PRAGMA – A Flexible Bidirectional Dialogue System

John M. Levine

University of Cambridge, Computer Laboratory,
Pembroke Street, Cambridge CB2 3QG, England.
jml@uk.ac.cam.cl

Abstract

This paper gives an overview of a natural language dialogue system called PRAGMA. This system contains a number of novel and important features, as well as integrating previous work into a unified mechanism. The most important advance that PRAGMA represents compared with previous systems is the high degree of bidirectionality employed in its design. A single grammar is used for interpretation and generation, and the same knowledge sources are used for plan recognition and response generation. The system is also flexible, in that it generates useful extended responses, not only to queries which allow the user's plan to be inferred, but also to queries which do not allow this.

1. Introduction

One of the most important and challenging areas of research in artificial intelligence is the design and construction of natural language dialogue systems. This task is one of the hardest artificial intelligence has to offer, since the overall goal is the implementation of a fully capable and fluent conversational partner. Research into the issues involved in building such a system is important for three main reasons. Firstly, it serves as an ideal context for the investigation of basic issues in natural language processing (e.g. syntax, semantics, parsing and generation) and artificial intelligence (e.g. knowledge representation, theorem proving, planning and plan recognition). Secondly, it is an important application area, in the form of natural language interfaces to database systems, explanation facilities for expert systems and interactive advisory services such as online help systems. Lastly, in terms of linguistic inquiry, it allows us to test existing theories of language and communication and to propose and test new ones by giving these substance in the form of a working computational model.

This paper describes a natural language dialogue system which contains many novel and significant features, as well as integrating previous work from heterogenous sources into a unified mechanism. The system is called PRAGMA, which is an acronym for 'Plan Recognition And Generation of Meaningful Answers.' The most important advance that PRAGMA represents compared with previous systems is the high degree of bidirectionality employed. A single unification grammar with Montague semantics (Montague, 1974; Dowty, Wall and Peters, 1981) is used for literal interpretation and tactical generation (i.e. sentence realisation from a representation of content and thematic organisation). Also, the same sources of information about possible plans, the domain, and the user's beliefs are used to perform plan recognition and strategic generation (i.e. planning the content and thematic organisation of an appropriate response).

Apart from this high degree of bidirectionality, there are three other aspects of the PRAGMA system which are novel and important. Firstly, we use a linguistically well-motivated set of functional features to specify the thematic organisation of a sentence. Secondly, the strategic generation component of the system provides useful extended responses, not only to queries which allow the user's plan to be inferred, but also to queries which do not allow this. Lastly, we use a special algorithm for comparison of logical forms which solves the problem of logical form equivalence (Appelt, 1987; Shieber, 1988; Calder, Reape and Zeevat, 1989).

The overall organisation of PRAGMA is shown in Figure 1. The Grammar Development Environment (Briscoe et al., 1987; Carroll et al., 1988) was used to define the bidirectional grammar and also acts as the major part of the literal interpretation component of the system. The propositional content of sentences is represented using standard first-order logical forms. Considered as a single unit, the logical form and functional features of a sentence are known as the *logical message* of that sentence.

The plan recognition algorithm used in PRAGMA is based on research by Litman and Allen (1987) and Allen (1987). If plan recognition is successful, four significant pieces of information are inferred: the user's discourse plan, discourse goal, domain plan, and domain goal. If plan recognition is unsuccessful, only

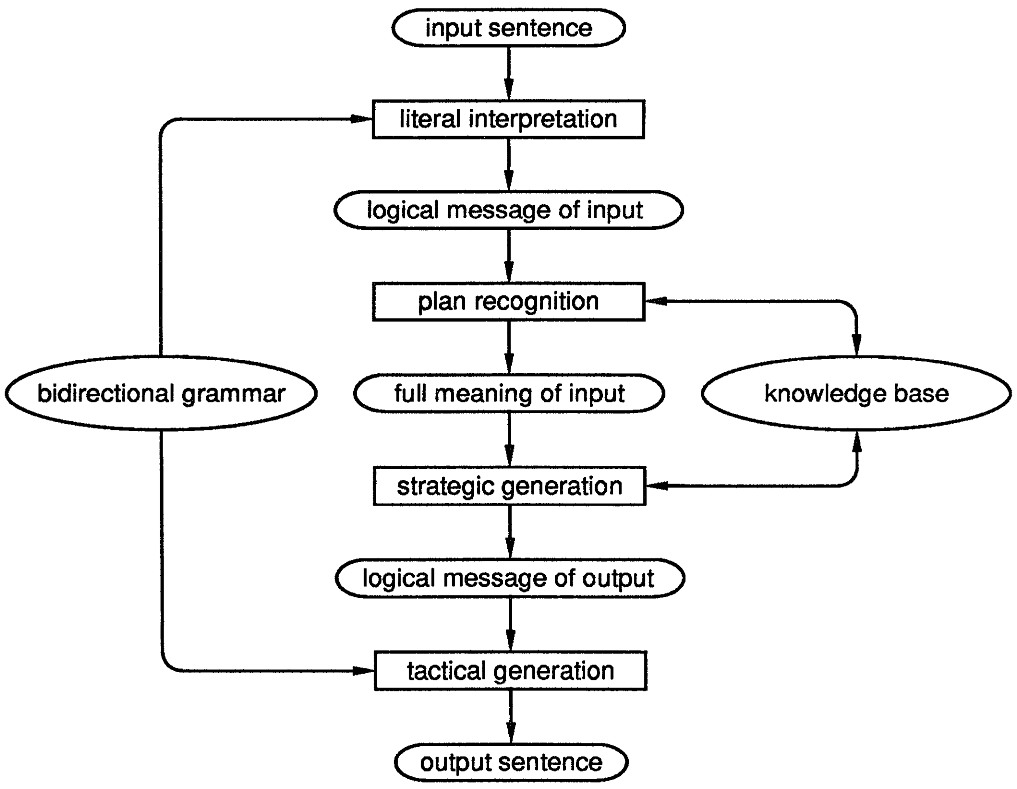

Figure 1 The Organisation of the System

the user's discourse plan and discourse goal are inferred. The output from the plan inference process, together with the logical message of the user's input sentence, is referred to as the *full meaning* of the sentence.

The strategic generation component of the system can deal with two types of dialogue. Task-oriented dialogues are handled by examining the user's domain plan and planning speech acts to enable or disable this where appropriate. An example of this type of interaction with PRAGMA is given below.

Q: Where is Pizzaland?
A: It's in Regent Street, but it isn't open today.
Q: Is the Pizza Hut open today?
A: No, but Pizza Express is.
Q: I don't know where that is.
A: It's in St. Andrew's Street.
Q: OK, thanks.

In the case of dialogues where no specific domain plan can be inferred, the functional features of the user's query are used as a guide to the construction of appropriate extended responses. An example of this type of dialogue is shown below.

Q: Who designed Trinity's library?
A: It was designed by Wren.
Q: Was Trinity founded by Wren?
A: No, it was founded by Henry the Eighth.
Q: Did Wren found King's?
A: No, it was Henry the Sixth who founded King's.

PRAGMA is written in Common Lisp and runs on a Hewlett Packard 9000 Series workstation under the HP-UX operating system. On average, a dialogue consisting of a single question and answer pair takes about 4 seconds of CPU time to run to completion.

The next four sections of the paper describe each of the processing components of PRAGMA. To demonstrate how the system works, a single example will be traced through each of these stages. The example to be used for this purpose is as follows:

Q: Is the Pizza Hut open today?
A: No, but Pizza Express is.

The final section of the paper briefly compares the work reported here with previously reported dialogue systems, evaluates PRAGMA, and discusses some directions for further research.

2. Literal Interpretation

The Grammar Development Environment (GDE) is a powerful software tool which facilitates the design and construction of large natural language grammars. Since it includes efficient facilities for parsing sentences and extracting the logical forms from the resulting parse trees, it is also used as the main part of the literal interpretation component of PRAGMA. The grammatical formalism employed by the GDE is similar to GPSG (Gazdar et al., 1985). This is compiled by the GDE to form a unification grammar consisting of a set of phrase structure rules whose categories are feature complexes. A simplified version of Montague semantics, similar to that used by Rosenschein and Shieber (1982), is used to compute the logical forms of sentences from their parse trees.

The logical message formalism used for the intermediate representation of a sentence consists of a logical form plus a set of functional features. The notion of functional features is suggested by Appelt (1987), but he does not suggest a taxonomy; the work reported by Levine (1989) and in more detail by Levine and Fedder (1989) attempts to provide this. The purpose of these features is to provide the information contained in a sentence which is not reflected in the logical form, since this is important in both interpretation and generation. Five functional features are used in PRAGMA: theme, linguistic focus, emphasis, sentence type, and tense. The first three of these are the most important, and are adapted from the research of Quirk et al. (1985).

In interpretation, the values of the functional features must be computed from the parse tree and logical form of the input sentence. The sentence level node of the parse tree contains three syntactic features: the sentence type (e.g. declarative), the tense (past or present), and the sentence style (e.g. passive). The sentence style is then used in conjunction with the logical form to determine the values of the three remaining features. For the example under consideration, the output from the literal interpretation module is shown below.

Logical form = $\exists$e.open(pizza-hut,e) $\wedge$ on(e,today)
Sentence type = yes/no question
Tense = present
Theme = pizza-hut
Linguistic focus = today
Emphasis = false

3. Plan Recognition

In order to perform plan recognition, we must first decide what kind of planning activity is going on in the mind of the user. The assumptions that we make about the nature of this process are as follows. The user has a domain goal and is constructing a domain plan to achieve this goal using the plan library. The plan library is a collection of action schemata and is assumed to be shared knowledge between the user and the system. The system is assisting the user in the process of constructing a domain plan, since the user has incomplete and possibly inaccurate knowledge of the domain. The system has complete and accurate knowledge of the domain, but incomplete knowledge about the user's beliefs. The user needs to select an appropriate plan and then instantiate it so that it can be executed. During the plan construction process, the user may be considering a plan which cannot be proved valid or invalid given the incomplete nature of the user's knowledge. The user will want to know whether such an undecidable plan is valid or not. This means that the user will have a discourse goal of the form knowif(user,p) or knowref(user,x,p(x)); the former arises from the need to find out whether the constraints on an action are true or false, the latter from the need to instantiate the variables of an action schema. The user then uses the plan library to construct a discourse plan to achieve the discourse goal, executes the first surface speech act of this discourse plan, and then waits for the system's response.

Having made these assumptions, the plan inference process can then proceed as follows. The system uses the plan library and its knowledge of the user's beliefs to infer the user's discourse plan and hence the user's discourse goal. By knowing how discourse goals relate to domain plans under construction, the system can then attempt to infer the user's domain plan, and hence the user's domain goal.

The formalism used for defining the plan library is based on that used by Litman and Allen (1987). Action schemata are defined by a header, a set of constraints, a set of preconditions, a list of effects, and a hierarchical decomposition. An example domain plan operator for the action of eating at a restaurant is shown below (actions are shown in capital letters to distinguish them from predicates and functions, and schema variables are shown with initial capital letters).

Header:	EAT-OUT(Agent,Rest,Food,Time)
Constr:	$\exists$e.serve(Rest,Food,e)
	$\exists$e.open(Rest,e) $\wedge$ on(e,Time)
Precond:	$\exists$e.have(Agent,Price,e) $\wedge$ on(e,Time)
Effects:	$\neg\exists$e.hungry(Agent,e) $\wedge$ on(e,Time)
Decomp:	MOVE(Agent,Place,Time)
	PURCHASE(Agent,Rest,Food,Time)
	EAT(Agent,Food,Time)
Where:	Price: $\exists$e.cost(Food,Price,e)
	Place: $\exists$e.in(Rest,Place,e)

Plan recognition is performed by identifying the surface speech act of the user's input by examining the logical form and sentence type, and then using decomposition chaining followed by plan instantiation with respect to the mutual beliefs of the user and the system. These are defined, along with the system's knowledge of the domain and of the user's beliefs, using a simple epistemic theorem prover based on the design given by Allen (1987: 435ff).

For our example query, the inferred discourse plan is an askif action, as shown below. The constant 'prop' is used here to represent the logical form of the input query, i.e. ∃e.open(pizza-hut,e) ∧ on(e,today).

```
Header:  ASKIF(user,pragma,prop)
Constr:  knowif(pragma,prop)
Effects: knowif(user,prop)
Decomp:  REQUEST(user,pragma,
            INFORMIF(pragma,user,prop))
         INFORMIF(pragma,user,prop)
```

The user's discourse goal is thus identified as knowif(user,prop). This implies that it should be unified against the constraints of the plan schemata for the domain plan to be inferred. The eat-out action is thus identified, decomposition chaining is attempted, and the plan is instantiated using the information that it is shared knowledge that the Pizza Hut serves pizza. The important parts of inferred domain plan are shown below.

```
Header:  EAT-OUT(user,pizza-hut,pizza,today)
Constr:  ∃e.serve(pizza-hut,pizza,e)
         ∃e.open(pizza-hut,e) ∧ on(e,today)
Precond  ...
Effects: ¬∃e.hungry(user,e) ∧ on(e,today)
Decomp:  ...
Where:   ...
```

Thus, the user's domain goal is identified as the effect of this plan, i.e. not to be hungry today. All the plans and goals inferred collected together with the logical message from which they were inferred to form the full meaning of the user's input query.

4. Strategic Generation

The strategic generation component of the system performs four tasks. Firstly, it expands out the inferred discourse plan with respect to its knowledge of the domain; this dictates the surface speech act of the first part of the response. Secondly, it attempts to provide additional useful information based on the user's domain plan (if one has been inferred) and the functional features of the user's input. Thirdly, having decided on the content of its response, it computes the values of the functional features of the output so that the resulting utterance is natural and appropriate. Lastly, it computes the skeletal logical form of the output from the full logical form; this involves applying pronominalisation and verb phrase ellipsis when these are appropriate.

The module used for computing extended responses based only on the functional features of the question is described by Levine (1989) and in more detail by Levine and Fedder (1989). In essence, the linguistic focus is regarded as the element of the logical form that the user is most unsure about and so this can be used as a guide to the construction of an appropriate follow-up query. The problem with this work is that it is not always possible for the user to phrase the query in such a way that the most uncertain element will be identified as the linguistic focus and the most certain element identified as the theme; English syntax only allows a certain amount of flexibility in the thematic forms it provides. Hence, it is necessary to use the plan-based approach in preference to the thematic approach, and only to use the latter when the former fails.

The plan-based approach to strategic generation attempts to assist in the achievement of the user's domain goal. PRAGMA attempts to do this by a process of plan verification and repair in the context of the user's query. The algorithm employed can be described briefly as follows. The inferred discourse plan dictates the first part of the system's response. If the first part of the response implies that the domain plan is valid where in fact it is not valid, the system must inform the user that the plan will not work. An example of this type of response is shown below.

Q: Where is Pizzaland?
A: It's in Regent Street, but it isn't open today.

If the first part of the response implies that the plan does not work, then the second part of the response should try to suggest an alternative plan. If no alternative plan is possible, then the second part of the response should address this instead. The suggestion of the alternative plan is constrained by the user's domain goal and discourse goal. The domain goal may not be altered. If the discourse goal is of the form knowif(user,p) and p is false, then the user is probably interested in some related true proposition, p′. This related true proposition may be found by considering (a) which variants of p lead to reasonable plans, (b) how high up in a plan the terms in p are instantiated, and (c) the thematic organisation of the user's input. The system can then inform the user '¬p, but p′.' In this situation, 'but' should be inserted between disablement-enablement pairs if the theme of the response is different to the theme of the query. For enablement-disablement pairs, 'but' should always be inserted.

For the example under consideration, PRAGMA first finds that the Pizza Hut is closed today and so the first part of the response is 'No,' which disables the plan the user is considering. It then tries to enable a different but related plan. It finds that 'today' cannot be varied, since this is present in the user's domain goal; however 'pizza-hut' can be varied and so this is replaced by a variable in the plan. The system then tries to verify this plan against its domain knowledge and this succeeds, with the variable being bound to 'pizza-express' in the process. The logical message of the output is then constructed, as shown below.

Logical form =
  No, but $\exists e.\text{open}(\text{pizza-express},e) \wedge \text{on}(e,\text{today})$
Sentence type = declarative
Tense = present
Theme = pizza-express
Linguistic focus = pizza-express
Emphasis = false

Finally, the full logical form is reduced to the skeletal logical form. For the example, this means that verb phrase ellipsis is applied by constructing a logical form containing a predicate which realises as the appropriate verb phrase anaphor.

5. Tactical Generation

The tactical generation component of PRAGMA consists of two modules. The first uses the functional features of the output logical message to compute the sentence style, essentially reversing the process carried out during interpretation. The second module is the sentence generator (originally designed and implemented by Lee Fedder), which operates by forming trees top-down, breadth-first. This process is guided by the syntactic features attached to the sentence level node, but some semantic information from the goal logical form is also used to cut down the search space.

One of the biggest difficulties in using a bidirectional grammar in a dialogue system is the problem of logical form equivalence. This occurs when the tactical generator is presented with a logical form which is logically equivalent to but syntactically distinct from one for which the grammar defines a set of surface forms. For example, this may occur because logical connectives are associative and commutative. The problem also arises with stative sentences like 'Trinity is a college' if the stative verb is translated as the predicate 'equal' to maintain compositionality in the semantics of the grammar.

However, the problem of logical form equivalence can be solved relatively easily for the sentence generator used in PRAGMA, since it only attempts to equate the goal logical form with the constructed logical form when tree formation is complete. Hence, what is required is an algorithm which converts these first-order logical forms into a standard format so that unification can then be applied. The algorithm used in PRAGMA to do this can be described as follows. First the logical form is skolemised and converted into Kowalski normal form; this process is described by Bundy (1983). Equalities are then massaged out of the resulting expression by replacing one side of an equality by the other when this is appropriate. A strict ordering is imposed on which side of the equality is eliminated to preserve information content: constants replace variables and skolems, variables replace skolems, skolem constants replace skolem functions, and skolem functions of arity n replace skolem functions of arity greater than n. Tautologies are then removed from the ifs and thens of all the clauses, and clauses with empty thens are removed. The ifs and thens of all the clauses are then sorted by alphabetical ordering, and the clauses are then sorted by similar means. The resulting expression is now in the standard form required, since all variants of the original logical form will convert to this standard form.

For the example under consideration, the final natural language response produced by PRAGMA is 'No, but Pizza Express is.' This is only one example of the variety of useful extended responses that PRAGMA is capable of producing, but it is hoped that this example demonstrates the basic workings of the system.

6. Conclusions and Further Research

The research reported here uses ideas from previously reported dialogue systems, such as those reported by Allen (1983), Kaplan (1983), and Wahlster et al. (1983), and integrates these ideas into a single system. However, it also represents a significant advance over these systems because of the high degree of bidirectionality employed.

The idea of a fully bidirectional system is yet to be realised, since the theory on which PRAGMA is built restricts it to being a supplier of cooperative responses. The next important step is to have two identically built systems conversing in natural language in order to achieve some task. A simple version of this would be to have one system emulate the user's behaviour as described in Section 3.

The plan recognition algorithm used in PRAGMA will only work well in domains where the plan library can be assumed to be shared knowledge. For the domain used in PRAGMA this is a reasonable assumption, but this will not always be the case, as demonstrated by Pollack (1986). An interesting piece of further research would involve the construction of a system which allowed Pollack's work to run alongside the work reported here.

Acknowledgements

I would like to thank my supervisor, Steve Pulman, for his expert advice and for suggesting the original idea from which this research developed. I would also like to thank Lee Fedder for allowing me to use his generator, and for many useful discussions. This research was funded by a studentship from the Science and Engineering Research Council.

References

Allen, J. F. (1983) 'Recognizing Intentions from Natural Language Utterances,' in Brady, M. and Berwick, R. C. (eds.) *Computational Models of Discourse*, MIT Press, 107–166.

Allen, J. F. (1987) *Natural Language Understanding*, Benjamin/Cummings.

Appelt, D. E. (1987) 'Bidirectional Grammars and the Design of Natural Language Generation Systems,' *Position Papers for TINLAP-3*, Association for Computational Linguistics, 206–212.

Briscoe, T., Grover, C., Boguraev, B. and Carroll, J. (1987) 'A Formalism and Environment for the Development of a Large Grammar of English,' *IJCAI-87*, 703–708.

Bundy, A. (1983) *The Computer Modelling of Mathematical Reasoning*, Academic Press.

Calder, J., Reape, M. and Zeevat, H. (1989) 'An Algorithm for Generation in Unification Categorial Grammar,' *Proceedings of the Fourth Conference of the European Chapter of the ACL*, 233–240.

Carroll, J., Boguraev, B., Grover, C. and Briscoe, T. (1988) *A Development Environment for Large Natural Language Grammars*, University of Cambridge, Computer Laboratory, Technical Report No. 127.

Dowty, D. R., Wall, R. and Peters, S. (1981) *An Introduction to Montague Semantics*, D. Reidel.

Gazdar, G., Klein, E., Pullum, G. K. and Sag, I. A. (1985) *Generalized Phrase Structure Grammar*, Basil Blackwell.

Kaplan, S. J. (1983) 'Cooperative Responses from a Portable Natural Language Database Query System,' in Brady, M. and Berwick, R. C. (eds.) *Computational Models of Discourse*, MIT Press, 167–208.

Levine, J. M. (1989) 'Taking Generation Seriously in a Natural Language Question Answering System,' *Extended Abstracts Presented at the Second European Natural Language Generation Workshop*, Department of Artificial Intelligence, University of Edinburgh, 45–51.

Levine, J. M. and Fedder, L. (1989) *The Theory and Implementation of a Bidirectional Question Answering System*, University of Cambridge, Computer Laboratory, Technical Report No. 182.

Litman, D. J. and Allen, J. F. (1987) 'A Plan Recognition Model for Subdialogues in Conversations,' *Cognitive Science 11*, 163–200.

Montague, R. (1974) 'The Proper Treatment of Quantification in Ordinary English,' in Thomason, R. H. (ed.) *Formal Philosophy: Selected Papers of Richard Montague*, Yale University Press, 247–270.

Pollack, M. E. (1986) 'A Model of Plan Inference that Distinguishes Between the Beliefs of Actors and Observers,' *Proceedings of the 24th Annual Meeting of the ACL*, 207–214.

Quirk, R., Greenbaum, S., Leech, G. and Svartvik, J. (1985) *A Comprehensive Grammar of the English Language*, Longman.

Rosenschein, S. J. and Shieber, S. M. (1982) 'Translating English into Logical Form,' *Proceedings of the 20th Annual Meeting of the ACL*, 1–8.

Shieber, S. M. (1988) 'A Uniform Architecture for Parsing and Generation,' *Proceedings of the 12th International Conference on Computational Linguistics*, 614–619.

Wahlster, W., Marburger, H., Jameson, A. and Busemann, S. (1983) 'Over-Answering Yes-No Questions: Extended Responses in a NL Interface to a Vision System,' *IJCAI-83*, 643–646.