Pragmatics Through Context Management

Joakim Nivre
Göteborg

Abstract

Pragmatically based dialogue management requires flexible and efficient representation of contextual information. The approach described in this paper uses logical knowledge bases to represent contextual information and special abductive reasoning tools to manage these knowledge bases. One of the advantages of such a reasoning based approach to computational dialogue pragmatics is that the same rules, stated declaratively, can be used both in analysis and generation.

1 Introduction

The purpose of the present paper is to illustrate an approach to computational dialogue pragmatics that has been developed within the ESPRIT project PLUS (A Pragmatics-based Language Understanding System, ESPRIT P5254).[1] The purpose of this project was to build a dialogue system for information-seeking (in the domain of the Yellow Pages), and the basic idea of the project was to improve on existing systems by making heavy use of pragmatics, i. e. by enabling the system to make systematic use of contextual information in interpretation, planning and response generation.[2]

In the PLUS system, contextual information is stored in a set of **knowledge bases**, represented as logic programs. The most important of these knowledge bases is the Discourse Model, which contains the information derived from the ongoing dialogue. The process of dialogue management, i. e. of interpreting the user's contributions, planning the system's actions, and generating appropriate responses, is then conceived as the process of maintaining the contextual knowledge bases (in particular, the Discourse Model) through such knowledge base operations as querying, updating, checking (and restoring) consistency, etc. From a computational point of view, then, the management of dialogue can be seen as a side effect of the process of maintaining the contextual

[1]The PLUS consortium consists of CAP GEMINI INNOVATION (France), CAP debis (Germany), ITK (The Netherlands), Omega Generation (Italy), UMIST (England), LIMSI (France), University of Bristol (England), University of Göteborg (Sweden). The Swedish part of the project has been funded by Teleannons AB and NUTEK.
[2]For an overview of the PLUS project, see Black et al (1991).

knowledge bases (cf. Gallagher et al 1992). In short, **dialogue management** is reduced to **context management.**

In what follows, I will attempt to illustrate the PLUS approach to dialogue management by means of a few simple examples. I will begin by giving a brief overview of the PLUS system (section 2). After that, I will present an extremely simplified version of the Discourse Model, containing only the features that are absolutely necessary to illustrate the basic process of dialogue management (section 3). Finally, in section 4, I will try to show how the process of dialogue management can be implemented through the management of contextual knowledge bases using a set of special abductive update procedures.

Needless to say, the work presented here draws extensively on collaborative work within the PLUS project. Most of these debts are acknowledged through references cited throughout the text. In addition, the paper is based on (so far unpublished) work carried out together with Jens Allwood and Björn Beskow after the completion of the PLUS project. It is also worth mentioning that the discussion of PLUS work contains many simplifications and omissions, mainly due to limitations of space. For a more complete account of the PLUS approach to pragmatics and dialogue management, the reader is referred to Bunt and Allwood (1992), Nivre et al (1992), Bunt et al (1992) and Bego et al (1992).

2 Overview of the PLUS System

The PLUS system is meant to be a prototype for an information dialogue system using typed terminal input. It consists of three main components:

- A Dialogue Manager (DM)
- A Natural Language Engine (NLE): parser and surface generator
- An application database (the Yellow Pages for the prototype)

The Dialogue Manager is the heart of the system. It receives parsed user input from the NLE, it queries the application database and it generates system responses which are converted into output strings by the surface generator. The Dialogue Manager itself can be broken down into three components:

- A World and Application Model
- A Discourse Model
- A Knowledge Base Management System (KBMS)

The World and Application Model is a static knowledge base containing general world knowledge as well as information about the application database. The Discourse Model is a dynamic knowledge base which is built up and modified during the course of a dialogue. The KBMS, finally, is a set of procedures for managing the knowledge bases (querying, updating, consistency checking, etc.).

The tasks of the Dialogue Manager include interpretation of user contributions (given the output of the parser), planning of system actions (such as querying the database), and generation of system responses (which are fed to the surface generator). These tasks are referred to collectively as **dialogue management**.

The pragmatics-based approach of PLUS entails that dialogue management be based heavily on contextual information. The contextual information includes information in the World and Application Model (static context) as well as information in the Discourse Model (dynamic context). In this paper, I will concentrate exclusively on the use of information in the Discourse Model.

3 A Simple Discourse Model

The contextual knowledge bases in the PLUS system are implemented as logic programs. In this section, I will outline a very simple Discourse Model to illustrate the basic principles of this approach. (For the specification of the actual PLUS Discourse Model, see Bunt et al 1992.)

3.1 Dialogue History

In order to keep track of the dialogue history, we need to record (at least) the following aspects of each contribution (or "utterance") in the dialogue:

- Contributor (or "speaker")
- Verbatim form
- Grammatical structure
- Semantic (propositional) content
- Communicative function (illocutionary force)

These aspects can be specified by simple facts of the following form:

```
(1) contribution(N,Agent).
    verbatim(N,String).
    gram_structure(N,Structure).
    prop_content(N,P).
    comm_function(N,CF).
```

The simple atomic formulas in (1) can be taken to represent a context where the Nth contribution to the dialogue was made by `Agent`, having the verbatim form `String`, the grammatical structure `Structure`, the propositional content `P`, and the communicative function `CF`.[1]

3.2 Attitudes

Both the interpretation of user contributions and the planning of system actions (including dialogue contributions) normally require reasoning about propositional attitudes, such as beliefs and intentions, attributed to the user and the system. For example, a context where the system believes some proposition `P`, where the user doesn't know whether `P`, and where the system wants the user to believe `P` can be represented as follows:

```
(2) bel(system,P).
    ¬know_wh(user,P).
    want(system,bel(user,P)).
```

3.3 Rules and Constraints

So far, we have only considered simple facts (i. e. atomic formulas and their negations). However, the Discourse Model must also contain **rules** (universally quantified conditionals) defining relations between different types of contextual information. For example, rules of the following kind may be proposed to capture the relations between communicative functions (such as `state` and `ask`) and the propositional attitudes underlying these communicative functions:

```
(3) comm_function(N,state)  ←  contribution(N,A),
                               gram_structure(N,S),
                               ¬interrogative(S),
                               prop_content(N,P),
                               bel(A,P),
                               want(A,bel(B,P)),
                               interlocutor(A,B).

(4) comm_function(N,ask)    ←  contribution(N,A),
                               prop_content(N,P),
                               ¬know_wh(A,P),
                               want(A,know_wh(A,P)),
                               interlocutor(A,B).
```

[1]In addition to these aspects of contributions, the real PLUS Discourse Model also includes information about such things as topic, focus and discourse referents (cf. Bunt et al 1992).

The first rule can be read as saying that a non-interrogative contribution with propositional content P is a statement (or has the communicative function state) if the contributor believes P and wants the interlocutor to believe P. In the same vein, the second rule says that a contribution with propositional content P is a question (or has the communicative function ask) if the contributor doesn't know whether P but wants to know whether P.

The rules in (3–4) define positive relationships between different types of facts (i. e. if the clauses in the antecedent are true, then the consequent is also true). However, we also have a need for negative **constraints** stating that a certain conjunction of clauses cannot be simultaneously true in the knowledge base. In order to do this, we introduce a special predicate inconsistent, occurring in the consequent of such constraints. For example, the following are constraints saying that the Discourse Model is inconsistent if it contains a contribution without a verbatim form, a contribution without a grammatical structure, a contribution without a propositional content, or a contribution without a communicative function.

```
(5) inconsistent   ←   contribution(N,Agent),
                       ¬verbatim_form(N,String).

(6) inconsistent   ←   contribution(N,Agent),
                       ¬gram_structure(N,Structure).

(7) inconsistent   ←   contribution(N,Agent),
                       ¬prop_content(N,P).

(8) inconsistent   ←   contribution(N,Agent),
                       ¬comm_function(N,CF).
```

The joint effect of these constraints is that every contribution is required to have a verbatim form, a grammatical structure, a propositional content as well as a communicative function in order for the Discourse Model to be consistent.

3.4 Reasoning Tools

The Discourse Model (and the other contextual knowledge bases) in the PLUS system are implemented as logic programs. The KBMS tools developed for the management of these knowledge bases support standard operations of asserting and retracting facts from a knowledge base, querying the knowledge base (to find out if a goal is a consequence of the knowledge base) and checking that the knowledge base is consistent.

In addition, special procedures for abductive updates have been developed and implemented (cf. Guessoum and Gallagher 1992). The two main predicates of these procedures are `insert` and `delete`, which can be characterised in the following way:

- The call `insert(P,KB,Trans)` returns the list `Trans` of transactions (asserts and retracts) that would make `P` a consequence of the knowledge base `KB`.
- The call `delete(P,KB,Trans)` returns the list `Trans` of transactions (asserts and retracts) that would ensure that `P` is no longer a consequence of the knowledge base `KB`.

A special use of these update procedures is the deletion of the special predicate `inconsistent` from the Discourse Model. As we will see in the next section, the insertion of new facts into the Discourse Model will often violate constraints in the Discourse Model, temporarily giving rise to states where the Discourse Model is "inconsistent" in the sense that the formula `inconsistent` is a consequence of the knowledge base. The normal way for the system to deal with this problem is to attempt to remove the inconsistency through an abductive update, i. e. by deleting the formula `inconsistent`. This move may then introduce new inconsistencies which have to be deleted through further updates and so on.

4 Dialogue Management

By means of a few simple examples, I will now try to outline how the process of dialogue management can be implemented through the use of abductive update procedures to maintain a contextual knowledge base of the kind described in the preceding section. I will subdivide the process of dialogue management into three subprocesses:

- Interpretation of user contributions
- Planning of system actions
- Generation of system responses

It is important to note, however, that this is an analytic division which is made primarily for purposes of exposition and which does not correspond in any straightforward way to "system modules". The basic computational process is the same in all three cases, and many of the rules and constraints involved apply across several subprocesses.

4.1 Interpretation

Whenever the user types some input (and hits the return key) the Discourse Model is updated by inserting facts of the following form (where `N` is some number and `String` is the verbatim form of the user input):

```
(9)  contribution(N,user).
     verbatim_form(N,String).
```

Since there are no rules in the Discourse Model which allows the system to prove such facts, they will simply be asserted into the Discourse Model. (In other words, user contributions and their verbatim form can only be observed, they can never be inferred, neither deductively nor abductively.)

Asserting these facts into the Discourse Model will make the knowledge base inconsistent, because of the following constraints (cf. section 3.3):

```
(6)  inconsistent   ←   contribution(N,Agent),
                        ¬gram_structure(N,Structure).

(7)  inconsistent   ←   contribution(N,Agent),
                        ¬prop_content(N,P).

(8)  inconsistent   ←   contribution(N,Agent),
                        ¬comm_function(N,CF).
```

In order to make the Discourse Model consistent again, the system must prove that the `N`th contribution from the user has a certain grammatical structure `Structure`, a certain propositional content `P`, and a certain communicative function `CF`. In a PLUS-like system, the first goal will be resolved by calling the parser, which will instantiate the variable `Structure` to a grammatical feature structure containing, among other things, a compositional semantic analysis of the input. From this semantic analysis, together with contextual information already stored in the Discourse Model, the system will then attempt to derive a propositional content `P` for the contribution in question.

Let us now consider in a little more detail how the analysis of communicative function can proceed. In order to make the Discourse Model consistent again, the system must prove the goal `comm_function(N,CF)`, for some `CF`. As noted above, the Discourse Model contains rules relating to communicative function, such as the following (cf. section 3.3):

```
(3)  comm_function(N,state)  ←  contribution(N,A),
                                gram_structure(N,S),
                                ¬interrogative(S),
                                prop_content(N,P),
                                bel(A,P),
                                want(A,bel(B,P)),
                                interlocutor(A,B).

(4)  comm_function(N,ask)    ←  contribution(N,A),
                                prop_content(N,P),
                                ¬know_wh(A,P),
                                want(A,know_wh(A,P)),
                                interlocutor(A,B).
```

The important point about these rules is that the attitude goals cannot be proven deductively in the Discourse Model but have to be abduced (if they are compatible with the rest of the system's knowledge). For example, when the system tries to insert that a certain user contribution is a statement, the update procedures will propose as a possible transaction to assert (i. e. to abduce) that the user believes the propositional content and wants the system to believe the same thing.

If there is no conflict with the rest of the information in the Discourse Model, these attitude facts can be assumed, representing an interpretation of the communicative function of the user's contribution. If there is conflicting information (the system may know on other grounds that the user does not believe the propositional content), then the abduction is blocked and the system will continue to search for another interpretation. If all interpretations are blocked in this way, the system will be forced to initiate a repair (asking the user what she meant, whether she has changed her mind, etc.).

4.2 Planning

As we have seen above, the interpretation of a user contribution will typically result in the abduction of a set of user attitudes (beliefs, goals, etc.) in order to restore the consistency of the Discourse Model. However, the addition of these user attitudes will normally generate new inconsistencies, because of constraints relating user attitudes to system attitudes. For example, if the system is meant to be ideally cooperative, then it seems reasonable to require that any goal of the user is also a goal of the system (with certain restrictions that I will not go into here). A cooperativity constraint of this kind would have the following form:

```
(10)  inconsistent    ←  want(user,P),
                         ¬want(system,P).
```

The presence of this constraint in the Discourse Model will guarantee that as soon as the system has inferred that the user has a certain goal, the system will try to insert that it has the same goal. This insertion will generate further system goals, such as the goal to find a certain piece of information in the database and give it to the user, etc. In this way, planning of system actions can be carried out by the same basic process of maintaining the Discourse Model through abductive updates that was used for the interpretation of user contributions.

4.3 Generation

A very basic requirement on a cooperative dialogue system, is that it should generate a response to every contribution from the user. This requirement can be implemented by adding the following constraint to the Discourse Model:

```
(11)  inconsistent  ←  contribution(N,user),
                       ¬contribution(N+1,system).
```

This constraint will ensure that the addition of a user contribution to the Discourse Model is always followed by the addition of a system contribution. Moreover, once the new contribution has been added, the constraints in (5–8) will come into play again and will drive the generation process further until a fully specified system contribution has been generated. In this way, the same constraints are used to drive both interpretation and generation.

Furthermore, the rules relating to communicative functions can also be exploited both in interpretation and in generation. Suppose, for example, that the Nth contribution has been interpreted as a question by the user with propositional content P. Suppose further that the system knows P to be the case (P may be a fact in the application database, such as the fact that a certain company has a certain phone number), and that we have the following (not too implausible) rules in the Discourse Model:

```
(12)  want(A,bel(B,P))   ←  want(A,know_wh(B,P)),
                            P.

(13)  bel(system,P)      ←  P.
```

We can then prove the following facts in the Discourse Model:

```
(14)  bel(system,P).
      want(system,bel(user,P)).
```

Now, given the constraint in (11), the system will sooner or later be forced to add a new system contribution to the Discourse Model:

(15) `contribution(N+1,system).`

And in order to satisfy the constraint in (8), the system must then be able to prove `comm_function(N+1,CF)` for some `CF`. If we consider the rules (3–4) and the facts in (14), we see that it may be possible for the system to prove `comm_function(N+1,state)`, thus generating an answer to the question, but not `comm_function(N+1,ask)`, which would result in a new question. Without going into all the details, the important point is that the same rules and constraints apply both in interpretation and generation.

5 Conclusion

In the present paper, I have tried to illustrate a certain approach to dialogue management based on knowledge base representations of contextual information and reasoning tools incorporating abductive updates. There are still many open problems in relation to the use of abductive reasoning, but I nevertheless think that the approach presented here is interesting enough to merit further attention. Hopefully, I am not alone in thinking this.

References

Bego, H. (ed.), B. Beskow, H. Bunt, A. Derain, L. Horel, K. Jokinen, W. Kraaij, D. Pernel and G. Tabuteau. 1992. *Pragmatic Knowledge in PLUS. Part III: Pragmatic Rules*, Review Report. ESPRIT-II-P5254 PLUS.

Black, W. J. (ed.), J. Allwood, H. C. Bunt, F. J. H. Dols, C. Donzella, G. Ferrari, J. Gallagher, R. Haidan, B. Imlah, K. Jokinen, J.-M. Lancel, J. Nivre, G. Sabah and T. J. Wachtel. 1991. *A Pragmatics-based Language Understanding System.* In INFORMATION PROCESSING SYSTEMS. RESULTS AND PROGRESS OF SELECTED PROJECTS IN 1991, ESPRIT, Brussels.

Bunt, H. and J. Allwood. 1992. *Pragmatics in PLUS.* Deliverable D2.3., ESPRIT-II-P5254 PLUS.

Bunt, H. and C. Godin, (eds.), K. Jokinen, W. Kraaij, R. Meyer, J. Nivre and D. Pernel. 1992. *Pragmatic Knowledge in PLUS. Part II: Discourse Model.* Review Report, ESPRIT-II-P5254 PLUS.

Gallagher, J., A. Guessoum, R. Haidan and R. Meyer. 1992. *Reasoning Tools and Pragmatic Reasoning.* Internal Report, ESPRIT-II-P5254 PLUS.

Guessoum, A. and Gallagher, J. 1992. *The PLUS Update Procedures.* Internal Report, ESPRIT-II-P5254 PLUS.

Nivre, J. (ed.), A. Cavalli, A. Guessoum, T. Lager, R. Meyer and N. Underwood. 1992. *Pragmatic Knowledge in PLUS. Part I: World and Application Model.* Review Report, ESPRIT-II-P5254 PLUS.