Towards KoS/TTR-based proof-theoretic dialogue management

Vladislav Maraev[1], **Jonathan Ginzburg**[2], **Staffan Larsson**[1],
Ye Tian[3,2] **and Jean-Philippe Bernardy**[1]

[1]Centre for Linguistic Theory and Studies in Probability (CLASP),
Department of Philosophy, Linguistics and Theory of Science, University of Gothenburg
{vladislav.maraev,staffan.larsson,jean-philippe.bernardy}@gu.se
[2]Laboratoire Linguistique Formelle (UMR 7110), Université Paris Diderot
yonatan.ginzburg@univ-paris-diderot.fr
[3]Amazon Research Cambridge
tiany.03@gmail.com

Abstract

This paper presents the first attempt to implement a dialogue manager based on the KoS framework for dialogue context and interaction. We utilise our own proof-theoretic implementation of Type Theory with Records (TTR) and implement a basic dialogue that involves mutual greeting. We emphasize the importance of findings in dialogue theory for designing dialogue systems which we illustrate by sketching an account for question-answer relevance.

1 Introduction

One of the most challenging tasks in the design of dialogue systems concerns their capability to support dialogue strategies that are similar to ones that happen in a dialogue between human participants. The key component of a dialogue system in this aspect is the dialogue manager, which selects appropriate system actions depending on the current state and the external context.

Two families of approaches to dialogue management can be considered: hand-crafted dialogue strategies (Allen et al., 1995; Larsson, 2002; Jokinen, 2009) and statistical modelling of dialogue (Rieser and Lemon, 2011; Young et al., 2010; Williams et al., 2017; Eshghi et al., 2017). Hand-crafted strategies range from finite-state machines and slot-filling to more complex dialogue planning and logical inference rules. Statistical models help to contend with the uncertainty that arises from noisy signals that arise from speech recognition and other sensors.

Although there has been a lot of development in dialogue systems in recent years, only a few approaches to *dialogue management* (Allen et al., 1995; Poesio and Traum, 1997; Larsson and Traum, 2000; Larsson, 2002) reflect advancements in *dialogue theory* (Ginzburg, 1996; Asher and Lascarides, 2003), and there has not been much progress in this respect since the early 2000s. Our aim is to closely integrate dialogue systems with work in theoretical semantics/pragmatics of dialogue which allows creating more human-like conversational agents. Here we illustrate this by exemplifying a rudimentary but potentially deep theory of answers which will be extended further in order to support phenomena discussed in Bos and Gabsdil (2000).

KoS (not an acronym but loosely corresponds to Conversation Oriented Semantics) (Ginzburg, 2012) provides among the most detailed theoretical treatments of domain general conversational relevance, especially for query responses—see Purver (2006) on Clarification Requests, (Łupkowski and Ginzburg, 2017) for a general account— and this ties into the KoS treatment of non sentential utterances, again a domain crucial for naturalistic dialogue systems and where KoS has among the most detailed analyses (Fernández et al., 2007; Ginzburg, 2012).

KoS is based on the formalism of Type Theory with Records (TTR). There has been a wide range of work in this formalism which includes the modelling of intentionality and mental attitudes (Cooper, 2005), generalised quantifiers (Cooper, 2013), co-predication and dot types in lexical innovation, frame semantics for temporal reasoning, reasoning in hypothetical contexts (Cooper, 2011), spatial reasoning (Dobnik and Cooper, 2017), enthymematic reasoning (Breitholtz, 2014), clarification requests (Purver,

2006; Ginzburg, 2012), negation (Cooper and Ginzburg, 2012), non-sentential utterance resolution (Fernández et al., 2007; Ginzburg, 2012) and iconic gesture (Lücking, 2016).

In the rest of the paper we briefly survey the basic features of KoS and TTR (section 2), describe our implementation (section 3) and a minimal working example of rules for a dialogue system (section 4). We illustrate this by an initial sketch of a theory of answers (section 5). We conclude with some brief discussion and pointers to future work.

2 A brief account of KoS and TTR

KoS (Ginzburg, 2012) is a formal semantic framework based on Type Theory with Records (TTR), oriented at dialogue, capturing the features of conversational *interaction*. In KoS (and other dynamic approaches to meaning), language is compared to a game, containing players (interlocutors), goals and rules. KoS represents language interaction by representing the dynamically changing context. The meaning of an utterance is how it changes the context. Compared to most formal semantics approaches (e.g. Roberts (2012), which represent a single context for both dialogue participants), KoS maintains a separate representation for each participant, using the *Dialogue Game Board* (DGB). DGBs represent the information states of the participants, which comprise a private part and the dialogue gameboard that represents information arising from publicized interactions. This tracks, at the very least, shared assumptions/visual space, moves (= utterances, form and content), and questions under discussion.

In TTR agents perceive an individual object that exists in the world in terms of being *of a particular type*. Such basic judgements performed by agents can be denoted as "a : Ind", meaning that a is an individual, in other words a is a *witness* of (the type) Ind(ividual). This is an example of a *basic* type in TTR, namely types that are not constructed from other types. An example of a more complex type in TTR is a *ptype* which is constructed from predicates, e.g. $greet(a, b)$, "a greets b". A witness of such a type can be a situation, a state or an event. To represent a more general event, such as "one individual greets another individual" *record types* are used. Record types consist of a set of fields, which are pairs of unique labels and types. The record type which will correspond to the aforementioned sentence is the following:

$$(1) \quad \begin{bmatrix} \text{x} & : & \text{Ind} \\ \text{y} & : & \text{Ind} \\ \text{c} & : & \text{greet(x,y)} \end{bmatrix}$$

The witnesses of record types are *records*, consisting of a set of fields which are pairs of unique labels and values. In order to be of a certain record type, a record must contain at least the same set of labels as the record type, and the values must be of a type mentioned in the corresponding field of the record type. The record may contain additional fields with labels not mentioned in the record type. For example, the record (2) is of a type in (1) iff a : Ind, b : Ind, s : greet(a, b) and q is of an arbitrary type.

$$(2) \quad \begin{bmatrix} \text{x} & = & a \\ \text{y} & = & b \\ \text{c} & = & s \\ \text{p} & = & q \end{bmatrix}$$

In our Dialogue Manager, a state is represented as a pair of a type S and an object s witnessing it. For example, if S is a record type containing greet(a, b), then s will contain an event witnessing the greeting. These abstract types and witnesses can be mapped to utterances using NLU and NLG.

TTR also defines a number of type construction operations. Here we mention only the ones that are used in the current paper:

1. *List types*: if T is a type, then $[T]$ is also a type – the type of lists each of whose members is of type T. The list $[a_1, \ldots, a_n] : [T]$ iff for all i, $a_i : T$. Additionally, we use a type of non-empty lists, written as ${}_{ne}[T]$, which is a subtype of $[T]$ where $1 \leq i \leq n$. We assume the following operations on lists: constructing a new list from an element and a list (cons), taking the first element of list

(head), taking the rest of the list (tail).

$$\begin{aligned} \text{cons}: & \; T \to [T] \to {}_{ne}[T] \\ \text{head}: & \; {}_{ne}[T] \to T \\ \text{tail}: & \; {}_{ne}[T] \to [T] \end{aligned}$$

2. *Function types*: if T_1 and T_2 are types, then so is $(T_1 \to T_2)$, the type of total functions from elements of type T_1 to elements of type T_2. Additionally, T_2 may *depend* on the parameter (the witness of type T_1 passed to the function).

3. *Meet types*: if T_1 and T_2 are types, then $T_1 \wedge T_2$ is also a type. $a : T_1 \wedge T_2$ iff $a : T_1$ and $a : T_2$.

4. *Singleton types*: if T is a type and x:T, then T_x is a type. a:T_x iff $a = x$. In record types we use manifest field notation to a represent singleton type. Notations $\left[\,\mathrm{a} \;:\; \mathrm{T}_x\,\right]$ and $\left[\,\mathrm{a}{=}x \;:\; \mathrm{T}\,\right]$ represent the same object.

3 Implementation

Our dialogue manager (DM) is based on a new implementation of Cooper's TTR (Cooper, in prep). The important parts of this implementation are: a type-checker, a subtype checker and a rule-application mechanism. Figure 1 shows such a dialogue manager integrated into a spoken dialogue system.

The type-checker's implementation follows the structure of MiniTT (Coquand et al., 2009). However the type system itself closely follows that described by Cooper. The significant differences are:

1. A more flexible behaviour for meet types: when applied to record types the meet operator reduces to another record type if possible. For example, $[f : A] \wedge [f : B, g : C]$ reduces to $[f : A \wedge B, g : C]$. This change means that meet behaves as the merge ($\wedge\!\!\!\cdot$) operator in Cooper's work.

2. Support for boolean types (true : *Bool*) and (false : *Bool*), as well as conditionals, such that "(IF true THEN x ELSE y) = x" and "(IF false THEN x ELSE y) = y".
With boolean types and records we can construct the type $A \sqcup B$, which is the *disjoint union* of the arbitrary types A and B. It is defined as:

$$(3) \quad A \sqcup B =_{def} \begin{bmatrix} \text{choice} & : & \text{Bool} \\ \text{result} & : & \text{IF choice THEN } A \text{ ELSE } B \end{bmatrix}$$

The rule-application mechanism is implemented as a thin layer over the typechecker and subtyping algorithm. The behaviour of the DM is implemented as a set of rules (see below), which are parsed, type-checked and evaluated to normal forms[1]. Then, at runtime, the dialogue manager maintains its state (*dialogue state*) as a pair of a value (s) and a type (S), such that $s : S$[2]. A rule r can be applied iff its type is a function type whose domain is a supertype of S_s. Formally, the applicability condition is $r : A \to B$ and $S_s \sqsubseteq A$. After an application of the rule r, the dialogue manager state becomes the pair $(r(s), B)$. At any point, several rules may apply. There are several possible rule-selection strategies. Useful strategies include backtracking search and user-defined selection.

4 TTR account for a dialogue system: a minimal example

As a starting point we define a basic set of rules that supports a very basic interaction (4) between an agent (A) and a user (U).

(4) U: hello
A: Hello world!

[1] by applying beta reduction, field extraction and the if-then-else rules shown above.
[2] And, additionally, $s : S_s$, by definition of singleton types.

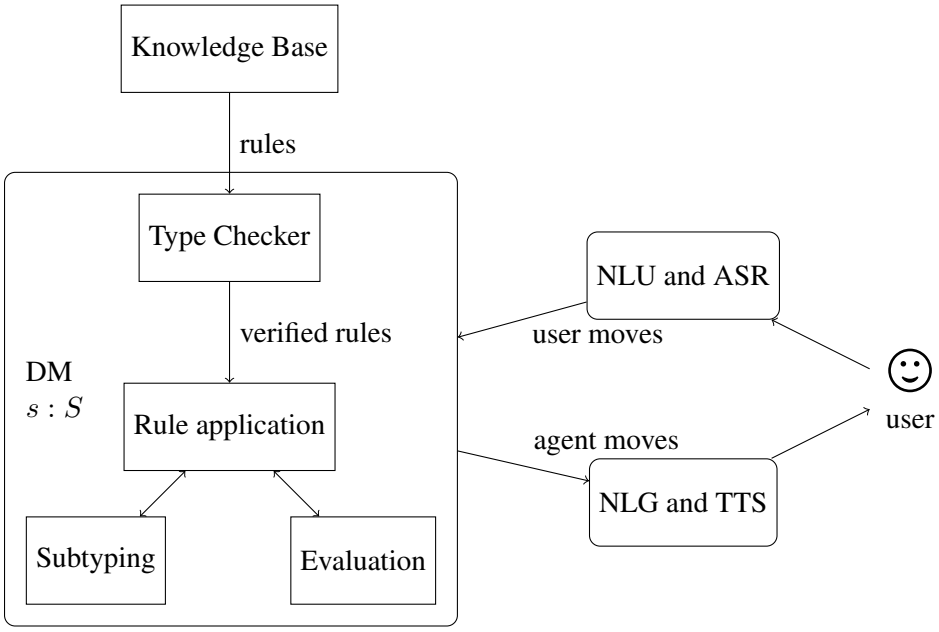

Figure 1: Architecture of a spoken dialogue system with a proof-theoretic dialogue manager.

Primary KoS types For the current purposes, we do not consider the user's information state; we manipulate solely the the agent's information state. We implement a minimal version of an *agent's information state* (5) consisting of a private part (a list of moves to be emitted) and a public part—the dialogue gameboard (DGB). In future work the DGB will be extended to support turn taking, questions under discussion, facts and other notions defined in (Ginzburg, 2012).

$$\text{(5)}\quad \text{InformationState} =_{def} \begin{bmatrix} \text{private} & : & \begin{bmatrix} \text{agenda} & : & [\text{Move}] \end{bmatrix} \\ \text{dgb} & : & \begin{bmatrix} \text{moves} & : & [\text{Move}] \\ \text{latestMove} & : & \text{Move} \end{bmatrix} \end{bmatrix}$$

By *Move* we mean a type albeit akin to Ginzburg's definition of illocutionary proposition:

$$\text{(6)}\quad \text{Move} =_{def} \begin{bmatrix} \text{spkr} & : & \text{Ind} \\ \text{addr} & : & \text{Ind} \\ \text{content} & : & \text{MoveContent} \end{bmatrix},$$

where *MoveContent* is a record type containing a proposition; for greeting it will correspond to $\left[\text{c:greet(spkr,addr)}\right]$ composing the record type (7) either produced by agent or by user.

$$\text{(7)}\quad \text{GreetingMove} =_{def} \begin{bmatrix} \text{spkr} & : & \text{Ind} \\ \text{addr} & : & \text{Ind} \\ \text{content} & : & \left[\text{c:greet(spkr,addr)}\right] \end{bmatrix}$$

Initial Dialogue State In order to implement an initial dialogue state we initialise the dialogue state to be the record (8) of a type *InformationState*, where $\emptyset$ is an initial dummy move.

$$\text{(8)}\quad \text{init} =_{def} \begin{bmatrix} \text{private} & = & \begin{bmatrix} \text{agenda} & = & [] \end{bmatrix} \\ \text{dgb} & = & \begin{bmatrix} \text{moves} & = & [] \\ \text{latestMove} & = & \emptyset \end{bmatrix} \end{bmatrix}$$

Conversational rules As a means of describing general, cross-domain patterns of conversational interaction *conversational rules* are provided in the form of functions that manipulate the dialogue state. One might expect that they would have the type (InformationState→InformationState). However, some rules will take as input (or provide as output) *subtypes* of InformationState. We define two basic rules: for the agent's reaction to the user's greeting[3] *counterGreeting* (9) is used and *fulfilAgenda* (10)—the

[3]For simplicity we restrict this rule to the case when only the agent can perform countergreeting.

rule pops information from the agenda (moves that have taken place) and puts it on the DGB. This set of rules will be extended to support other dialogue phenomena, such as turn-taking[4], adjacency pairs, queries, assertions etc. Domain-dependent dialogue strategies will be supported in a similar fashion.

(9) counterGreeting : InformationState

$$\wedge \left[\begin{array}{l}\text{dgb} : \left[\begin{array}{l}\text{latestMove} : \left[\begin{array}{lll}\text{spkr=user0} & : & \text{Ind}\\ \text{addr=agent} & : & \text{Ind}\\ \text{content} & : & \left[\text{c} : \text{greet(spkr,addr)}\right]\end{array}\right]\end{array}\right]\end{array}\right]$$

$$\rightarrow \text{InformationState} \wedge \left[\text{private} : \left[\text{agenda} : {}_{ne}[\text{Move}]\right]\right]$$

$$\text{counterGreeting} =_{def} \lambda s.$$

$$\left[\begin{array}{lll}\text{private} & = & \left[\begin{array}{lll}\text{agenda} & = & \text{cons}(\left[\begin{array}{lll}\text{spkr} & = & \text{agent}\\ \text{addr} & = & \text{user0}\\ \text{content} & = & \left[\text{c=gs(agent,user0)}\right]\end{array}\right], s.\text{private.agenda})\end{array}\right]\\ \text{dgb} & = & s.\text{dgb}\end{array}\right],$$

where gs(agent,user0) : greet(spkr,addr) is a greeting situation.

(10) fulfilAgenda : InformationState $\wedge$ [private:[agenda : ${}_{ne}$[Move]]] $\rightarrow$ InformationState

$$\text{fulfilAgenda} = \lambda s. \left[\begin{array}{lll}\text{private} & = & \left[\text{agenda} = \text{tail}(s.\text{private.agenda})\right]\\ \text{dgb} & = & \left[\begin{array}{lll}\text{latestMove} & = & \text{head}(s.\text{private.agenda})\\ \text{moves} & = & \text{cons}(s.\text{private.agenda}, s.\text{dgb.moves})\end{array}\right]\end{array}\right]$$

NLU and NLG In order to integrate the user's move—a result of natural language understanding—the rule (11) is defined. The move for natural language generation is selected automatically in the case of having non-empty agenda.

(11) integrateUserMove : Move $\rightarrow$ InformationState $\rightarrow$ InformationState

$$\text{integrateUserMove} = \lambda m.\lambda s.$$

$$\left[\begin{array}{lll}\text{private} & = & s.\text{private}\\ \text{dgb} & = & \left[\begin{array}{lll}\text{latestMove} & = & m\\ \text{moves} & = & \text{cons}(m, s.\text{dgb.moves})\end{array}\right]\end{array}\right]$$

Greeting example In Appendix A we present an example of applying the update rules in order to establish the basic greeting exchange (4).

5 Primary treatment of question-answer relevance

5.1 Questions

We provide a general definition of *question*, as a way to establish a connection between a possible answer and its expected meaning in a given context:

(12) Question : Type

$$\text{Question} =_{def} \left[\begin{array}{lll}\text{A} & : & \text{Type}\\ \text{Q} & : & \text{A} \rightarrow \text{Prop}\end{array}\right],$$

where the field A corresponds to the expected type of an answer and the field Q is a family of propositions, such that for any answer a, Q(a) is the meaning of answer a as a proposition. In other words, Q is the family of expected answers, as propositions.

We can define subtypes for polar and wh- questions as follows:

(13) $$\text{PolarQuestion} =_{def} \left[\begin{array}{lll}\text{A=Bool} & : & \text{Type}\\ \text{Q} & : & \text{A} \rightarrow \text{Prop}\end{array}\right]$$

(14) $$\text{UnaryWhQuestion} =_{def} \left[\begin{array}{lll}\text{A=Ind} & : & \text{Type}\\ \text{Q} & : & \text{A} \rightarrow \text{Prop}\end{array}\right]$$

[4]Procedural coordination can be established in KoS via rules for turn assignment. We thank an anonymous reviewer for SEMDIAL for raising this issue.

We can illustrate the semantic interpretation of polar questions (15) and unary wh-questions (16) as follows[5]:

(15) $[\![$"Do you live in Paris?"$]\!] = \begin{bmatrix} \text{A} & = & \text{Bool} \\ \text{Q} & = & \lambda a.\ \text{IF } a \text{ THEN live(Paris) ELSE } \neg\text{live(Paris)} \end{bmatrix}$

(16) $[\![$"Where do you live?"$]\!] = \begin{bmatrix} \text{A} & = & \text{City} \\ \text{Q} & = & \lambda a.\text{live}(a) \end{bmatrix}$

5.2 Answers

For every question q : $\begin{bmatrix} \text{A} & : & \text{Type} \\ \text{Q} & : & \text{A} \rightarrow \text{Prop} \end{bmatrix}$, we construct a type of answers that fully resolve the posed question:

(17) Answer : Question $\rightarrow$ Type

$$\text{Answer} =_{def} \lambda q. \begin{bmatrix} \text{answer} & : & q.\text{A} \\ \text{sit} & : & q.\text{Q(answer)} \end{bmatrix},$$

where the first field is an answer of the type presumed by the question q and the second field represents the situation where the answer to the question holds, or in general a witness that the answer is correct. In type theory, this witness is necessary to consider the proposition associated with the answer as true.[6]

Continuing the examples (15, 16) above, we can see how possible answers (20, 21) can be interpreted in the context of the corresponding questions . First, we compute the type of answers:

(18) Answer($[\![$"Do you live in Paris?"$]\!]$)

$$= \begin{bmatrix} \text{answer} & : & \text{Bool} \\ \text{sit} & : & \text{IF answer THEN live(Paris) ELSE } \neg\text{live(Paris)} \end{bmatrix}$$

(19) Answer($[\![$"Where do you live?"$]\!]$)

$$= \begin{bmatrix} \text{answer} & : & \text{City} \\ \text{sit} & : & \text{live(answer)} \end{bmatrix}$$

Then we see that suitable answers have the appropriate type:

(20) $[\![$"yes"$]\!]$: (q : PolarQuestion) $\rightarrow$ Answer(q)

$$[\![\text{"yes"}]\!] = \lambda q. \begin{bmatrix} \text{answer} & = & \text{true} \\ \text{sit} & = & \text{s}_{lp} \end{bmatrix}$$

(21) $[\![$"in Paris"$]\!]$: (q : UnaryWhQuestion) $\rightarrow$ Answer(q)

$$[\![\text{"in Paris"}]\!] = \lambda q. \begin{bmatrix} \text{answer} & = & \text{Paris} \\ \text{sit} & = & \text{s}_{lp} \end{bmatrix}$$

where s_{lp} is such a situation where user lives in Paris.

5.3 Interpreting answers in form of propositions

Not all answers are provided as a simple element of the requested types. Instead, an utterance can take the form of a declarative sentence which can be interpreted as a proposition $(P : \text{Prop})$ and a witness $(p : P)$. We now describe a heuristic procedure which can be used to check if such an utterance can be interpreted as an answer to a given question $(q : \text{Question})$, and if so, how.

1. Unify $q.Q(a)$ with P, where a is a fresh metavariable. If unification succeeds, it will yield a substitution σ, such that $q.Q(\sigma(a)) = P$.

[5]Following the simplification made in Larsson (2002) we are using reduced semantic representations, e.g., **live(Paris)** instead of $\begin{bmatrix} \text{x=user0} \\ \text{c=live(Paris,x)} \end{bmatrix}$.

[6]In a dialogue system the user will in general be trusted, and so the witnesses will only consist of a representation of the users' utterances in context. This could be represented formally by making the situation depend on the agent's context: $s_{lp}(ctxt)$. Conversely, when the system replies to the user, requiring a witness means that the system must be able to justify its answer using facts from a knowledge base or a proof constructed from those.

2. Construct an answer as $\begin{bmatrix} \text{answer} & = & \sigma(a) \\ \text{sit} & = & p \end{bmatrix}$: Answer (q). Indeed the record fields have the expected types: i) $\sigma(a) : q.\text{A}$ because a occurs as an argument to $q.\text{Q}$, and ii) $p : q.\text{Q}(\sigma(a))$ because $p : P$ and $P = q.\text{Q}(\sigma(a))$.

For example, assume $[\![\text{“I live in Paris”}]\!] = \begin{bmatrix} \text{P} & = & \text{live(Paris)} \\ \text{p} & = & \text{s}_{lp} \end{bmatrix}$ and q as in (16). We thus unify live(Paris) with $q.\text{Q}(a) = \text{live}(a)$ and find $\sigma(a) = \text{Paris}$. The answer is then $\begin{bmatrix} \text{answer} & = & \text{Paris} \\ \text{sit} & = & \text{s}_{lp} \end{bmatrix}$.

5.4 Partial resolution of questions

In any question context, an utterance can either *be unrelated* to the question at hand, *fully resolve* the question or *partially resolve* it. Thus, in a spoken dialogue system, one should have a procedure to classify utterances in this way.

(22) questionResolutionClassifier : Utterance $\rightarrow$ (q : Question)
$\rightarrow$ UnrelatedUtterance $\sqcup$ ResolvingAnswer(q) $\sqcup$ PartiallyResolvingAnswer(q)

The implementation of such a classifier may use the procedure described in the above section — we will not discuss it further here and just assume that its output is available. Resolving answers were discussed above in section 5.2, and further interpretation of unrelated utterances is out of the scope of this paper. In the rest of the section we propose a treatment for partially resolving answers.

(23) $\text{ResolvingAnswer}(q) =_{def} \text{Answer}(q) = \lambda q. \begin{bmatrix} \text{answer} & : & q.\text{A} \\ \text{sit} & : & q.\text{Q(answer)} \end{bmatrix}$

(24) $\text{PartiallyResolvingAnswer}(q) =_{def} \begin{bmatrix} \text{q}_{rem} & : & \text{Question} \\ \text{resolution} & : & \text{Answer}(\text{q}_{rem}) \rightarrow \text{Answer}(q) \end{bmatrix}$

That is, a partial answer is understood as a pair of i) the question that remains (q_{rem}) and ii) a *resolution*, which provides a way to fully resolve the initial question from the answer to q_{rem}.

We illustrate the partial resolution of a question with an example from a prototypical goal-oriented dialogue system that operates incrementally, on input that is smaller than utterances (Schlangen and Skantze, 2009):

(25) A: What do you want today?
U: A beer, please, and chips.

We assume that q_1 has a domain-specific interpretation.

(26) $q_1 = [\![\text{“What do you want today?”}]\!] = \begin{bmatrix} \text{A} & = & \begin{bmatrix} \text{food} & : & \text{Food} \\ \text{drink} & : & \text{Drink} \end{bmatrix} \\ \text{Q} & = & \lambda a.\text{order}(a.\text{food}, a.\text{drink}) \end{bmatrix}$

(27) $a_1 : \text{PartialAnswer}(q_1)$

$$a_1 = [\![\text{“A beer please”}]\!] = \begin{bmatrix} \text{q}_{rem} & = & \begin{bmatrix} \text{A} & = & \text{Food} \\ \text{Q} & = & \lambda a.\text{order}(a, \text{beer}) \end{bmatrix} \\ \text{resolution} & = & \lambda a_{rem}. \begin{bmatrix} \text{answer} & = & \begin{bmatrix} \text{food} & = & a_{rem}.\text{answer} \\ \text{drink} & = & \text{beer} \end{bmatrix} \\ \text{sit} & = & a_{rem}.\text{sit} \end{bmatrix} \end{bmatrix}$$

We can interpret the remaining implicit question $a_1.\text{q}_{rem}$ as something similar to “Would you like any food with your beer?”.

(28) $a_2 = [\![\text{“and chips”}]\!] = \begin{bmatrix} \text{answer} & = & \text{chips} \\ \text{sit} & = & \text{s}_{b\&c} \end{bmatrix}$, where $\text{s}_{b\&c}$ is a situation when customer wants beer and chips.

We can see that (28) is an answer that fully resolves $a_1.\mathrm{q}_{rem}$ and thereby q_1.

We are aware of the existence of situations when a_1 might fully resolve q_1. Handling this would require a notion of planning and question resolution according to the plan (Larsson, 2002). This issue will be addressed in future work.

6 Discussion: update rule output as objects or types

The formalisation of information state update proposed here is slightly different from previous work. In (Cooper, in prep), update rules are functions of the form $f = \lambda r : A.B(r)$ with type $f : A \rightarrow Type$. An issue with this formulation is that the output of a rule is a type (S'), while rules take as input objects (s). Therefore, after application of any rule, an object s' of type S' needs to be constructed, to be used as input to the next rule. If the type S' is a fully specified type (i.e., it is a singleton type or a record type whose components are fully specified), this computation is possible because an object of a fully specified type can be constructed as record s' with the same fields as the record type and with value a for each fully specified type T in S'. If the output type is not fully specified, however, so-called "hypothetical objects" will need to be constructed corresponding to the non-singleton types in S'. In such a case, the type S' is not guaranteed to have a witness — it is even possible that S' is the empty type, leading to logical inconsistency.

In this paper, rules have the form of well-typed functions. For example a rule f may be $f = \lambda r.b(r)$ where $r : A, b(r) : B$ with $f : A \rightarrow B$. The difference from Cooper's work is that the rules in this paper (1) specify type constraints in the rule types and (2) output records (generally: objects) rather than record types (generally: types). One reason for doing things this way is that the rules are applied to records, and if they also output records, then a sequence of updates can be seen as a simple threading of update rules where the output of one rule is the input to the next. The potential disadvantage with rules producing objects as output is that underspecified information states are more difficult to deal with.

7 Conclusions and future work

We hope that the proposed approach to dialogue management will enable one to bring significant advances from dialogue theory into the state-of-the-art of dialogue system development and design. It is important to support important principles of interaction domain-independently, however our approach does not constrain creation of domain-specific dialogue rules and strategies.

We are aiming at developing a hybrid system which: (a) maintains a rich information state, (b) has sets of domain-independent and domain-dependent conversational rules and (c) will allow the assignment of probabilities to rules and to the components of the information state and to train the probabilities according to the new observations. In this sense our approach follows (Lison, 2015), which is based on probabilistic rules.

We intend to develop a fully fledged spoken dialogue system on this basis that will enable it to support theoretical notions similar to the ones developed in frameworks like KoS. Creating such an implemented account of theoretical dialogue frameworks will enable researchers to test theories of dialogue and discourse and exhibit the results of their research to a broader public.

Acknowledgements

This research was supported by a grant from the Swedish Research Council for the establishment of the Centre for Linguistic Theory and Studies in Probability (CLASP) at the University of Gothenburg. We also acknowledge the support of the French Investissements d'Avenir-Labex EFL program (ANR-10-LABX-0083). In addition, we would like to thank Robin Cooper and our anonymous reviewers for their useful comments.

References

James F Allen, Lenhart K Schubert, George Ferguson, Peter Heeman, Chung Hee Hwang, Tsuneaki Kato, Marc Light, Nathaniel Martin, Bradford Miller, Massimo Poesio, et al. 1995. The TRAINS project: A case study in building a conversational planning agent. *Journal of Experimental & Theoretical Artificial Intelligence* 7(1):7–48.

Nicholas Asher and Alex Lascarides. 2003. *Logics of conversation*. Cambridge University Press.

Johan Bos and Malte Gabsdil. 2000. First-order inference and the interpretation of questions and answers. In Massimo Poesio and David Traum, editors, *Proceedings of the Götalog, the 4th Workshop on the Formal Semantics and Pragmatics of Dialogue*, Götaborg.

Ellen Breitholtz. 2014. Reasoning with topoi–towards a rhetorical approach to non-monotonicity. In *Proceedings of the 50th anniversary convention of the AISB, 1st–4th April 2014, Goldsmiths, University of London*.

Robin Cooper. 2005. Austinian truth, attitudes and type theory. *Research on Language and Computation* 3(4):333–362.

Robin Cooper. 2011. Copredication, quantification and frames. In Sylvain Pogodalla and Jean-Philippe Prost, editors, *Logical Aspects of Computational Linguistics (LACL 2011)*. Springer.

Robin Cooper. 2013. Clarification and generalized quantifiers. *Dialogue and Discourse* 4:125.

Robin Cooper. in prep. Type theory and language: From perception to linguistic communication. https://sites.google.com/site/typetheorywithrecords/drafts.

Robin Cooper and Jonathan Ginzburg. 2012. Negative inquisitiveness and alternatives-based negation. In *Logic, Language and Meaning*, Springer, pages 32–41.

Thierry Coquand, Yoshiki Kinoshita, Bengt Nordstrm, and Makoto Takeyama. 2009. A simple type-theoretic language: Mini-TT .

Simon Dobnik and Robin Cooper. 2017. Interfacing language, spatial perception and cognition in type theory with records. *Journal of Language Modelling* 5(2):273–301.

Arash Eshghi, Igor Shalyminov, and Oliver Lemon. 2017. Interactional dynamics and the emergence of language games. In *Proceedings of the ESSLLI 2017 workshop on Formal approaches to the Dynamics of Linguistic Interaction. Barcelona*.

Raquel Fernández, Jonathan Ginzburg, and Shalom Lappin. 2007. Classifying ellipsis in dialogue: A machine learning approach. *Computational Linguistics* 33(3):397–427.

Jonathan Ginzburg. 1996. Interrogatives: Questions, facts and dialogue. *The handbook of contemporary semantic theory. Blackwell, Oxford* pages 359–423.

Jonathan Ginzburg. 2012. *The interactive stance*. Oxford University Press.

Kristiina Jokinen. 2009. *Constructive dialogue modelling: Speech interaction and rational agents*, volume 10. John Wiley & Sons.

Staffan Larsson. 2002. *Issue-based dialogue management*. Department of Linguistics, Göteborg University.

Staffan Larsson and David R Traum. 2000. Information state and dialogue management in the TRINDI dialogue move engine toolkit. *Natural language engineering* 6(3-4):323–340.

Pierre Lison. 2015. A hybrid approach to dialogue management based on probabilistic rules. *Computer Speech & Language* 34(1):232–255.

Andy Lücking. 2016. Modeling co-verbal gesture perception in type theory with records. In *Proceedings of the 2016 Federated Conference on Computer Science and Information Systems*. pages 383–392.

Paweł Łupkowski and Jonathan Ginzburg. 2017. Query responses. *Journal of Language Modelling* 4(2):245–292.

Massimo Poesio and David R Traum. 1997. Conversational actions and discourse situations. *Computational intelligence* 13(3):309–347.

M. Purver. 2006. CLARIE: Handling clarification requests in a dialogue system. *Research on Language & Computation* 4(2):259–288.

Verena Rieser and Oliver Lemon. 2011. *Reinforcement learning for adaptive dialogue systems: a data-driven methodology for dialogue management and natural language generation*. Springer Science & Business Media.

Craige Roberts. 2012. Information structure: Towards an integrated formal theory of pragmatics. *Semantics and Pragmatics* 5:6–1.

David Schlangen and Gabriel Skantze. 2009. A general, abstract model of incremental dialogue processing. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, pages 710–718.

Jason D Williams, Kavosh Asadi, and Geoffrey Zweig. 2017. Hybrid code networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning. *arXiv preprint arXiv:1702.03274* .

Steve Young, Milica Gašić, Simon Keizer, François Mairesse, Jost Schatzmann, Blaise Thomson, and Kai Yu. 2010. The hidden information state model: A practical framework for POMDP-based spoken dialogue management. *Computer Speech & Language* 24(2):150–174.

A Supplemental Material: "Hello world!" example

Differences between the field values in s_{i-1} and s_i are marked with an asterisk (*).

1. $s_0 = \text{init} =$
$$\begin{bmatrix} \text{private} & = & [\text{agenda} & = & [\,]\,] \\ \text{dgb} & = & [\text{moves} & = & [\,]\,] \end{bmatrix}$$

2. `USER0> hello` is interpreted by NLU as a move $m_0 =$
$$\begin{bmatrix} \text{spkr} & = & \text{user0} \\ \text{addr} & = & \text{agent} \\ \text{content} & = & [\text{c=gs(spkr,addr)}] \end{bmatrix}$$

3. $s_1 = \text{integrateUserMove}(s_0, m_0) =$
$$\begin{bmatrix} \text{private} & = & [\text{agenda} & = & [\,]\,] \\ \text{dgb} & = & [\text{moves*} & = & [\,[\text{spkr} = \text{user0}, \text{addr} = \text{agent}, \text{content} = [\text{c=gs(user0,agent)}]]\,] \\ & & \text{latestMove*} & = & [\text{spkr} = \text{user0}, \text{addr} = \text{agent}, \text{content} = [\text{c=gs(user0,agent)}]] \end{bmatrix}$$

4. $s_2 = \text{counterGreeting}(s_1) =$
$$\begin{bmatrix} \text{private} & = & [\text{agenda*} & = & [\,[\text{spkr} = \text{agent}, \text{addr} = \text{user0}, \text{content} = [\text{c=gs(agent,user0)}]]\,] \\ \text{dgb} & = & [\text{moves} & = & [\,[\text{spkr} = \text{user0}, \text{addr} = \text{agent}, \text{content} = [\text{c=gs(user0,agent)}]]\,] \\ & & \text{latestMove} & = & [\text{spkr} = \text{user0}, \text{addr} = \text{agent}, \text{content} = [\text{c=gs(user0,agent)}]] \end{bmatrix}$$

5. State s_2 has non-empty agenda, thus agenda's content will be emitted and NLG will produce an utterance: `AGENT> Hello world!`.

6. $s_3 = \text{fulfilAgenda}(s_2) =$
$$= \begin{bmatrix} \text{private} & = & [\text{agenda*} & = & [\,]\,] \\ \text{dgb} & = & [\text{moves*} & = & [\,[\text{spkr} = \text{agent}, \text{addr} = \text{user0}, \text{content} = [\text{c=gs(agent,user0)}]], [\text{spkr} = \text{user0}, \text{addr} = \text{agent}, \text{content} = [\text{c=gs(user0,agent)}]]\,] \\ & & \text{latestMove*} & = & [\text{spkr} = \text{agent}, \text{addr} = \text{user0}, \text{content} = [\text{c=gs(agent,user0)}]] \end{bmatrix}$$