

# Do delicious lunches take a long time?\*

Robin Cooper  
Göteborg University

Extended abstract for the GSLT internal conference, 2005

## 1 Introduction

In most language technology applications that include a lexicon, this lexicon is a collection of static accounts of the properties of words, such as their meaning. However, in human conversations it is often the case that word-meaning is adjusted to fit the context. Pustejovsky's [10] theory of the Generative Lexicon explores some regular ways in which word meanings shift in context and thus represents an important step towards the implementation of systems which can assign meanings to words dynamically depending on the context in which they occur.

In a recent paper Asher and Pustejovsky [1] propose a complex type theoretical approach to account for cases which had motivated Pustejovsky [10] to introduce dot types. The intuitive idea behind dot types is that they are compositions of two types which nevertheless allow the two individual types to be recovered. Consider the examples in (1), based on examples given in Pustejovsky [10] and Asher and Pustejovsky [1].

- (1) a. The lunch was delicious  
b. The lunch took forever

*delicious* is a predicate that can be used of food but not events.

- (2) a. The blancmange was delicious  
b. ?IFK Göteborg's last game was delicious

The “?” in (2b) indicates that we have to work harder to get an interpretation of the sentence. Certainly, it is not impossible to think in terms of something like the taste of victory but games in general are not something we talk of as being delicious. Pustejovsky would say that game is being coerced to another type in order to get an interpretation. Similarly *take forever* is a predicate which holds of events and not of food.

- (3) a. ?The blancmange took forever  
b. IFK's last game took forever

---

\*This work was supported by Vetenskapsrådet project number 2002-4879 *Records, types and computational dialogue semantics*, <http://www.ling.gu.se/cooper/records/>.

Again we can certainly get interpretations out of (3a) but it involves some kind of coercion. It is not the actual food that takes forever but something like the eating of it, the preparing of it or waiting for it. In the case of *lunch* we do not feel that coercion is necessary. This word seems equally easy to interpret as representing food or as representing an event. Pustejovsky therefore associates it with a dot type *Food·Event*. This is meant to represent that it will simultaneously behave as something of type *Food* and something of type *Event* and accept predicates of food and predicates of events.

A natural view might be that this is simply a case of polysemy, that is, that *lunch* is ambiguous between a food interpretation and an event interpretation. If this were the case it would be hard to see what the motivation of the dot type would be. There would simply be two types one associated with each meaning of the word. However, Asher and Pustejovsky [1] discuss in detail cases of copredication where one occurrence of the word simultaneously has both interpretations.<sup>1</sup> Compare (4a) with (4b) (both examples from Asher and Pustejovsky).

- (4) a. The lunch was delicious but took forever  
 b. <sup>1</sup>The bank specializes in IPO's and is being quickly eroded by the river

The word *bank* is ambiguous between meaning a financial institution and the ground at the side of a river. A single occurrence of the word cannot be used in both senses. However, no such problems seem to occur with *lunch*. It seems that in some way natural language manages to treat lunch as a single object which is simultaneously an event and food.

In this paper I will propose an alternative treatment to that given by Asher and Pustejovsky using type theory with records of the kind that I have developed in Cooper [4, 5, 6] based on work in Martin-Löf type theory [11, 2, 3, 7]. I will suggest that using record types not only gives us a simple and intuitive account of dot types but also allows us to preserve the original intuitive feature structure based accounts of lexical analysis proposed in Pustejovsky [10] while at the same time introducing functions and binding in the way that is necessary for giving formal accounts of compositional semantics.

## 2 Records and record types

In this section we give a very brief intuitive introduction to the kind of type theory we are employing. A more detailed and formal account can be found in [6] and work in progress on the project can be found on <http://www.ling.gu.se/~cooper/records>.

The central idea of records and record types can be expressed informally as follows, where  $T(a_1, \dots, a_n)$  represents a type  $T$  which depends on the objects  $a_1, \dots, a_n$ .

If  $a_1 : T_1, a_2 : T_2(a_1), \dots, a_n : T_n(a_1, a_2, \dots, a_{n-1})$ , a record:

$$(5) \quad \left[ \begin{array}{l} l_1 \\ l_2 \\ \dots \\ l_n \\ \dots \end{array} = \begin{array}{l} a_1 \\ a_2 \\ \dots \\ a_n \end{array} \right]$$

is of type:

---

<sup>1</sup>I believe that such examples were first reported in the literature by McCawley.

$$(6) \quad \left[ \begin{array}{l} l_1 : T_1 \\ l_2 : T_2(l_1) \\ \dots \\ l_n : T_n(l_1, l_2, \dots, l_{n-1}) \end{array} \right]$$

A record is to be regarded as a set of fields consisting of a label and an object. A record type is to be regarded as a set of fields consisting of a label and type. A record is of this type just in case for each field in the type there is a corresponding field in the record (with the same label) and the object in the record field is of the type in the type field. Notice that the record may have additional fields not mentioned in the type. Thus a record will generally belong to several record types and any record will belong to the empty record type. This gives us a notion of subtyping.

Let us see how this notion can be applied to a simple linguistic example. We will take the content of a sentence to be modelled by a record type. The sentence

(7) *a man owns a donkey*

corresponds to a record type:

$$(8) \quad \left[ \begin{array}{l} x : Ind \\ c_1 : \text{man}(x) \\ y : Ind \\ c_2 : \text{donkey}(y) \\ c_3 : \text{own}(x,y) \end{array} \right]$$

A record of this type will be:

$$(9) \quad \left[ \begin{array}{l} x = a \\ c_1 = p_1 \\ y = b \\ c_2 = p_2 \\ c_3 = p_3 \end{array} \right]$$

where

$$(10) \quad \begin{array}{l} a, b \text{ are of type } Ind, \text{ individuals} \\ p_1 \text{ is a proof of } \text{man}(a) \\ p_2 \text{ is a proof of } \text{donkey}(b) \\ p_3 \text{ is a proof of } \text{own}(a, b). \end{array}$$

Note that the record may have had additional fields and still be of this type. The types  $\text{man}(x)$ ,  $\text{donkey}(y)$ ,  $\text{own}(x,y)$  are dependent types of proofs. The use of types of proofs for what in other theories would be called propositions is often referred to as the notion of “propositions as types”. Exactly what type  $\text{man}(x)$  is depends on which individual you choose in your record to be labelled by  $x$ . If the individual  $a$  is chosen then the type is the type of proofs that  $a$  is a man. If another individual  $d$  is chosen then the type is the type

of proofs that  $d$  is a man, and so on. What is a proof? Martin-Löf considers proofs to be objects rather than arguments or texts. For non-mathematical propositions proofs can be regarded as situations or events. For useful discussion of this see [9], p 53ff. We discuss it in more detail in [6].

There is an obvious correspondence between this record type and a discourse representation structure (DRS) as characterised in [8]. The characterisation of what it means for a record to be of this type corresponds in an obvious way to the standard embedding semantics for such a DRS which Kamp and Reyle provide.

Records (and record types) are also recursive in the sense that the value corresponding to a label in a field can be a record (or record type)<sup>2</sup>. For example,

$$(11) \quad r = \left[ \begin{array}{l} f = \left[ \begin{array}{l} f = \left[ \begin{array}{l} ff = a \\ gg = b \end{array} \right] \\ g = c \end{array} \right] \\ g = \left[ \begin{array}{l} h = \left[ \begin{array}{l} g = a \\ h = d \end{array} \right] \end{array} \right] \end{array} \right]$$

is of type

$$(12) \quad R = \left[ \begin{array}{l} f : \left[ \begin{array}{l} f : \left[ \begin{array}{l} ff : T_1 \\ gg : T_2 \end{array} \right] \\ g : T_3 \end{array} \right] \\ g : \left[ \begin{array}{l} h : \left[ \begin{array}{l} g : T_1 \\ h : T_4 \end{array} \right] \end{array} \right] \end{array} \right]$$

given that  $a : T_1, b : T_2, c : T_3$  and  $d : T_4$ . We can use *path-names* in records and record types to designate values in particular fields, e.g.

$$(13) \quad \begin{array}{l} r.f = \left[ \begin{array}{l} f = \left[ \begin{array}{l} ff = a \\ gg = b \end{array} \right] \\ g = c \end{array} \right] \\ R.f.f.ff = T_1 \end{array}$$

The recursive nature of records and record types is important for capturing aspects of linguistic theories which use feature structures, for example, in writing HPSG style grammars.

The theory of records and record types is embedded in a general type theory. This means that we have functions and function types available giving us a version of the  $\lambda$ -calculus. We can thus use Montague's techniques for compositional interpretation. For example, we can interpret the common noun *donkey* as a function which maps records  $r$  of the type  $[x:Ind]$  (i.e. records which introduce an individual labelled with the label 'x') to a record type dependent on  $r$ . We notate the function as follows:

$$(14) \quad \lambda r: [x:Ind] ([c:donkey(r.x)])$$

<sup>2</sup>There is a technical sense in which this recursion is non-essential. These records could also be viewed as non-recursive records whose labels are sequences of atomic labels. See [6] for more discussion.

The type of this function is

$$(15) \quad P = ([x:Ind])RecType$$

This corresponds to Montague's type  $\langle e, t \rangle$  (the type of functions from individuals (entities) to truth-values). In place of individuals we use records introducing individuals with the label 'x' and in place of truth-values we use record types which, as we have seen above, correspond to an intuitive notion of proposition (in particular a proposition represented by a DRS).

Following the proposal for the treatment of generalized quantifiers in Cooper [4] we will treat determiners such as *the* as predicates which hold between two such functions. Thus *the donkey runs* could be represented as

$$(16) \quad [ \text{c} : \text{the}(\lambda r: [x:Ind]([c:donkey(r.x)]), \lambda r: [x:Ind]([c:run(r.x)])) ]$$

In our treatment of dynamic quantifiers in Cooper [4] we required such quantifier predicates to be polymorphic. We will use the notation  $X \sqsubseteq T$  to represent a variable over types which are subtypes of the record type  $T$ , that is, record types which contain at least fields with the same labels as  $T$  and each type corresponding to a particular label,  $\ell$ , corresponds to a subtype of the type in the  $\ell$ -field of  $T$ .<sup>3</sup>

If  $q$  is a quantifier predicate then

1.  $\text{arity}(q) = \langle (X \sqsubseteq [x:Ind])RecType, (X \sqsubseteq [x:Ind])RecType \rangle$
2.  $q$  is associated with a relation between sets  $q^*$  (the relation between sets from classical generalized quantifier theory) such that
 
$$p : q(f_1 : (T_1)RecType, f_2 : (T_2)RecType)$$
 iff
 
$$p = \langle \{a | \exists r : T_1[f_1(r) \text{ is non-empty} \wedge a = r.x]\}, \{a | \exists r : T_2[f_2(r) \text{ is non-empty} \wedge a = r.x]\} \rangle$$
 and  $q^*$  holds between  $p_1$  and  $p_2$

### 3 Treating coercion and copredication

Allowing generalized quantifier predicates to be polymorphic allows us to add additional constraints on the domains of functions corresponding to nouns and verb-phrases. For example, we require that records which are arguments to *take forever* in addition to introducing an individual also introduce a proof that that individual is an event:

$$(17) \quad \lambda r: \left[ \begin{array}{l} x : Ind \\ c_1 : \text{event}(x) \end{array} \right] ([c_2 : \text{take\_forever}(r.x)])$$

Similarly *be delicious* will require that its subject is food.

<sup>3</sup>The presence of dependent types means that we have to be careful with the exact formulation of the subtype relation, but for present purposes we will just use this intuitive informal formulation.

$$(18) \quad \lambda r: \left[ \begin{array}{l} x : \text{Ind} \\ c_1: \text{food}(x) \end{array} \right] ([c_2: \text{be\_delicious}(r.x)])$$

The conjunction *be delicious and take forever* needs to require that its subject is both food and an event:

$$(19) \quad \lambda r: \left[ \begin{array}{l} x : \text{Ind} \\ c_1: \text{food}(x) \\ c_2: \text{event}(x) \end{array} \right] \left( \left[ \begin{array}{l} c_3: \text{be\_delicious}(r.x) \\ c_4: \text{take\_forever}(r.x) \end{array} \right] \right)$$

Similarly, nouns will place additional constraints on their domains.

$$(20) \quad \lambda r: \left[ \begin{array}{l} x : \text{Ind} \\ c_1: \text{food}(x) \end{array} \right] ([c_2: \text{blancmange}(r.x)])$$

$$\lambda r: \left[ \begin{array}{l} x : \text{Ind} \\ c_1: \text{event}(x) \end{array} \right] ([c_2: \text{game}(r.x)])$$

$$\lambda r: \left[ \begin{array}{l} x : \text{Ind} \\ c_1: \text{food}(x) \\ c_2: \text{event}(x) \end{array} \right] ([c_3: \text{lunch}(r.x)])$$

Lunch, it will be noted, holds of objects which are both food and an event.

In the theory of dynamic quantifiers developed in Cooper [4] the function which is the first argument to the quantifier predicated is used to further restrict the domain of the second function. Thus *the game took forever* would actually be interpreted in terms of

$$(21) \quad \text{the}(\lambda r: \left[ \begin{array}{l} x : \text{Ind} \\ c_1: \text{event}(x) \end{array} \right] ([c_2: \text{game}(r.x)]), \lambda r: \left[ \begin{array}{l} x : \text{Ind} \\ c_1: \text{event}(x) \\ c_2: \text{game}(x) \end{array} \right] ([c_3: \text{took\_forever}(r.x)]))$$

Thus the first argument to the quantifier predicate provides a context in which the second argument is interpreted by restricting the domain of the second argument. Normally the information passed on by the first argument is a subtype of the domain type of the original second argument. However, if we try to say that the game is delicious, this will not be the case since the second argument will require that objects in its domain are food whereas *game* will only provide the event restriction but not the food restriction. Thus we have something like presupposition accommodation going on within a predication. The verb-phrase *be delicious* will require us to go back and reinterpret the noun by accommodating some information about it being food(-like). With *lunch*, however, we can use either verb-phrases requiring food or events since in either case the function corresponding to *lunch* will provide us with a subtype of the domain type of the second argument to the quantifier.

## 4 An application to dialogue systems for networked devices

Long delicious lunches are all very well but does this have anything to do with practical language technology applications? Here is an example from work on dialogue systems. In smart house applications we want to be able to talk with networked devices such as stereos, video machines, lights, fire alarms etc. We want

the language facility of the system to be centralised to at least some degree so that we can give instructions to more than one device at the same time, e.g. *if the phone rings turn down the volume on the stereo*. At the same time we wish to have plug and play facilities on our network so that we do not have to reprogram the whole system if we wish to add a new kind of device to the network. Nevertheless, a new kind of device should add new language capabilities to the central system. One imagines that each device comes with its own language module which can somehow be compiled together with the central language processing unit for the house. How should something like this be organized? One way would be to organize devices into various types which determine the kinds of things you can say to them. For example, stereos, videocorders and lights all belong to the on-off category meaning that you can turn them on and off. Stereos and TVs belong to the volume category which means that you can turn the volume up or down, but you cannot modify the volume on lights. This could be reflected in the language in terms of the following:

$$\begin{aligned}
 (22) \quad & \lambda r: \left[ \begin{array}{l} x : \text{Ind} \\ c_1 : \text{on-off}(x) \end{array} \right] ([c_2 : \text{light}(r.x)]) \\
 & \lambda r: \left[ \begin{array}{l} x : \text{Ind} \\ c_1 : \text{on-off}(x) \\ c_2 : \text{volume}(x) \end{array} \right] ([c_3 : \text{stereo}(r.x)]) \\
 & \lambda r: \left[ \begin{array}{l} x : \text{Ind} \\ c_1 : \text{on-off}(x) \end{array} \right] ([c_2 : \text{turn\_on}(r.x)]) \\
 & \lambda r: \left[ \begin{array}{l} x : \text{Ind} \\ c_1 : \text{volume}(x) \end{array} \right] ([c_2 : \text{turn\_up\_volume}(r.x)])
 \end{aligned}$$

(For convenience I am ignoring the fact that *turn\_on* and *turn\_up\_volume* should properly be two-place predicates.)

Thus the system will allow you to say (23a) but will object to (23b).

- (23) a. Turn on the stereo and turn up the volume  
 b. Turn on the light and turn up the volume

The difference between stereos and lights is parallel to the difference between lunch and blancmange in terms of their respective types in our analysis. New devices plugged into the system need to declare their types and language libraries need to be available to tell the system how to talk to devices of various types. And, of course, this all needs to be governed by standards to make plug and play work in general.

## References

- [1] Asher, Nicholas and James Pustejovsky (2005) Word Meaning and Commonsense Metaphysics, in course materials for Type Selection and the Semantics of Local Context, ESSLLI 2005.
- [2] Betarte, Gustavo (1998) *Dependent Record Types and Algebraic Structures in Type Theory*. Ph.D. thesis. Department of Computing Science, Göteborg University and Chalmers University of Technology.
- [3] Betarte, Gustavo and Alvaro Tasistro (1998) Extension of Martin-Löf's type theory with record types and subtyping. In Sambin, Giovanni and Jan Smith, eds. *Twenty-Five Years of Constructive Type Theory*, Oxford Logic Guides, 36, Oxford University Press, Oxford.

- [4] Cooper, Robin (2004) Dynamic generalised quantifiers and hypothetical contexts. In *Ursus Philosophicus*, a festschrift for Björn Haglund. Department of Philosophy, Göteborg University. Available from <http://www.ling.gu.se/~cooper/records>.
- [5] Cooper, Robin (2005) Records and Record Types in Semantic Theory, *Journal of Logic and Computation*, Vol. 15 No. 2, pp. 99–112.
- [6] Cooper, Robin (forthcoming) Austinian truth, attitudes and type theory. *Research on Language and Computation*.
- [7] Coquand, Thierry, Randy Pollock and Makoto Takeyama (2004) A Logical Framework with Dependently Typed Records. *Fundamenta Informaticae*, **XX**, 1–22.
- [8] Kamp, Hans and Uwe Reyle. *From Discourse to Logic*, Kluwer, Dordrecht, 1993.
- [9] Ranta, Aarne. *Type-Theoretical Grammar*, Clarendon Press, Oxford, 1994.
- [10] Pustejovsky, James (1995) *The Generative Lexicon*, MIT Press, Cambridge, Mass.
- [11] Tasistro, Alvaro (1997) *Substitution, record types and subtyping in type theory, with applications to the theory of programming*. PhD thesis, Department of Computing Science, Göteborg University and Chalmers University of Technology.