CoDyLan: Towards Doing (some) Inference for DyLan

Stergios Chatzikyriakidis*
CLASP, Department of Philosophy, Linguistics and Theory of Science
University of Gothenburg

November 7, 2018

1 Introduction

DyLan [6] is a parser based on the Dynamic Syntax framework (DS, [7, 2]) that is able, among other things, to cope with a number of problematic dialogue phenomena including self-repair (SR), ellipsis, short-answers (SA), clarification interaction (CR), corrections (COR), split-utterances (SU) and backchannels (ACK).

A: Who did you meet yesterday?
B: Arash [SA]
A: The guy from your group? [CR]
B: no, my cousin [COR]
A: right [ACC]. I think I have met him.
(1)

A: Yesterday, I finally cooked uhh
B: What? [CR,SU]
A: Stew, uhh, Beef Stew [SR,SU]
B: with carrots? [SU]
A: yeah [ACC]
(2)

A: Irthe xtes ke tis...
B: Tis to edose (SU)
(3)

The logical forms, representing the growth of semantic representations, are expressed using Type Theory with Records (TTR: Cooper, 2005), a rich type theory, which can be seen as a hybrid type theory combining assumptions from classical Montague Semantics, Situation Theory [1], and Constructive Type Theory [8, 3]. In effect, the semantic represenations are very rich and provide a lot of information that could be used in order to further perform reasoning tasks. However, these rich semantic representations are not used to their full capacity and no further reasoning can be done using DyLan based on these structures. In this talk, I will claim that this is something missing from the system and I will try to show how this can be done by suggest to connect DyLan with a well-known proof-assistant that supports reasoning, and most importantly, is able to support the expressive semantics DyLan outputs. The intention is to initiate a discussion on how this can develop into a pipeline that will automatically send the semantics built out of parsed dialogues (or dialogue fragments) to the assistant in order to perform reasoning tasks.

*The research reported in this paper was supported by grant 2014-39 from the Swedish Research Council, which funds the Centre for Linguistic Theory and Studies in Probability (CLASP) in the Department of Philosophy, Linguistics, and Theory of Science at the University of Gothenburg.

2 The Coq Proof Assistant

The idea behind Coq is simple and can be roughly summarized as follows: you use Coq in order to see whether propositions based on statements previously pre-defined or user defined (definitions, parameters, variables) can be proven or not. Coq is a dependently typed proof-assistant implementing the calculus of Inductive Constructions (CiC, see [5]). This means that the language used for expressing these various propositions (or events for linguistics) is a rich type theory that is also suited to express the TTR-based logical forms we get in DyLan. To give an example of reasoning, assuming *John : Human, Man : Set and walk : Human* $\rightarrow$ *Prop*, one could use Coq to prove that the proposition *there is a man that walks* follows from the proposition *John walks*.[1] This will be expressed in the form of a Coq theorem. The user will then have to provide the relevant proof tactics in order to derive a proof. The example below shows the theorem in Coq notation:

(1) A theorem to be proven in Coq

```
Theorem EX: walks(John) -> exists x: Human, walks x.
```

3 On constructing CoDyLan

DyLan, as already mentioned, has the ability to deal with an impressive array of dialogue data of high complexity and does so in a way that further outputs rich semantic representations. In this respect, one would hope that these rich semantic representations are used in order to perform reasoning tasks. Such an inference engine is something that DyLan lacks. But what if we connected DyLan with Coq? TTR record types can be straightforwardly encoded in Coq using the record type mechanism and other types of rich typing constructions can also be supported (most importantly dependent typing). Consider the way Cooper's well-known shown in (2) can be encoded in Coq (3):

(2)
$$\begin{bmatrix} \text{x} & : & \textit{Ind} \\ \text{c1} & : & \textit{man x} \\ \text{y} & : & \textit{Ind} \\ \text{c2} & : & \textit{donkey y} \\ \text{c3} & : & \text{own(x,y)} \end{bmatrix}$$

(3)

[1] *Set* and *Prop* are type universes in Coq (roughly collections of types). Types of type *Set* are specifications with proofs of these specifications being programs, while types of type *Prop* are logical propositions (*Prop* is the universe of logical propositions). Reducing our perspective to lingistics, *Set* would contain the elements that *e* would contain in Montague Grammar and *Prop* would be closer to type *t* accordingly. The reader has to be aware that these analogies are crude oversimplifications given the absence of space to properly explain Coq's type system.

```
Record amanownsadonkey : Type := mkamanownsadonkey{
x : Ind;
c1 : man x;
y : Ind;
c2 : donkey y;
c3 : own x y }.
```

Now, Coq can now be used to reason with the record type and one can prove propositions like the following: *there is an entity which is a man*, *there is an entity that is a donkey*, *there are two entities, one a man, one a donkey, that are in an owning relationship* etc. A proof from the record type *amanownsadonkey* to the latter proposition is shown below. Coq is put into proof-mode by the command theorem:

```
Theorem RECORD : amanownsadonkey -> exists x, man x
/\ exists y, donkey y /\ own x y. .
```

Using simple proof-tactics, *intro* (moves the antecedent of a conditional as a premise in the context and the goal is to prove the consequent), *decompose record* (decomposes a record type into its individual fields), *split* (splits a conjunction in the goal) and *assumption* (matches the goal with a premise), the theorem can be proven.

In principle, reasoning can also be performed for the record types that are the output of DyLan. One has then to connect the two systems. More examples of how this can be done will be presented in the talk. Furthermore, ideas on how to create a pipeline based on DyLan that will perform reasoning tasks based on the DyLan's output semantics will also be discussed.

References

[1] Jon Barwise and John Perry. *Situations and Attitudes*. MIT Press, 1983.

[2] R. Cann, R. Kempson, and L. Marten. *The Dynamics of Language*. Elsevier, Oxford, 2005.

[3] Stergios Chatzikyriakidis, Zhaohui Luo, et al. *Modern perspectives in type-theoretical semantics*, volume 98. Springer.

[4] Robin Cooper. Records and record types in semantic theory. *Journal of Logic and Computation*, 15(2):99–112, 2005.

[5] The Coq Development Team. *The Coq Proof Assistant Reference Manual (Version 8.1), INRIA*, 2007.

[6] A. Eshghi, M. Purver, and Julian Hough. Dylan: Parser for dynamic syntax. Technical report, Queen Mary University of London, 2011.

[7] R. Kempson, W. Meyer-Viol, and D. Gabbay. *Dynamic Syntax: The Flow of Language Understanding*. Blackwell, 2001.

[8] P. Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, 1984.