called Maud, coded as a record with fields for a brain (a network in the sense of the neural TTR implementation), a measure of food (floating point number), a measure of water (floating point number), and a location in an environment. Maud's environment is encoded as a one-dimensional array of areas characterized by the types *Green* (grass), *Blue* (water), *Brown* (sand) and *Green*∧*Blue* (swamp). For simplicity all of the types to which Maud has access are neurally encoded as events involving the activation of a single neuron. For the types *Green*, *Blue* and *Brown* there are corresponding perception neurons: `Green_n`, `Blue_n` and `Brown_n`. Maud has four available action types: *Eat*, *Drink*, *MoveLeft* and *MoveRight*. For each of these types there is a corresponding action neuron: `Eat_n`, `Drink_n`, `MoveLeft_n` and `MoveRight_n`. The activation of one of these neurons indicates that Maud has the corresponding action on her agenda. However, we will see that these neurons may be inhibited again before the action is carried out. In addition Maud has `Pleasure` and `Pain` neurons. `Pleasure` is activated when she eats in a green area or drinks in a blue area (provided she is hungry or thirsty). `Pain` is activated when she eats or drinks in an area which is both green and blue (a swamp area) since she gets sick and loses food and water. Learning for Maud involves a simplified variant of reinforcement learning where excitation of the pleasure neuron

[3] https://github.com/GU-CLASP/pyttr/blob/master/animat.ipynb.

corresponds to positive reinforcement and excitation of the pain neuron corresponds to punishment. She dies if she has either no food or no water, that is, either of the food or water indicators shows zero or less.

If Maud carries out random actions and does not learn from them, she survives in this environment for around ten actions. If she learns based on single types, that is she learns to drink in a blue area and eat in a green area but cannot learn that she should not eat or drink in an area which is both green and blue, she survives for around 15–20 actions. However, if she is able to learn from conjunctions of types, that is, she can learn the simple non-monotonic inference system in (30), then with some luck she seems to be able to survive indefinitely.

$$(30)\quad \frac{Green}{Eat}\quad \frac{Blue}{Drink}\quad \frac{Green \wedge Blue}{\neg Eat}\quad \frac{Green \wedge Blue}{\neg Drink}$$

The technique used for achieving this involves adding connections between perception neurons and action neurons, excitatory if the pleasure neuron is active and inhibitory if the pain neuron is active. This is carried out if no such connection already exists. If more than one perception neuron is active at the same time a new perception neuron is created to represent the conjunction (if no such neuron already exists). Thus in Maud's neural activity pattern in (31) we see that while the activation of the perception neuron `Green_n` activates the action neuron `Eat_n` (indicating an intention to eat), the fact that the perception neuron `Blue_n` is simultaneously active with `Green_n` activates the added neuron `Perception0` (corresponding to *Green*∧*Blue*) which then inhibits the neuron `Eat_n` before the action is carried out.

(31)

`Green_n`	0	0	1	1	1	1	1
`Blue_n`	1	0	0	1	1	1	1
`Brown_n`	0	0	0	0	0	0	0
`Eat_n`	0	0	0	0	0	1	0
`Drink_n`	0	0	0	0	0	0	0
`MoveLeft_n`	1	0	0	0	0	0	0
`MoveRight_n`	0	0	0	0	0	0	0
`Pleasure_n`	0	0	0	0	0	0	0
`Pain_n`	0	0	0	0	0	0	0
`Perception0`	0	0	0	0	1	1	1

While this is perhaps all rather trivial from the point of view of reinforcement learning in a machine learning context, it might nevertheless give us a way to start thinking about how an agent equipped with a neural system representing types in the way that we have suggested in Sects. 2–6 could perceive and interact with its environment.

9 Conclusion

We have suggested a way in which types as discussed in TTR could be represented as neural events in a network. Representing types as neural events rather than neural

architecture enabled us to give simple-minded solutions to the problem of binding and the problem of recursion where the fact that the network can grow (or adjust itself) during the course of computation seems important for the latter. We also suggested a way in which this event approach to representation can be made compatible with storage in memory by introducing memory neurons which, when activated, will give rise to appropriate events. The introduction of delay circuitry was important for this. This neural model of types suggests three kinds of neural inference: structural (immediate), associative (fast) and rule governed (slow).

This proposal, rather like formal semantics, does not say anything about the way in which representations of such types could be grounded in actual experience and used by an agent interacting with its environment (that is, for example, how it could be incorporated in a system on board a robot). In the final section we suggested a couple of strategies for addressing this. The first involves relating neural classifiers to types and the other involves techniques related to reinforcement learning. Of course, this is still only a minor step towards understanding what might actually be going on in the brain.

Acknowledgements This work was supported by the Swedish Research Council (VR) Project 2013-4873, Networks and Types and a Grant from the Swedish Research Council (VR Project 2014-39) for the establishment of the Centre for Linguistic Theory and Studies in Probability (CLASP) at the University of Gothenburg. I am grateful to Simon Dobnik, Staffan Larsson and Claes Strannegård for important comments on various stages of this work. I am also grateful to Ann Copestake and Oluf Andersen for pointing me to some important references. Three anonymous referees provided insightful and extremely useful comments which have led to a number of modifications to the original submitted version. This paper is a revised and extended version of Cooper (2017b).

References

Barwise, J., & Cooper, R. (1981). Generalized quantifiers and natural language. *Linguistics and Philosophy*, *4*(2), 159–219.

Barwise, J., & Perry, J. (1983). *Situations and attitudes*. Cambridge: MIT Press.

Breitholtz, E. (2014a). *Enthymemes in dialogue: A micro-rhetorical approach*. Ph.D. thesis, University of Gothenburg.

Breitholtz, E. (2014b). Reasoning with topoi—Towards a rhetorical approach to non-monotonicity. In *Proceedings of AISB symposium on "questions, discourse and dialogue: 20 years after making it explicit"*.

Chomsky, N. (2000). *New horizons in the study of language and mind*. Cambridge: Cambridge University Press.

Christiansen, M. H., & Chater, N. (1999). Toward a connectionist model of recursion in human linguistic performance. *Cognitive Science*, *23*(2), 157–205. https://doi.org/10.1016/S0364-0213(99)00003-8.

Cooper, R. (2017a). Adapting type theory with records for natural language semantics. In S. Chatzikyriakidis & Z. Luo (Eds.), *Modern perspectives in type-theoretical semantics, no. 98 in studies in linguistics and philosophy* (pp. 71–94). Berlin: Springer.

Cooper, R. (2017b). Neural TTR and possibilities for learning. In: S. Dobnik, S. Lappin (Eds.), *CLASP papers in computational linguistics: Proceedings of the conference on logic and machine learning in natural language (LaML 2017), Gothenburg, 12–13 June* (Vol. 1). Department of Philosophy,

Linguistics and Theory of Science (FLOV), University of Gothenburg, CLASP, Centre for Language and Studies in Probability, Gothenburg, Sweden, CLASP Papers in Computational Linguistics. http://hdl.handle.net/2077/54911. Accessed 10 Mar 2019.

Cooper, R. (in prep). Type theory and language: From perception to linguistic communication. Draft of book chapters. https://sites.google.com/site/typetheorywithrecords/drafts.

Cooper, R., Dobnik, S., Lappin, S., & Larsson, S. (2015). Probabilistic type theory and natural language semantics. *Linguistic Issues in Language Technology*, *10*(4), 1–45.

Cooper, R., & Ginzburg, J. (2015). Type theory with records for natural language semantics. In S. Lappin & C. Fox (Eds.), *The handbook of contemporary semantic theory* (2nd ed., pp. 375–407). Hoboken: Wiley-Blackwell.

Draguhn, A. (2018). Making room for new memories. *Science*, *359*, 1461–1462.

Fedorenko, E., & Kanwisher, N. (2009). Neuroimaging of language: Why hasn't a clearer picture emerged? *Language and Linguistics Compass*, *3*, 839–65. https://doi.org/10.1111/j.1749-818X.2009.00143.x.

Fedorenko, E., & Kanwisher, N. (2011). Functionally localizing language-sensitive regions in individual subjects with fMRI: A reply to Grodzinsky's critique of Fedorenko & Kanwisher (2009). *Language and Linguistics Compass*, *5*(2), 78–94. https://doi.org/10.1111/j.1749-818X.2010.00264.x.

Fodor, J. A. (1976). *The language of thought*. Sussex: Harvester Press.

Fodor, J. A. (2008). *LOT 2: The language of thought revisited*. Oxford: Oxford University Press.

Gibson, J. J. (1979). *The ecological approach to visual perception*. Houghton: Mifflin and Company.

Jackendoff, R. (2002). *Foundations of language: Brain, meaning, grammar, evolution*. Oxford: Oxford University Press.

Kiela, D. (2011). *Variable binding in biologically plausible neural networks*. Master's thesis, Universiteit van Amsterdam.

Larsson, S. (2013). Formal semantics for perceptual classification. *Journal of Logic and Computation*, *25*(2), 335–369. https://doi.org/10.1093/logcom/ext059.

Luo, Z. (2010). Type-theoretical semantics with coercive subtyping. *Proceedings of SALT*, *20*, 38–56.

Luo, Z. (2011). Contextual analysis of word meanings in type-theoretical semantics. In S. Pogodalla, J. P. Prost (Eds.), *Logical aspects of computational linguistics: 6th international conference, LACL 2011, no. 6736 in lecture notes in artificial intelligence* (pp. 159–174). Springer.

Maguire, E. A., Gadian, D. G., Johnsrude, I. S., Good, C. D., Ashburner, J., Frackowiak, R. S. J., et al. (2000). Navigation-related structural change in the hippocampi of taxi drivers. *Proceedings of the National Academy of Sciences*, *97*(8), 4398–4403. https://doi.org/10.1073/pnas.070039597.

Maguire, E. A., Woollett, K., & Spiers, H. J. (2006). London taxi drivers and bus drivers: A structural MRI and neuropsychological analysis. *Hippocampus*, *16*(12), 1091–1101. https://doi.org/10.1002/hipo.20233.

Martin-Löf, P. (1984). *Intuitionistic type theory*. Naples: Bibliopolis.

Matsson, A. (2018). *Implementing perceptual semantics in type theory with records (TTR)*. Master's thesis, University of Gothenburg, Masters in Language Technology.

Montague, R. (1973). The proper treatment of quantification in ordinary english. In: J. Hintikka, J. Moravcsik, 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, R. (1974). Formal philosophy: Selected papers of Richard Montague. New Haven: Yale University Press, ed. and with an introduction by R. H. Thomason.

Moser, E. I., Moser, M. B., & McNaughton, B. L. (2017). Spatial representation in the hippocampal formation: A history. *Nature Neuroscience*, *20*, 1448–1464.

O'Keefe, J., & Dostrovsky, J. (1971). The hippocampus as a spatial map: Preliminary evidence from unit activity in the freely-moving rat. *Brain Research*, *34*, 171–175.

Pulvermüller, F. (2002). *The neuroscience of language: On brain circuits of words and serial order*. Cambridge: Cambridge University Press.

Pulvermüller, F. (2013). How neurons make meaning: Brain mechanisms for embodied and abstract-symbolic semantics. *Trends in Cognitive Sciences*, *17*(9), 458–470.

Schöneich, S., Kostarakos, K., & Hedwig, B. (2015). An auditory feature detection circuit for sound pattern recognition. *Science Advances*, *1*(8), e1500325–e1500325. https://doi.org/10.1126/sciadv.1500325.

Shastri, L. (1999). Advances in SHRUTI—A neurally motivated model of relational knowledge representation and rapid inference using temporal synchrony. *Applied Intelligence*, *11*(1), 79–108.

Shastri, L., & Ajjanagadde, V. (1993). From simple associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings using temporal synchrony. *Behavioral and Brain Sciences*, *16*, 417–494.

Strannegård, C., Cirillo, S., & Wessberg, J. (2015). Emotional concept formation. In *Proceedings of the eighth conference on artificial general intelligence* (pp. 166–176). Springer.

Strannegård, C., & Nizamani, A. R. (2016). Integrating symbolic and sub-symbolic reasoning. In *International conference on artificial general intelligence* (pp. 171–180). Springer.

Strannegård, C., Svangård, N., Bach, J., & Steunebrink, B. (2017). Generic animats. In *Proceedings of the 10th international conference on artificial general intelligence, Melbourne*.

van der Velde, F., & de Kamps, M. (2006). Neural blackboard architectures of combinatorial structures in cognition. *Behavioral and Brain Sciences*, *29*(1), 37–70. https://doi.org/10.1017/S0140525X06009022.