

A type theoretic approach to information state update in issue based dialogue management*

Robin Cooper
Göteborg University

Abstract

Although Barwise came from the model theoretic tradition in logic he was sceptical of the kind of model theoretic analysis of natural language that Montague proposed based on the notion of natural languages as sets of model theoretically interpreted sentences, i.e. as formal languages. Instead Barwise sought to build a mathematical approach to language which incorporated an Austinian view of language as speech events relating to small parts of the world (situations). The classic book in situation semantics (Barwise and Perry, 1983) was thought provoking and ground breaking although it did not give a mathematically precise theory of the kind of compositional semantics that Montague had proposed. It also suffered in many parts of the linguistic community because it was trying to address questions about communication rather than a theory of meaning in the classical Montagovian sense. Barwise was interested in characterising the information which an agent can extract from an utterance event rather than the meaning of an expression. This did not sit easily with the linguistic tradition based on the formal language approach which followed Montague.

Such questions of communication and information (rather than meaning) do, however, arise when we try to create theories relating to what is known as dialogue management in computational dialogue systems. In this paper we try to show how a formal theory of dialogue management could be approached by using type theory as a tool to characterise a kind of dialogue management (issue-based dialogue management) developed by Larsson (2002) using the information state update approach and based on Ginzburg's (1996a,b) work on "questions under discussion" which was carried out in a situation semantic framework.

The exercise leads us to some conclusions about the possibility of unifying the formal language approach with the Austinian approach that Barwise advocated.

1 Introduction

During the twentieth century there was a great deal of development in our understanding of the mathematical analysis of natural language both in regard to its structural aspects (syntax, phonology, morphology) and in how it is to be interpreted (semantics, pragmatics). Our approaches to both of these aspects flourished to a large extent as a result of the development of logic and formal language theory. One significant line of development in natural language interpretation runs through Frege, Tarski, Montague and a great deal of linguistic work following on Montague. This line of development is very much a formal language approach, treating natural language as a set of sentences¹ with a model theoretic interpretation. This led to

*This work was supported by Vetenskapsrådet project number 2002-4879, Records, types and computational dialogue semantics. I am grateful to Staffan Larsson for important discussion concerning issue based dialogue management and to Bengt Nordström and Aarne Ranta for teaching me about type theory and how it can be used for the analysis of natural language. Jonathan Ginzburg has been a great source of inspiration and encouragement for many years on matters relating to dialogue semantics and dialogue management.

¹In subsequent linguistic work on Discourse Representation Theory and dynamic semantics sentences were replaced by discourses, i.e. sequences of sentences, in order to account for intersentential semantic phenomena such as discourse anaphora.

significant and powerful results in what came to be known as Montague semantics.

Another line of work ran through Austin and Searle. This line did not regard natural languages as sets of sentences but talked instead of speech acts. It emphasised the fact that when people talk speech events occur. This led to a significant understanding of different kinds of speech acts and appropriateness conditions for certain kinds of acts, such as queries, assertions, warnings and promises. However, it was not clear how this work should be put together with a compositional semantics of the kind that Montague was proposing.

Barwise came to natural language from foundational work on set theory, a line through Tarski and Feferman. Among other things he was influenced by Feferman's work on intensional set theory. Barwise initially believed that foundational problems in set theory could be illuminated if you took into account that mathematicians use natural language to talk informally about mathematical objects. After having been exposed to some work in natural language semantics he quickly came to the view that natural language semantics had its own serious foundational problems and decided that he should first try to tackle those problems before returning to set theory.

While he came from the model theoretic tradition in mathematics he reacted against Montague's model theoretic approach to natural language as a formal language. One problem he saw was the "extensional" treatment of intensionality, treating, for example, propositions as sets of possible worlds with the well known attendant problems of distinguishing between logically equivalent propositions. In contrast, partly through Feferman's influence, he saw intensionality in terms of universes containing real (if abstract) intensional objects like properties and relations (not modelled as extensional sets of some kind). Secondly, it seemed clear to him that natural languages, unlike formal languages, were not sets of strings provided with an interpretation. Rather, his view, following Austin, was that people were agents engaging in speech events, conveying and receiving information about the limited parts of the world they had knowledge of, something which seemed to be very different from interpreting language in terms of possible worlds corresponding to "a way the whole universe could be". The very notion of meaning or sense inherited from Frege seemed suspect. Rather, human beings were engaged in extracting information from utterance events. Two people engaged in the same conversation may get different information from the same speech event depending on the context they were in and the previous knowledge that they had.

In the last ten years or so of the twentieth century speech recognition for machines became viable and considerable effort was devoted to the development of dialogue systems. This work is currently becoming increasingly relevant in commercial development, although there remains a great need for research in order to support the development of natural and general dialogue systems. While much of the practical work that has been done in building running systems has not been based on theoretical linguistic work in semantics, it nevertheless seems clear that the kind of problems which dialogue management needs to address relate to this view of language as events and interpretation as being agent specific and local to the domain. It is perhaps for this reason that it seems so difficult to make mainstream linguistic semantics seem useful for practical dialogue systems. Even if the ideas expressed in classical situation semantics (as in Barwise and Perry, 1983) might seem appealing in this connection, they were not yet systematically developed in a way that could be immediately applicable to the implementation of a dialogue manager.

For several years the research group at our Dialogue Systems Lab has been involved in the development of the information state update approach to the building of dialogue systems and in particular Issue based dialogue management developed in Larsson (2002) and based on Ginzburg's (1996a,b) situation semantic gameboard approach to dialogue, focussing on the notion of questions (or issues) under discussion. Larsson's computational approach to information state updates involves a large collection of update rules which fire when certain conditions in the information state are met in a regime determined by a general control algorithm. An utterance by a dialogue participant will in general unleash a whole chain of such updates and part of the power of the approach lies in the fact that we can define very general update rules which have small effects on the information state and which are not necessarily linked to any particular form of utterance. It gives us a much finer grain on update rules than thinking in terms of single monolithic updates

associated with speaker utterances.

Larsson's formulation of update rules is based on a Prolog implementation and exploits some aspects of Prolog: logic programming variables, backtracking to deal with non-determinism, ordering of update actions within an update rule. This raises the question of what might be a good foundation for a more abstract account of this work which does not depend on any particular features of a programming language. Surprisingly, we are discovering that an approach developed within the proof theoretic tradition seems promising both in this regard and in relating the computational work back to the original situation semantic approach. Martin L of type theory viewed, perhaps, at an appropriate level of abstraction seems to share some of the basic intuitions of situation theory. For example, we have claimed in Cooper (forthcoming a) that the "propositions as types" principle whereby propositions are regarded as types of proof objects can be regarded as relating to the notion of Austinian proposition if we consider proof objects of non-mathematical propositions to be situations, i.e. parts of the world. Type theory also provides us with a "strong" notion of intensionality where distinct types can have the same extension and where types can also be regarded as first class objects with structure. In Cooper (forthcoming a) we argue that this corresponds to the kind of intensionality which was proposed in situation theory.

The work is part of a broader project whose aim is to present a coherent unified approach to natural language dialogue semantics using tools from type theory. We are seeking to do this by bringing together Head Driven Phrase Structure Grammar (HPSG)², Montague semantics³, Discourse Representation Theory (DRT)⁴, situation semantics and issue-based dialogue management into a single type-theoretic formalism. A survey of our approach to the semantic theories (i.e., Montague semantics, DRT and situation semantics) and HPSG can be found in Cooper (forthcoming b). Other work in progress can be found on <http://www.ling.gu.se/~cooper/records>. We give a brief summary here: Record types can be used as discourse representation structures (DRSs). Truth of a DRS corresponds to there being an object of the appropriate record type and this gives us the effect of simultaneous binding of discourse referents (corresponding to labels in records) familiar from the semantics of DRSs in Kamp and Reyle (1993). Dependent function types provide us with the classical treatment of donkey anaphora from DRT in a way corresponding to the type theoretic treatments proposed by Sundholm (1986) and Ranta (1994). At the same time record types can be used as feature structures of the kind found in HPSG since they have recursive structure and induce a kind of subtyping which can be used to mimic unification. Because we are using a general type theory which includes records we have functions available and a version of the λ -calculus. This means that we can use Montague's λ -calculus based techniques for compositional interpretation. From the HPSG perspective this gives us the advantage of being able to use "real" variable binding which can only be approximately simulated in pure unification based systems. From the DRT perspective this use of compositional techniques gives us an approach similar to that of Muskens (1995) and work on λ -DRT Kohlhase *et al.* (1996).

In this paper we shall now show how a treatment of issue-based dialogue management can be developed using type-theoretical tools.⁵ The main aim of this is to show that type theory enables us to give a declarative theoretical treatment of the information state update approach to dialogue and at the same time integrate it with traditional semantic concerns such as intensionality and anaphora. In this we are inspired by Ginzburg's (1996a,b) work which suggests that there is no clear boundary between dialogue management and semantics, just as many researchers have suggested that there is no clear boundary between semantics and pragmatics.

Within the work on computational dialogue systems, dialogue management has been developed as an area separate from semantics. Even researchers with an interest in developing computational semantics have been surprised to find that dialogue management appears to have little need for the kind of intricacies

²Sag *et al.* (2003)

³Montague (1974) is the classic reference.

⁴Kamp and Reyle (1993), van Eijck and Kamp (1997) and much other literature.

⁵A version of this material was originally presented at Catalog '04, The 8th Workshop on the Semantics and Pragmatics of Dialogue, Barcelona, July 19-21, 2004.

involved in traditional formal semantics. And yet it is well-known that the traditional concerns of formal semantics such as intensionality and anaphora are interwoven with dialogue. We will take as a running example in this paper an example of cross-speaker intensional identity

- A: Sam says there's a bug in the program.
 B: Pat is looking for it.

The point of this example is that there is an understanding of the dialogue in which *it* is anaphorically related to *a bug*. However, the dialogue as a whole does not require that there is a bug in the program and neither is either of the speakers *A* or *B* committed to the existence of such a bug. How can we interpret the occurrence of *it* without introducing a bug for it to refer to?

In the remainder of the paper we will give an account of how records in type theory can be used to give an account of compositional semantics (section 2). We will then show how issue based dialogue management can be accommodated within this framework (section 3) and look in some detail at how the updating of information states should be formulated (section 3.2).

2 Records and compositional semantics

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 Cooper (forthcoming a) and work in progress on the project can be found on <http://www.lingu.se/~cooper/records>. While the type theoretical machinery is based on work carried out in the Martin-Löf approach we are making a serious attempt to give it a foundation in standard set theory using Montague style recursive definitions of semantic domains. There are two main reasons for this. The first is that we think it important to show the relationship between the Montague model theoretic tradition which has been developed for natural language semantics and the proof-theoretic tradition associated with type theory. We believe that the aspects of this kind of type theory that we need can be seen as an enrichment of Montague's original programme. The second reason is that we are interested in exploring to what extent intuitionistic and constructive approaches are appropriate or necessary for natural language. For example, we make important use of the notion "propositions as types" which is normally associated with an intuitionistic approach. However, we suspect that our Montague-like approach to defining the type theory to some extent decouples the notion from intuitionism. We would like to see type theory as providing us with a powerful collection of tools for natural language analysis which ultimately do not commit one way or the other to philosophical notions associated with intuitionism.

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:

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

is of type:

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

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 a 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

a man owns a donkey

corresponds to a record type:

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

A record of this type will be:

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

where

- a, b are of type Ind , individuals
- p_1 is a proof of $\text{man}(a)$
- p_2 is a proof of $\text{donkey}(b)$
- p_3 is a proof of $\text{own}(a, b)$.

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 Ranta (1994), p 53ff. We discuss it in more detail in Cooper (forthcoming a).

There is an obvious correspondence between this record type and a discourse representation structure (DRS) as characterised in Kamp and Reyle (1993). 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)⁶. For example,

⁶There 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 Cooper (forthcoming a) for more discussion.

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

is of type

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

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.

$$r.f = \left[\begin{array}{l} f = \left[\begin{array}{l} ff = a \\ gg = b \end{array} \right] \\ g = c \end{array} \right]$$

$R.f.f.f = T_1$

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.

Another important aspect of the type theory we are using is that types themselves can also be treated as objects.⁷ A simple example of how this can be exploited is the following representation for *a girl believes that a man owns a donkey*. This is a simplified version of the treatment discussed in Cooper (forthcoming a).

$$\left[\begin{array}{l} x : Ind \\ c_1 : girl(x) \\ c_2 : believe(x, \left[\begin{array}{l} y : Ind \\ c_3 : man(y) \\ z : Ind \\ c_4 : donkey(z) \\ c_5 : own(y, z) \end{array} \right]) \end{array} \right]$$

The treatment of types as first class objects in this way is a feature which this type theory has in common with situation theory and it is an important component in allowing us to incorporate analyses from situation semantics in our type theoretical treatment.

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 λ -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:

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

⁷In order to do this safely we stratify the types. We define the type system as a family of type systems of order n for each natural number n . The idea is that types which are not defined in terms of other types are of order 0 and that types which are defined in terms of types of order n are of order $n + 1$. We will not discuss this in detail here but rely on the discussion in Cooper (forthcoming a). In this paper we will suppress reference to order in the specification of our types.

The type of this function is

$$P = ([x:Ind])\text{RecType}$$

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). Using the power of the λ -calculus we can treat determiners Montague-style as functions which take two arguments of type P and return a record type. For example, we represent the indefinite article by

$$\lambda R_1:([x:Ind])\text{RecType} \lambda R_2:([x:Ind])\text{RecType} \left(\begin{array}{l} \text{par} \quad : \quad [x : Ind] \\ \text{restr} \quad : \quad R_1 @ \text{par} \\ \text{scope} \quad : \quad R_2 @ \text{par} \end{array} \right)$$

Here we use $F @ a$ to represent the result of applying function F to argument a .

The type theory includes dependent function types. These can be used to give a classical treatment of universal quantification corresponding to DRT's ' \Rightarrow '. For example, an interpretation of *every man owns a donkey* can be the following record type:

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

Records of the type

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

map records r of type

$$\begin{array}{l} x : Ind \\ c_1 : \text{man}(x) \end{array}$$

to records of type

$$\begin{array}{l} y : Ind \\ c_2 : \text{donkey}(y) \\ c_3 : \text{own}(r.x,y) \end{array}$$

Our interpretation of *every man owns a donkey* requires that there exist a function of this type. Why do we use the record type with the label 'f' rather than the function type itself as the interpretation of the sentence? One reason is to achieve a uniform treatment where the interpretation of a sentence is always a record type. Another reason is that the label gives us a handle which can be used to anaphorically refer to the function. This can, for example, be exploited in so-called paycheck examples Karttunen (1969) such as *Everybody receives a paycheck. Not everybody pays it into the bank immediately, though.*

We have shown that the content of a sentence can be treated as a record type. However, we also need a notion of *meaning*. A meaning will be a function from contexts (modelled as records) to contents (modelled as record types), i.e. meanings are objects of type $(T)\text{RecType}$, where T is some record type. A meaning for *A man owns a donkey*, treated as a sentence that does not depend on context would be

$$\lambda r: \text{Rec} \left(\begin{array}{l} x : \text{Ind} \\ c_1 : \text{man}(x) \\ y : \text{Ind} \\ c_2 : \text{donkey}(y) \\ c_3 : \text{own}(x,y) \end{array} \right)$$

of type $(\text{Rec})\text{RecType}$. That is, it is a function that maps any context (record) to the content we previously specified for the sentence.

However, meanings in general are *dependent* functions which are defined only on contexts of a type corresponding to the presuppositions of the sentence. Consider an example with a proper name: *Sam owns a donkey*. The meaning of this will be a function which is defined only on contexts in which there is an individual named Sam.

$$\lambda r: \left[\begin{array}{l} x : \text{Ind} \\ c_1 : \text{named}(x, \text{"Sam"}) \end{array} \right] \left(\begin{array}{l} y : \text{Ind} \\ c_2 : \text{donkey}(y) \\ c_3 : \text{own}(r.x,y) \end{array} \right)$$

Treating meanings as functions from contexts to content yields a notion of meaning like that of Montague (1974) and Kaplan (1989). However, in the classical Montague/Kaplan approach the domain of meanings is fixed to a finite number of indices, such as speaker, hearer, location of speech, time of speech and assignment to free pronouns (variables). In our approach the domain varies depending on the complexity of the sentence. For example, each proper name occurrence will introduce a new parameter for an individual required to have the name corresponding the proper name. There is no upper limit on how many context parameters a given utterance may depend on. This is one of the important claims that situation semantics made about context dependence. In previous treatments we have used simultaneous abstraction (Seligman and Moss, 1997) in order to capture this. The effect of simultaneous abstraction is achieved here by abstracting over records. Records are type theoretic objects which correspond to the notion of context in type theory and a number of researchers (e.g. Ranta, 1994, Ahn, 1995, Piwek, 1998) have proposed the use of type theoretical contexts to treat context dependence in natural language. Whereas contexts in type theory are syntactic expressions, records provide us with type theoretic objects (i.e. objects that have types, can have functions defined on them and so on) which can in addition do the work of contexts.

There are two techniques we exploit in order to treat anaphora compositionally: manifest fields and metavariables.

Manifest fields. This notion is introduced in Coquand *et al.* (2004). It builds on the notion of singleton type. If $a : T$, then T_a is a *singleton type* and $b : T_a$ iff $b = a$. A *manifest field* in a record type is one whose type is a singleton type, e.g.

$$\left[x : T_a \right]$$

written for convenience as

$$\left[x=a : T \right]$$

This notion allows record types to be “progressively instantiated”, i.e. intuitively, for values to be specified within a record type. A record type that only contains manifest fields is completely instantiated and there will be exactly one record of that type. We will allow dependent singleton types, where a in T_a can be represented by a path in a record type. Manifest fields are important not only for the treatment of anaphora but also for using record types to represent feature structures as used in linguistic theories like HPSG.

Metavariables. Metavariables have been used in type theoretical approaches to proof editing (Magnusson and Nordström, 1994, Magnusson, 1996). We will use occurrences of metavariables (anonymous variables) ‘?’ in manifest fields in order to treat anaphoric constructions. They will be resolved to paths. In the future we plan to use a variant of David Beaver’s (2004) optimality theoretic (OT) version of centering theory for resolution and we suspect that metavariables can be used for other kinds of underspecification such as quantifier scope and that we may be able to use OT here as well.

In order to treat the definite description *the program* in our running example we will need a notion of resource corresponding to resource situation in situation semantics. This is discussed in Cooper (forthcoming b). We give a brief characterisation here. The essential idea is to interpret a noun-phrase such as *the program* as “the unique object which is a program in situation (or context) s ”. In our record-based approach we use records to model situations (contexts). Thus the interpretation becomes “the unique thing which is a program in record r ”. In order to achieve this we introduce a new way of constructing types which is similar to the construction of singleton types discussed on page 8. We will use the notation $T \upharpoonright r$ to represent type T restricted to record r . If T is a type and r is a record (of some type) then $T \upharpoonright r$ is a type. $a : T \upharpoonright r$ just in case for some ℓ occurring in r , $r.\ell = a$ and $a : T$ (equivalently, $r : [\ell:T]$). Intuitively, $T \upharpoonright r$ is the type of objects of type T which occur as a value in r . We refer to r in $T \upharpoonright r$ as a *resource*.

Exploiting these techniques we can assign meanings to the two sentences in our running example.⁸

Meaning of *Sam says there’s a bug in the program*

$$\lambda r : \left[\begin{array}{l} x \quad : \quad Ind \\ c_1 \quad : \quad \text{named}(x, \text{“Sam”}) \\ y \quad : \quad Ind \\ \text{res} \quad : \quad Rec \\ c_2 \quad : \quad \text{program}(y) \upharpoonright \text{res} \\ c_3 \quad : \quad (r_1 : \left[\begin{array}{l} z \quad : \quad Ind \\ c \quad : \quad \text{program}(z) \upharpoonright \text{res} \end{array} \right]) \left[\begin{array}{l} c \quad : \quad \text{eq}(r_1.z, y, Ind) \end{array} \right] \end{array} \right]$$

$$\left(\left[\begin{array}{l} p = \lambda r_2 : Rec \left(\left[\begin{array}{l} x \quad : \quad Ind \\ c_4 \quad : \quad \text{bug}(x, r.y) \end{array} \right] \right) \\ c_5 \quad : \quad \text{say}(r.x, p) \end{array} \right] : (Rec)RecType \right)$$

Meaning (underspecified) of *Pat is looking for it*

$$\lambda r_1 : \left[\begin{array}{l} x \quad : \quad Ind \\ c_1 \quad : \quad \text{named}(x, \text{“Pat”}) \end{array} \right]$$

$$\left(\left[\begin{array}{l} p = \lambda r_2 : ? ([y=?Ind]) \quad : \quad (?)RecType \\ c_2 \quad : \quad \text{seek}(r_1.x, p) \end{array} \right] \right)$$

There are three occurrences of metavariables, ‘?’, in the latter meaning. Anaphora resolution, which we will now integrate with dialogue management, needs to specify: (i) the type of context in which the *it* is interpreted (first and third question marks) and (ii) which individual in a context of that type *it* is to denote (second question mark).

⁸We do not show the details of compositional derivation here as that is not the central focus of this paper.

3 Issue based dialogue management

The term *issue based dialogue management* was introduced in Larsson (2002) and we will follow certain parts of this text quite closely in formulating our type theoretic approach. Issue based dialogue management is a species of the *information state update approach* to dialogue management. The main characteristic of this approach is that dialogue is seen in terms of information states associated with each dialogue participant representing information about the state of the dialogue and other information relevant to the interpretation and progress of the dialogue. An important aspect of the information state approach as it has grown up in the literature is that the information states involved are explicitly treated as complex structured objects rather than atomic states as in a finite state approach. Thus, for example, one finds the gameboard concept of Ginzburg(1996a,b) and the use of records in the TRINDI approach⁹ from which Larsson (2002) has developed. With each contribution to the dialogue an agent's information state gets changed in order to reflect (the agent's view of) the new state of information in the dialogue. This change is referred to as an information state *update*. An important feature of the information state update approach is that such updates are often non-monotonic. For example, issue based dialogue management makes use of a list of *questions under discussion (QUD)*, introduced in Ginzburg (1996a,b). This keeps track of issues that are currently the focus of attention in the dialogue. It is obvious that QUD potentially changes from one dialogue turn to another and that questions must not only be added to QUD when they are raised but also removed when they are no longer under discussion. In this way an information state based approach is quite different to update semantics (Veltman, 1996) or classical dynamic semantics for natural language (Groenendijk and Stokhof, 1991) or even classical discourse representation theory (DRT) (Kamp and Reyle, 1993) although DRT, given its structured discourse representation structures, has the potential for straightforward treatment of non-monotonic updates.

In issue based dialogue management determination of the next dialogue contribution is driven largely by QUD. Almost all contributions either raise or address an issue (a question under discussion). An important feature of the issue based approach is that issues can be raised by addressing them, i.e. by giving an answer to a question that has not been stated explicitly. This phenomenon is called *question accommodation*. Another important aspect of Larsson's formulation of issue based dialogue management is that a single utterance may unleash a whole chain of updates. This means that the system allows us to capture generalisations about updates beyond those associated with monolithic utterance updates.

Given that Larsson has already given a formulation of issue based dialogue management, why should we be interested in giving a type theoretic formulation of it? Firstly, we are interested in integrating dialogue management into our approach to semantics and feature based analysis and producing a single comprehensive framework which uses the same formal tools for the analysis of syntax, semantics and dialogue management. Secondly, we are interested in approaching dialogue management at a slightly higher level of abstraction than Larsson was able to do in his implementation based formulation. The advantage of this we see not only as providing a possibility for including dialogue management in a general linguistic theory (as envisioned by Ginzburg, 1996a,b) but also showing the way to a more abstract computational treatment such as the definition of an abstract dialogue update machine that can in principle be implemented in any programming language and does not for example depend on the Prolog style constructions which Larsson employs such as ordering of effects within update rules, ordering of update rules themselves and also Prolog style variables. (For the last see also discussion by Johan Bos *et al.*, 2003, in connection with the Dipper system.)

⁹<http://www.ling.gu.se/research/projects/trindi>

3.1 Information states as records

Here we introduce the information states from the second chapter of Larsson (2002) which presents the simplest of a series of increasingly complex systems discussed in the work. We start with an example of a simple information state in our formulation¹⁰

$$\left[\begin{array}{l} \text{private=} \\ \text{shared=} \end{array} \left[\begin{array}{l} \text{agenda} = [] \\ \text{plan} = \left[\begin{array}{l} \text{raise}(\lambda r: \left[\begin{array}{l} x:Ind \\ c:means_of_transport(x) \end{array} \right] ([c:how(r.x)])), \\ \text{findout}(\lambda r: \left[\begin{array}{l} x:Ind \\ c:city(x) \end{array} \right] ([c:dest_city(r.x)])), \\ \text{findout}(\lambda r: \left[\begin{array}{l} x:Ind \\ c:city(x) \end{array} \right] ([c:dep_city(r.x)])), \\ \text{findout}(\lambda r: \left[\begin{array}{l} x:Ind \\ c:month(x) \end{array} \right] ([c:dep_month(r.x)])), \\ \text{findout}(\lambda r: \left[\begin{array}{l} x:Ind \\ c:date(x) \end{array} \right] ([c:dep_date(r.x)])), \\ \text{findout}(\lambda r: \left[\begin{array}{l} x:Ind \\ c:travel_class(x) \end{array} \right] ([c:class(r.x)])), \\ \text{consultDB}(\lambda r: \left[\begin{array}{l} x:Ind \\ c:amount_money(x) \end{array} \right] ([c:price(r.x)])) \end{array} \right] \\ \text{bel} = [] \\ \text{com} = [] \\ \text{qud} = \left[\begin{array}{l} \text{speaker} = \text{usr} \\ \text{moves} = \{ \text{ask}(\lambda r: \left[\begin{array}{l} x:Ind \\ c:amount_money(x) \end{array} \right] ([c:price(r.x)])) \} \end{array} \right] \end{array} \right]$$

In the following discussion we will use the variable r to refer to this record. r is an information state which represents one of the dialogue participant's view of the state of the dialogue (in this case the system's view in a dialogue system concerned with travel booking). The information is divided into that which is private to the dialogue participant and the dialogue participant's view of that which is shared information. The private information consists of

- an agenda, a list of actions which are to be carried out by the agent immediately
- a plan, a list of actions which represent goals which the agent has for the dialogue as a whole. In issue based dialogue management these actions often are based on questions such as raising an issue for discussion (in this example, raising the issue of what means of transport is to be used for the trip), finding out the answer to a question (e.g. what the destination city is) and consulting the database to find the answer to a question (represented here by *consultDB*).
- a representation of private beliefs

The agent's view of shared information, in this example, consists of

- a representation of shared commitments (labelled *com*), corresponding to what has been established jointly in the dialogue so far. Note that shared commitments may not necessarily be true or even

¹⁰It is a recoding of the information state to be found on Larsson(2002), p. 56. There is an issue as to whether objects which are means of transport, travel classes, months etc. should be regarded as individuals (i.e., of type *Ind*). I will ignore this issue here as it has not relevance for the present discussion.

believed to be true by the dialogue participants. One may agree to make assumptions which one does not believe for the sake of moving the dialogue forward.

- a list of questions under discussion (*qud*). In this example, the question of the price of the trip is under discussion.
- a representation of the latest utterance (*lu*) indicating who the speaker was and the set of dialogue moves carried out by that utterance, in this example, the single move of asking the price of the trip.

This is an example of the simplest kind of information state that Larsson introduces. In later chapters he introduces more fields into the information state in order to be able to handle more complex phenomena.

r is of the type¹¹:

$$\left[\begin{array}{l} \text{private} \\ \text{shared} \end{array} : \left[\begin{array}{l} \text{agenda} : [Action] \\ \text{plan} : [Action] \\ \text{bel} : RecType \\ \text{com} : RecType \\ \text{qud} : [Question] \\ \text{lu} : \left[\begin{array}{l} \text{speaker} : Participant \\ \text{moves} : \{Move\} \end{array} \right] \end{array} \right] \right]$$

We call this type *IS*, the type of information states. The types *Move*, *Action* and *Participant* require some explanation here. *Move* is the type of *dialogue moves*. Dialogue moves can be realised by an utterance. In this discussion we will only consider moves involving the asking of a question. We will consider a function *ask* which maps questions to dialogue moves, i.e.,

$ask : (Question)Move$

Such functions correspond to what has been called *illocutionary force* in the literature on speech acts (e.g., Searle, 1969). We might then call such functions *illocutionary functions*. The type *Move* will be defined inductively by a collection, \mathcal{C} , of such illocutionary functions, i.e.

$a : Move$ iff for some type T , function f in \mathcal{C} such that $f : (T)Move$ and object $b : T$, $a = f(b)$

Dialogue moves can be used to perform *actions*.¹² For example, a move of asking a question can be used to raise that question for discussion and might be a step in finding out the answer to the question. Questions can be raised for discussion by other means than asking them. Providing the answer to a question (i.e. an answer dialogue move) that has not been asked can also result in the raising of a question. For example, in

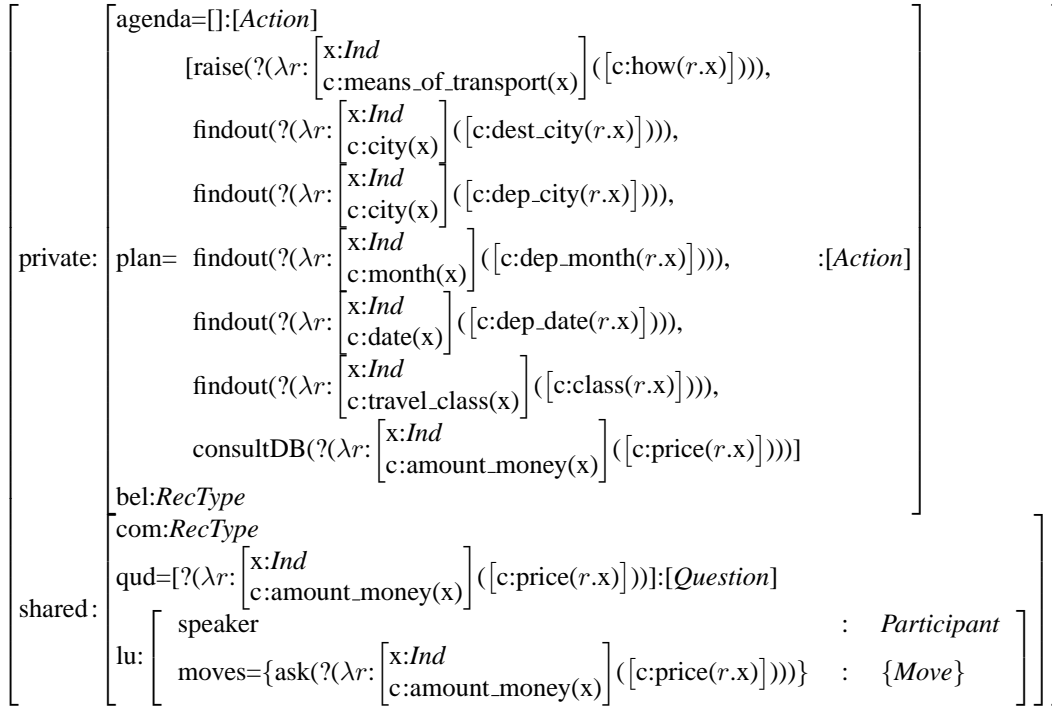
A: Where do you want to go?
B: Chicago, tomorrow

B raises the question of when she wants to go by introducing an answer to it. The type *Action* is defined inductively by a collection of functions which create actions from elements of other types. Here we will use four such functions which map questions to actions.

raise : (Question)Action
findout : (Question)Action
respond : (Question)Action
consultD(ata)B(ase) : (Question)Action

¹¹Adapted from Larsson(2002), p. 32

¹²This is a slight different view to that expressed by Larsson(2002), p. 30, where dialogue moves are said to be actions.



The subtyping provided by record types combined with manifest fields gives us a convenient way of talking about underspecification. Types can be used as underspecified representations since there is typically more than one object of each type. Singleton types correspond to total specification giving us a way of talking about particular objects without leaving the type level. Using types in this way can give us a way of reasoning about non-deterministic updates without resorting to programming control techniques such as Prolog backtracking. A non-deterministic update at the object level can be recast as a deterministic update at the type level. We shall model information states as records (as Larsson does) and thus think of updates as mappings from records to records, graphically:

Updates

$$\text{record}_1 \Rightarrow \text{record}_2 \Rightarrow \text{record}_3 \Rightarrow \dots$$

However, we shall reason about updates in an underspecified way using types of information states (i.e. record types). Thus our reasoning will involve drawing conclusions about the type of the next information state based on the type of the current information state. As the types need not necessarily be singleton types we may not know exactly what either the current or next information state is. Reasoning about updates thus involves mappings from record types to record types, graphically:

Reasoning about updates

$$\text{record type}_1 \Rightarrow \text{record type}_2 \Rightarrow \text{record type}_3 \Rightarrow \dots$$

Before we move on to a discussion of the nature of this reasoning we need to say something more about the nature of information states as they are presented in Larsson (2002). Records of the type *IS*, referred to as the “information state proper”, are embedded in larger record structures which provide information about resources the system has access to (e.g. grammars, lexicons, databases), module interfaces (e.g. information specifically related to output from or input to various modules such as speech recognition, speech synthesis, interpreter) and possibly also various flags relating to the state of the system (e.g. language in

use, application domain). This larger structure is referred to as the *total information state*. It is not our purpose here to discuss the details of this although it will be important to be able to refer to it in our discussion of update rules. We will call the type of total information states *TIS* and represent it as:

$$\left[\begin{array}{l} \text{is} \quad : \quad IS \\ \text{res} \quad : \quad RES \\ \text{mod} \quad : \quad MOD \\ \text{flags} \quad : \quad FLAGS \end{array} \right]$$

3.2 Update rules in type theory

The basic intuition behind our reasoning about information state updates can be expressed as follows:

If $r_i : T_i$, then $r_{i+1} : T_{i+1}(r_i)$

That is, given that we believe that the current information state is of type T_i (recall that we can come to this belief without having any belief about which specific information state is involved), then we can conclude that the next information state is of type T_{i+1} which can depend on the current information state. According to this, we can have a hypothesis about the type of the next information state even though we may not know exactly what the current information state is. Exactly which type the next information state belongs to depends, though, on the exact nature of the current information state. Thus the dependency in our types provides us with an additional means for representing underspecification.

This basic rule of inference corresponds to a function from records to record types, i.e., a function of type $(T_i)RecType$ (where T_i is a type of the current information state):

$\lambda r : T_i(T_{i+1}(r_i))$

We will call such functions *update functions*.¹³ It is important to note that update functions are the same kinds of functions as our previous (static) meaning functions. Meaning functions are construed as functions from contexts (records) to contents of utterances (record types). Update functions can similarly be regarded as taking contexts as their arguments. The difference is that in the case of update functions it is the discourse so far that provides the context for the current utterance.

We will need to express additional conditions on the information state and we will do this using records with a field for the total information state and another field which expresses conditions on the total information state, i.e., of the following type:

$$\left[\begin{array}{l} \text{tis} \quad : \quad TIS \\ \text{cond} \quad : \quad T_{cond} \end{array} \right]$$

where T_{cond} is a type which expresses some conditions which must hold of the total information state.

Let us look at a specific example in order to make this concrete. We formulate an update function corresponding to Larsson's update rule *integrateUsrAsk* (Larsson, 2002, p. 40). The rule is used to update the system's information state when the user has just asked a question. The update involves adding an action to respond to the question at the head of the system's agenda and also pushing the question onto the top of the QUD stack, i.e. it is now the topmost question under discussion. The corresponding update function is as follows:

¹³Update *rules* will specify various pieces of information including an update function. If we follow Chapter 2 of Larsson (2002) the other information specified will be the name of the rule and its class but one could imagine other information being specified in an update rule.

$$\lambda r : \left[\begin{array}{l} \text{tis} : \left[\begin{array}{l} \text{is} : \left[\begin{array}{l} \text{private} : \left[\begin{array}{l} \text{agenda} : [Action] \\ \text{lu} : \left[\begin{array}{l} \text{speaker=usr} : Participant \\ \text{moves} : \{Move\} \end{array} \right] \\ \text{qud} : [Question] \end{array} \right] \\ \text{shared} : \left[\begin{array}{l} \text{q} : Question \\ \text{c} : \text{member}(\text{ask}(\text{cond.q}), \text{tis.is.shared.lu.moves}) \end{array} \right] \end{array} \right] \\ \text{cond} : \left[\begin{array}{l} \text{private} : \left[\begin{array}{l} \text{agenda} = \text{respond}(r.\text{cond.q}) | r.\text{tis.is.private.agenda} : [Action] \\ \text{qud} = r.\text{cond.q} | r.\text{tis.is.shared.qud} : [Question] \end{array} \right] \end{array} \right] \end{array} \right] \end{array} \right]$$

The domain type of this function requires that the speaker of the latest utterance (corresponding to the label ‘lu’) was the user and the condition type requires that there is some question q such that $\text{ask}(q)$ is a member of the set of moves associated with the latest utterance. The type that results from applying the function to such a record requires that an action $\text{respond}(q)$ is added to the beginning of the agenda and that q is added to the beginning of QUD.¹⁴

While this update function represents an inference from the type of the current information state to the type of the next information state, it does not represent all the information that we need for inference about updates. Suppose, for example, that the current information state is of type¹⁵

$$\left[\begin{array}{l} \text{tis} : \left[\begin{array}{l} \text{is} : \left[\begin{array}{l} \text{private} : \left[\begin{array}{l} \text{agenda} = [] : [Action] \\ \text{plan} = [] : [Action] \\ \text{bel} = [] : RecType \\ \text{com} = [] : RecType \\ \text{qud} = [] : [Question] \end{array} \right] \\ \text{shared} : \left[\begin{array}{l} \text{speaker} = \text{usr} : Participant \\ \text{lu} : \left[\begin{array}{l} \text{moves} = \{ \text{ask}(\lambda r : [x:Ind] \\ (\text{c:time_next_train}(\text{Barcelona}, r.x))) \} : \{Move\} \end{array} \right] \end{array} \right] \end{array} \right] \\ \dots \end{array} \right] \end{array} \right]$$

that is, we know (or think we know) that the agenda, plan, beliefs, commitments and QUD are all empty and that the user has just asked the time of the next train to Barcelona. Then we should be able to reason that the next information state is of type

$$\left[\begin{array}{l} \text{tis} : \left[\begin{array}{l} \text{is} : \left[\begin{array}{l} \text{private} : \left[\begin{array}{l} \text{agenda} = [\text{respond}(\lambda r : [x:Ind] (\text{c:time_next_train}(\text{Barcelona}, r.x)))] : [Action] \\ \text{plan} = [] : [Action] \\ \text{bel} = [] : RecType \\ \text{com} = [] : RecType \\ \text{qud} = [\lambda r : [x:Ind] \\ (\text{c:time_next_train}(\text{Barcelona}, r.x))] : [Question] \end{array} \right] \\ \text{shared} : \left[\begin{array}{l} \text{speaker} = \text{usr} : Participant \\ \text{lu} : \left[\begin{array}{l} \text{moves} = \{ \text{ask}(\lambda r : [x:Ind] \\ (\text{c:time_next_train}(\text{Barcelona}, r.x))) \} : \{Move\} \end{array} \right] \end{array} \right] \end{array} \right] \\ \dots \end{array} \right] \end{array} \right]$$

where an action to respond to the question has been pushed onto the agenda, the question itself has been pushed onto qud and the plan, beliefs, commitments and latest utterance fields remain the same. We know more about the type of the new information state than the function expresses since we must carry over all that we knew about the type which is not explicitly referred to in the function. There are three approaches that we could take to making sure that this information is carried over in our reasoning about updates.

The first is to require that our update functions specify all the fields in *TIS* both in the domain type and

¹⁴We use standard list notation $h|t$ for a list whose head is h and tail is t .

¹⁵We use ‘...’ to suppress reference to fields other than *is* within *tis* as they are not relevant to the present discussion.

the range. This would perhaps be the simplest strategy though it would make the functions harder to read. It would have a serious disadvantage in that we would not be able to reuse update functions if we decide to redefine *TIS*, for example by including an extra field. We do not want to have to alter all our update functions every time we decide to include additional information in the total information state.

The second is to derive destructive operations on types from our update functions. This would mean defining operations that modify a particular type and produce a new type, for example, by using procedural operations such as `push` and `pop`. From the point of view of implementation this makes a lot of sense. The most straightforward way to preserve information in a data structure is to carry out destructive modifications on the data structure itself. One way to give a logical account of destructive operations in information state updates is to use dynamic logic (Fernández, 2003).

The third approach shows what kind of logical manipulations could underly an implementation with destructive operations. It involves reasoning declaratively with the types and the update functions. It is this approach which we will develop here, in part to show that the reasoning involved is fairly straightforward and makes crucial use of fixed point types which were used in connection with anaphora in Cooper (forthcoming b).

There are five steps involved in the computation of the type of the next information state from the type of the current information state on the basis of an update function.

1. instantiation of the update function with the type of the current information state
2. relabelling of shared paths
3. computing the fixed point type of the resulting function
4. reasoning about identity
5. garbage collection of unused “support” fields

We shall take each of these in turn.

Instantiation of an update function Suppose that an update rule has the update function $\lambda r : T_1(T_2(r))$ and that what we believe about the current information state is that it is of type T_{curr} . We will say that the rule only *fires* if $T_{curr} \sqsubseteq T_1$.¹⁶ That is, the rule cannot be used unless this condition is met. We *instantiate* the update function to $\lambda r : T_{curr}(T_2(r))$.

Relabelling of shared paths For the next step we will need to ensure that T_{curr} and $T_2(r)$ do not have any paths in common which may have different values (see `integrateUsrAsk`). We therefore prefix any shared path in T_{curr} with the label ‘prev(ious)’.¹⁷ The result is thus a relabelled instantiated update function.

Fixed point types of functions A *fixed point type* for a function $f : (T)RecType$ is a type T such that $a : T$ iff $a : f(a)$.

If $f = \lambda r : T_1(T_2(r))$ and $T_1, T_2(r)$ do not share any paths then the fixed point type of f is $T_1 \cup T_2(r)'$ (where $T_2(r)'$ is like $T_2(r)$ except that dependence on r has been replaced by dependence on corresponding

¹⁶We use ‘ \sqsubseteq ’ to represent the subtype relation.

¹⁷Any record or record type can be equivalently considered as a set of paths and values (Cooper, forthcoming a). A record type in graph format can be first converted to path-value format and the prefix can be inserted where necessary before converting the result back to graph format.

fields in the new type). The third step is to compute the fixed point type of the relabelled instantiated update function.

Let us take the example update function from page 16. We present a version of it instantiated by the current information state on page 16 and also relabelled.

$$\lambda r : \left[\begin{array}{l} \text{prev} : \left[\begin{array}{l} \text{tis} : \left[\begin{array}{l} \text{is} : \left[\begin{array}{l} \text{private} : \left[\begin{array}{l} \text{agenda}=[] : [Action] \end{array} \right] \\ \text{shared} : \left[\begin{array}{l} \text{qud}=[] : [Question] \end{array} \right] \end{array} \right] \end{array} \right] \\ \text{tis} : \left[\begin{array}{l} \text{is} : \left[\begin{array}{l} \text{private} : \left[\begin{array}{l} \text{plan}=[]:[Action] \\ \text{bel}=[] :RecType \\ \text{com}=[]:RecType \end{array} \right] \\ \text{shared} : \left[\begin{array}{l} \text{speaker}=usr \\ \text{lu} : \left[\begin{array}{l} \text{moves}=\{\text{ask}(\lambda r : [x:Ind] \\ \quad ([c:\text{time_next_train}(\text{Barcelona}, r.x))))\} : \{Move\} \end{array} \right] \end{array} \right] \end{array} \right] \\ \text{cond} : \left[\begin{array}{l} \text{q} : Question \\ \text{c} : \text{member}(\text{ask}(\text{cond.q}), \text{tis.is.shared.lu.moves}) \end{array} \right] \end{array} \right] \\ \left(\text{tis} : \left[\begin{array}{l} \text{is} : \left[\begin{array}{l} \text{private} : \left[\begin{array}{l} \text{agenda}=\text{respond}(r.\text{cond.q})|r.\text{prev.tis.is.private.agenda} : [Action] \\ \text{shared} : \left[\begin{array}{l} \text{qud}=r.\text{cond.q}|r.\text{prev.tis.is.shared.qud} : [Question] \end{array} \right] \end{array} \right] \end{array} \right] \end{array} \right) \end{array} \right] \end{array} \right]$$

Computing the fixed point type We then compute the fixed point type of this function:

$$\left[\begin{array}{l} \text{prev} : \left[\begin{array}{l} \text{tis} : \left[\begin{array}{l} \text{is} : \left[\begin{array}{l} \text{private} : \left[\begin{array}{l} \text{agenda}=[] : [Action] \\ \text{shared} : \left[\begin{array}{l} \text{qud}=[] : [Question] \end{array} \right] \end{array} \right] \end{array} \right] \\ \text{tis} : \left[\begin{array}{l} \text{is} : \left[\begin{array}{l} \text{private} : \left[\begin{array}{l} \text{agenda}=\text{respond}(r.\text{cond.q})|r.\text{prev.tis.is.private.agenda}:[Action] \\ \text{plan}=[] :[Action] \\ \text{bel}=[] :RecType \\ \text{com}=[]:RecType \\ \text{qud}=r.\text{cond.q}|r.\text{prev.tis.is.shared.qud}:[Question] \end{array} \right] \\ \text{shared} : \left[\begin{array}{l} \text{speaker}=usr \\ \text{lu} : \left[\begin{array}{l} \text{moves}=\{\text{ask}(\lambda r : [x:Ind] \\ \quad ([c:\text{time_next_train}(\text{Barcelona}, r.x))))\} : \{Move\} \end{array} \right] \end{array} \right] \end{array} \right] \\ \text{cond} : \left[\begin{array}{l} \text{q} : Question \\ \text{c} : \text{member}(\text{ask}(\text{cond.q}), \text{tis.is.shared.lu.moves}) \end{array} \right] \end{array} \right] \end{array} \right] \end{array} \right]$$

Reasoning about identity The type labelled by *tis* is now the type assigned by the update to the new information state. It is dependent on both the *prev* and *cond* fields. However, there are some equivalences that can be used to obtain a type for the total information state which does not depend on these fields. Firstly, since *tis.is.shared.lu.moves* is required to be a singleton set we can determine the value of *cond.q* and insert a manifest field at the end of this path.

$$\left[\begin{array}{l} \text{prev} : \left[\begin{array}{l} \text{tis} : \left[\begin{array}{l} \text{is} : \left[\begin{array}{l} \text{private} : \left[\begin{array}{l} \text{agenda}=[] : [Action] \end{array} \right] \\ \text{shared} : \left[\begin{array}{l} \text{qud}=[] : [Question] \end{array} \right] \end{array} \right] \\ \text{private} : \left[\begin{array}{l} \text{agenda}=\text{respond}(r.\text{cond}.q)|r.\text{prev}.\text{tis}.\text{is}.\text{private}.\text{agenda}:[Action] \\ \text{plan}=[] : [Action] \\ \text{bel}=[] : RecType \\ \text{com}=[]:RecType \\ \text{qud}=r.\text{cond}.q|r.\text{prev}.\text{tis}.\text{is}.\text{shared}.\text{qud}:[Question] \\ \text{shared} : \left[\begin{array}{l} \text{speaker}=\text{usr} : Participant \\ \text{lu} : \left[\begin{array}{l} \text{moves}=\{\text{ask}(\lambda r:[x:Ind] \\ (\text{c}:\text{time_next_train}(\text{Barcelona}, r.x))))\}: \{Move\} \end{array} \right] \end{array} \right] \end{array} \right] \\ \text{cond} : \left[\begin{array}{l} q=?(\lambda r:[x:Ind] \\ (\text{c}:\text{time_next_train}(\text{Barcelona}, r.x))) : Question \\ \text{c}:\text{member}(\text{ask}(\text{cond}.q), \text{tis}.\text{is}.\text{shared}.\text{lu}.\text{moves}) \end{array} \right] \end{array} \right] \end{array} \right]
\end{array}$$

When types depend on manifest fields we can substitute the value specified in the manifest field in the type specification and thus remove the dependency. Two such dependencies occur in each of the agenda and QUD fields and thus we can derive the following type.¹⁸

$$\left[\begin{array}{l} \text{prev} : \left[\begin{array}{l} \text{tis} : \left[\begin{array}{l} \text{is} : \left[\begin{array}{l} \text{private} : \left[\begin{array}{l} \text{agenda}=[] : [Action] \\ \text{shared} : \left[\begin{array}{l} \text{qud}=[] : [Question] \end{array} \right] \end{array} \right] \\ \text{private} : \left[\begin{array}{l} \text{agenda}=[\text{respond}(\lambda r:[x:Ind] \\ (\text{c}:\text{time_next_train}(\text{Barcelona}, r.x)))]:[Action] \\ \text{plan}=[] : [Action] \\ \text{bel}=[] : RecType \\ \text{com}=[]:RecType \\ \text{qud}=[\lambda r:[x:Ind] \\ (\text{c}:\text{time_next_train}(\text{Barcelona}, r.x))]:[Question] \\ \text{shared} : \left[\begin{array}{l} \text{speaker}=\text{usr} : Participant \\ \text{lu} : \left[\begin{array}{l} \text{moves}=\{\text{ask}(\lambda r:[x:Ind] \\ (\text{c}:\text{time_next_train}(\text{Barcelona}, r.x))\}: \{Move\} \end{array} \right] \end{array} \right] \end{array} \right] \\ \text{cond} : \left[\begin{array}{l} q=?(\lambda r:[x:Ind] (\text{c}:\text{time_next_train}(\text{Barcelona}, r.x))) : Question \\ \text{c}:\text{member}(\text{ask}(\text{cond}.q), \text{tis}.\text{is}.\text{shared}.\text{lu}.\text{moves}) \end{array} \right] \end{array} \right] \end{array} \right]
\end{array}$$

Garbage collection Note that the type for the total information state is now no longer dependent on the *prev* and *cond* types. As our interest is in deriving the type for the total information state we can remove all paths headed by *prev* or *cond* which are not depended on by any path headed by *tis*. We will in general refer to such a process as *garbage collection* since it corresponds to garbage collection processes in programming languages where objects are removed from memory when nothing points to them. The result of this garbage collection is the type of the new total information state:

$$\left[\begin{array}{l} \left[\begin{array}{l} \left[\begin{array}{l} \left[\begin{array}{l} \text{private} : \left[\begin{array}{l} \text{agenda}=[\text{respond}(\lambda r:[x:Ind] \\ (\text{c}:\text{time_next_train}(\text{Barcelona}, r.x)))]:[Action] \\ \text{plan}=[] : [Action] \\ \text{bel}=[] : RecType \\ \text{com}=[]:RecType \\ \text{qud}=[\lambda r:[x:Ind] \\ (\text{c}:\text{time_next_train}(\text{Barcelona}, r.x))]:[Question] \\ \text{shared} : \left[\begin{array}{l} \text{speaker}=\text{usr} : Participant \\ \text{lu} : \left[\begin{array}{l} \text{moves}=\{\text{ask}(\lambda r:[x:Ind] \\ (\text{c}:\text{time_next_train}(\text{Barcelona}, r.x))\}: \{Move\} \end{array} \right] \end{array} \right] \end{array} \right] \end{array} \right] \end{array} \right] \end{array} \right]
\end{array}$$

¹⁸Note that the list $a[]$ is identical with $[a]$.

3.3 Using meanings to refine beliefs and commitments

This discussion relates to how private beliefs and shared commitments in the information state can be refined by the meanings of dialogue contributions. In Larsson (2002) both these fields are typed as sets of propositions. In our version we use a record type instead of a set of propositions. Record types, it will be recalled, play the role of propositions in our system, in the sense that they represent the content of declarative sentences. A single record type can be thought of as representing a conjunction of various propositions. The advantage of using a single record type rather than a set of record types corresponding to individual propositions is that it becomes more straightforward to represent dependencies corresponding to anaphoric relations. The idea here is similar to proposals in DRT for using a single DRS to represent information presented by a discourse or the shared commitments in a dialogue.

Suppose that we have a meaning function $m = \lambda r : T_1(T_2(r))$. The type T_1 represent constraints on the context that are required and $T_2(r)$ represents the content of the utterance given a certain context r . T_1 corresponds to what is normally called the *presuppositions* of an utterance. Presuppositions, i.e. backgrounded or assumed information, are associated with speech acts but when the commitments of an utterance are stored in memory there should be no distinction between backgrounded and foregrounded information since this is a matter of presentation.¹⁹ The formal technique we have for computing the “flattened” information to be stored is that of computing fixed point types. Thus what we use to refine beliefs or commitments is the fixed point type of m (or possibly m with some relabelling in order to ensure that labels do not clash), in symbols $\mathcal{F}(m)$. The basic operation for refinement of a record type with another record type is union of the two record types. This view of accumulating information in a dialogue by adding record types together is closely related to the use of type theoretical contexts to capture dynamic aspects of meaning in works such as Ahn (1995), Piwek (1998) and Ranta (1994) as well as the DRT literature.

Let us look at our example. *A* says *Sam says there’s a bug in the program*. The meaning of this is represented on p. 9. Assuming that prior to this utterance the commitments are the empty record type (i.e. there are no commitments) and that the utterance is accepted as a commitment, then both dialogue participants should update their information states with the fixed point type of the meaning, as follows:

$$\left[\begin{array}{c} \text{tis:} \\ \text{is:} \\ \text{shared:} \\ \text{com=} \end{array} \right] \left[\begin{array}{c} x:Ind \\ c_1:\text{named}(x, \text{“Sam”}) \\ y:Ind \\ \text{res:Rec} \\ c_2:\text{program}(y)\upharpoonright\text{res} \\ c_3:(r_1: \left[\begin{array}{c} z:Ind \\ c:\text{program}(z)\upharpoonright\text{res} \end{array} \right]) [c:\text{eq}(r_1.z, y, Ind)] \\ p=\lambda r_2 : \text{Rec} \left(\left[\begin{array}{c} x:Ind \\ c_4:\text{bug}(x, y) \end{array} \right] \right) : (\text{Rec})\text{RecType} \\ c_5:\text{say}(x, p) \end{array} \right] : \text{RecType}$$

Now *B* says *Pat is looking for it*. Assuming that this is accepted then both dialogue participants should update shared commitments with the fixed point type of the meaning on p. 9:

¹⁹I am grateful to Staffan Larsson for convincing me of the truth of this.

$$\dots \text{ com} = \left[\begin{array}{l} x:Ind \\ c_1:\text{named}(x, \text{"Sam"}) \\ y:Ind \\ \text{res}:Rec \\ c_2:\text{program}(y)|\text{res} \\ c_3:(r_1: \left[\begin{array}{l} z:Ind \\ c:\text{program}(z)|\text{res} \end{array} \right]) [c:\text{eq}(r_1.z, y, Ind)] \\ p_1 = \lambda r_2 : Rec \left(\left[\begin{array}{l} x:Ind \\ c_4:\text{bug}(x, y) \end{array} \right] \right) : (Rec)RecType \\ c_5:\text{say}(x, p_1) \\ z:Ind \\ c_6:\text{named}(z, \text{"Pat"}) \\ p_2 = \lambda r : ? ([y=?:Ind]):(?)RecType \\ c_7:\text{seek}(z, p_2) \end{array} \right] : RecType$$

Note that the metavariables introduced by the interpretation of *it* now occur within the larger (underspecified) type representing commitments. Anaphora resolution can work within the context of this larger type and the result can be a case of intentional identity (see Cooper, forthcoming a):

$$\dots \text{ com} = \left[\begin{array}{l} x:Ind \\ c_1:\text{named}(x, \text{"Sam"}) \\ y:Ind \\ \text{res}:Rec \\ c_2:\text{program}(y)|\text{res} \\ c_3:(r_1: \left[\begin{array}{l} z:Ind \\ c:\text{program}(z)|\text{res} \end{array} \right]) [c:\text{eq}(r_1.z, y, Ind)] \\ p_1 = \lambda r_2 : Rec \left(\left[\begin{array}{l} x:Ind \\ c_4:\text{bug}(x, y) \end{array} \right] \right) : (Rec)RecType \\ c_5:\text{say}(x, p_1) \\ z:Ind \\ c_6:\text{named}(z, \text{"Pat"}) \\ p_2 = \lambda r : \mathcal{F}(p_1) ([y=r.x:Ind]):(\mathcal{F}(p_1))RecType \\ c_2:\text{seek}(z, p_2) \end{array} \right] : RecType$$

Exactly which anaphora resolution algorithm is used is a matter of choice and we will not discuss this here. We will, however, reflect on how an anaphora resolution algorithm should interact with dialogue management. Our simple suggestion so far has been that the underspecified meaning (with metavariables) should first be accepted and entered into shared commitments before it is resolved. Certainly, it is interesting that our framework allows this as a possibility and it may indeed be important that this is possible in cases where we accept contributions for the sake of moving the dialogue forward even though we have not been able to figure out what exactly the content of the contribution is. However, in the general case, it would seem more intuitive that dialogue participants try to resolve underspecified meanings *before* they get accepted as shared commitments. This becomes particularly clear in cases where the contribution is not accepted:

- A: Sam says there's a bug in the program.
- B: Pat is looking for it.
- A: Are you sure? Last I heard Pat was working on another project.

After *A*'s second turn the question of whether Pat is looking for "it" is under discussion and there is at this point no shared commitment to the claim. It seems that in order for this discussion to take place *it* must be resolved before *B*'s utterance gives rise to a shared commitment. While there probably are cases where underspecified meanings become commitments, they are possibly the exception rather than the rule, at least in cases involving pronoun resolution. The tools we have developed will allow us to treat both cases. In addition to doing the resolution within the commitments as we showed before we can refine the meaning of

Pat is looking for it as follows (assuming that r_{curr} is the current information state after A 's first utterance as specified on page 20)²⁰:

$$\lambda r_1 : \left[\begin{array}{l} \text{cntxt} : \left[\begin{array}{l} x : Ind \\ c_1 : \text{named}(x, \text{"Pat"}) \end{array} \right] \\ \text{com} : r_{curr}.tis.is.shared.com \end{array} \right] \\ \left(\left[\begin{array}{l} p = \lambda r_2 : \mathcal{F}(r_1.com.p_1) ([y=r_2.x:Ind]) : (\mathcal{F}(r_1.com.p_1))RecType \\ c_2 : seek(r_1.cntxt.x, p) \end{array} \right] \right)$$

We can make reference to the record type representing the commitments prior to the utterance of *Pat is looking for it* just as well from outside the record type as from the inside.

4 Conclusion

We have discussed a way of formulating information state updates in dialogue using tools from type theory. We have argued that information state update is to be seen as a way of reasoning about the type of a new information state on the basis of the presumed type of the current information state. Reasoning at the level of types, rather than the information states themselves, provides a kind of underspecification since there may be several information states which belong to a single type.

Reasoning about information state updates, rather than just doing compositional semantics, is a way of trying to make precise part of Barwise's claim that we should view language in terms of speech events and the information that an agent can extract from them. Our proposal can also be viewed as a kind of radical or extended dynamic semantics (cf Barwise, 1989, as well as dynamic approaches in DRT and the work of Groenendijk and Stokhof, 1991). This is interesting because the dynamic approach of DRT and Groenendijk and Stokhof and others seems firmly rooted in the formal language approach. However, by making the information states richer, allowing downdating (non-monotonic updates) and allowing the updated information state to depend to a greater extent on the previous information state of the agent we move closer to Barwise's Austinian view.

However, we have also shown that compositional semantics plays a role in these updates. The kind of meaning we use is related to Barwise's work on situation semantics both in its approach to intensionality and in its treatment of context dependence (see Cooper, forthcoming a,b). The objects we call meanings are static rather than dynamic meanings, functions from contexts to contents following the Montague-Kaplan approach to meaning. However, meaning functions are modelled as functions from records (modelling contexts) to record types (modelling contents). This means that they are the same kind of objects as update rules which are functions from records (modelling information states) to record types (modelling types of information states). We have shown how this similarity of static meanings and update rules can be exploited so that static meanings can be used to reason about information state updates.²¹ While the present discussion has been rather technical, we feel that it points to the possibility of a powerful theory of language which unifies the speech-event based approach to language that Barwise advocated with the formal language approach which has enjoyed a great deal of success in linguistic semantics.

²⁰Compare this with the underspecified meaning on page 9.

²¹For an earlier discussion of a similar intuition see Cooper (2000).

References

- Ahn, René (1995) Communicating Contexts: A Pragmatic Approach to Information Exchanges, in Dybjer, Nordström and Smith (1995).
- Barwise, Jon. (1989), *The Situation in Logic*, CSLI Publications, Stanford.
- Barwise, Jon and John Perry (1983) *Situations and Attitudes*, MIT Press.
- Beaver, David (2004) The Optimization of Discourse Anaphora, *Linguistics and Philosophy*, 27(1), pp. 3–56.
- van Benthem, Johan and Alice ter Meulen, eds., (1997) *Handbook of Logic and Language*, North Holland, Amsterdam and MIT Press, Cambridge, Mass.
- Bos, Johan, Ewan Klein, Oliver Lemon, Tetsushi Oka (2003) DIPPER: Description and Formalisation of an Information-State Update Dialogue System Architecture, *Proceedings of 4th SIGdial Workshop on Discourse and Dialogue*, Sapporo, Japan.
- Cooper, Robin (2000) Information States, Attitudes and Dependent Record Types, in *Logic, Language and Computation, Volume 3*, ed. by Lawrence Cavedon, Patrick Blackburn, Nick Braisby, and Atsushi Shimojima, CSLI Publications, pp. 85–106
- Cooper, Robin (forthcoming a) Austinian truth, attitudes and type theory, in *Research on Language and Computation*.
- Cooper, Robin (forthcoming b) Records and record types in semantic theory, in *Journal of Logic and Computation*.
- Coquand, Thierry, Randy Pollack and Makoto Takeyama (2004) A Logical Framework with Dependently Typed Records, *Fundamenta Informaticae*, XX, pp. 1–22.
- Dybjer, Peter, Bengt Nordström and Jan Smith, eds., (1995) *Types for Proofs and Programs: selected papers*, Springer Verlag, Berlin.
- van Eijck, Jan and Hans Kamp (1997) Representing Discourse in Context, in van Benthem and ter Meulen (1997).
- Fernández, Raquel (2003) A Dynamic Logic Formalisation of the Dialogue Gameboard, in *Proceedings of the 10th Conference of the European Chapter of the ACL*, Student Research Workshop, pp. 17–24, Budapest, Hungary.
- Gabbay, Dov and Franz Guenther, eds (1986) *Handbook of Philosophical Logic, Vol. III*, Reidel, Dordrecht.
- Ginzburg, Jonathan (1996a) Dynamics and the Semantics of Dialogue, in Seligman and Westerståhl (1996).
- Ginzburg, Jonathan (1996b) Interrogatives: Questions, Facts and Dialogue in Lappin (1996).
- Groenendijk, Jeroen and Martin Stokhof (1991) Dynamic Predicate Logic, *Linguistics and Philosophy*, Vol. 14, pp. 39–100.
- Kamp, Hans and Uwe Reyle (1993) *From Discourse to Logic*, Kluwer, Dordrecht.
- Kaplan, David (1989) Demonstratives: An essay on the semantics, logic, metaphysics, and epistemology of demonstratives and other indexicals, in J. Almog, J. Perry, and H. K. Wettstein, editors, *Themes from Kaplan*, Oxford University Press, Oxford.
- Karttunen, Lauri (1969) Pronouns and Variables, *Papers from the Fifth Regional Meeting of the Chicago Linguistic Society*, ed. by R.I. Binnick, A. Davison, G.M. Green and J.L. Morgan, Department of Linguistics, University of Chicago, Chicago, Illinois, pp. 108–115.

- Kohlhase, Michael, Susanna Kuschert and Manfred Pinkal (1996) A type-theoretic semantics for λ -DRT, in *Proceedings of the 10th Amsterdam Colloquium*, ed. by P. Dekker and M. Stokhof, ILLC, Amsterdam, pp. 479–498.
- Lappin, Shalom, ed., (1996) *The Handbook of Contemporary Semantic Theory*, Blackwell, Oxford.
- Larsson, Staffan (2002) *Issue-based Dialogue Management*, PhD thesis, Göteborg University.
- Magnusson, Lena (1996) An algorithm for checking incomplete proof objects in type theory with localization and unification, in *Types for Proofs and Programs, Lecture Notes in Computer Science, 1158*, Springer Verlag, pp. 183–200.
- Magnusson, Lena and Bengt Nordström (1994) The ALF proof editor and its proof engine, in *Types for Proofs and Programs, Lecture Notes in Computer Science, 806*, ed. by H. Barendregt and T. Nipkow, Nijmegen, Springer Verlag, pp 213–237
- Montague, Richard (1974) *Formal Philosophy: Selected Papers of Richard Montague*, ed. and with an introduction by Richmond H. Thomason, Yale University Press, New Haven.
- Muskens, Reinhard (1995) Combining Montague semantics and discourse representation, *Linguistics and Philosophy*, Vol. 19, pp. 143–186.
- Piwek, P. (1998) *Logic, Information and Conversation*, PhD Thesis, Eindhoven University of Technology.
- Ranta, Aarne (1994) *Type-Theoretical Grammar*, Clarendon Press, Oxford.
- Sag, Ivan, Thomas Wasow and Emily Bender (2003) *Syntactic Theory*, Second Edition, CSLI Publications, Stanford.
- Searle, J.R. (1969) *Speech Acts, An Essay in the Philosophy of Language*, Cambridge: Cambridge University Press.
- Seligman, Jerry and Larry Moss (1997) Situation Theory, in van Benthem and ter Meulen (1997)
- Seligman, Jerry and Dag Westerståhl, eds (1996) *Logic, Language and Computation, Vol. 1*, CSLI Publications, Stanford.
- Sundholm, Göran (1986) Proof Theory and Meaning, in Gabbay and Guentner (1986).
- Veltman, Frank (1996) Defaults in update semantics, *Journal of Philosophical Logic*, Vol. 25, pp. 221–261.