A Planner for Efficient Dialogues

Bryan McEleney

M. Sc.

Thesis presented for the degree of

Doctor of Philosophy

to the

School of Computer Science and Informatics

College of Engineering, Mathematical and Physical Sciences

University College Dublin

Research Supervisor

Gregory O'Hare

Head of Department

Barry Smyth

February 2006

Contents

1 Introduction **2**

1.1 Motivation 2

1.2 Objectives 3

1.3 The structure of the thesis 6

2 Planning of Dialogue **9**

2.1 Introduction 9

2.2 Planning 10

2.3 Belief, desire and intention 12

2.4 Speech act theory 14

2.5 Planning and plan recognition 17

2.6 Meta-level planning 19

2.7 Plan recognition and cooperative response 21

2.8 User modelling in dialogue systems 24

2.9 Generating and understanding dialogues without planning . . 28

2.9.1 Deciding dialogue strategies using policies 29

2.10 Game theory 31

2.11 Cooperative distributed planning 34

2.12 Dialogue management and user modelling systems 35

2.13 Evaluating dialogue systems 39
2.14 Conclusion 42

3 Design of the Planner **44**
3.1 Introduction 44
3.2 Requirements 45
3.2.1 Treating dialogue as a game 47
3.2.2 Using a probabilistic belief model 48
3.2.3 Bayesian games 49
3.2.4 Maintaining the user model 50
3.3 Assumptions 51
3.3.1 Agenthood 51
3.3.2 Cooperation and sincerity 52
3.4 Design of the planner 53
3.4.1 Overview 53
3.4.2 Agent state 55
3.4.3 Representation and construction of the game tree . . . 57
3.4.4 Example problem 61
3.4.5 Evaluation of the game tree 63
3.4.6 Evaluation example 68
3.4.7 Belief revision 69
3.4.8 Stereotypes 75
3.4.9 Complexity 77
3.4.10 Multilogues 82
3.4.11 Conclusion 83

4 Evaluation **85**
4.1 Introduction 85

4.2 Implementation 86
4.3 Evaluation method 87
4.4 Amenable problems 90
4.5 Example 1: Risking misinterpretation 92
4.5.1 Plan library 94
4.5.2 Demonstrations 100
4.6 Example 2: Goal introduction problem 117
4.6.1 Demonstrations 119
4.7 Belief model acquisition 129
4.7.1 Demonstration 1 131
4.7.2 Demonstration 2 132
4.7.3 Demonstration 3 133
4.8 Related work on initiative and simulation evaluation 135
4.9 Conclusion 135

5 Planning of Negotiation Dialogue 138
5.1 Introduction 138
5.2 Value of information 141
5.2.1 Value of information example 141
5.3 Negotiation acts and their pragmatic definition 144
5.4 Realism 151
5.5 Demonstrations 153
5.5.1 Demonstration 1: Telling and proposing 153
5.5.2 Demonstration 2: Holding the floor 157
5.5.3 Demonstration 3: Request and propose 164
5.6 Acts for non-cooperative dialogue 166
5.7 Design and implementation of the negotiation planner 171
5.8 Related work 173

5.9 Conclusion 175

6 Future Work **177**

6.1 Introduction 177

6.2 Evaluation 178

6.2.1 Comparing reinforcement learning with planning 178

6.2.2 Dialogues with human beings 180

6.3 Improvements to design features of the planner 181

6.3.1 Evaluation of game trees using a logical model of belief 181

6.3.2 Beliefs about plan rules 182

6.3.3 Improved belief revision 183

6.3.4 Multilogue planning 187

6.3.5 Dealing with complexity 188

6.4 Further example problems 189

6.4.1 Long-range planning and user-model acquisition 189

6.4.2 Dry-land algorithm 191

6.4.3 Knowledge engineering and large problems 191

6.5 Negotiation dialogue 193

6.5.1 Request and propose 193

6.5.2 Planning in self-interested settings 194

6.5.3 Focussing and plan recognition in negotiation dialogue 195

6.6 Combining statistical methods in speech recognition with probabilistic planning 195

6.7 Conclusion 197

7 Conclusions **199**

A Description and usage guide for the implemented planner 202

A.1 Software architecture 202

A.1.1 Domain-level planner 203

A.1.2 Negotiation planner 204

A.2 Input file format 205

A.3 Example input file 206

A.4 Index of experiment materials 213

Abstract

This thesis describes the design, implementation, and evaluation of a dialogue management system. The system is based on current theories of planned dialogue, produced by an intentional agent whose plan is derived from beliefs about the domain state, and beliefs about plan rules. The planner is used to generate the alternatives in a game tree, which is evaluated in the context of the agent's probabilistic nested belief model. It is shown using simulation experiments that the efficiency of the generated dialogues is dependent on knowing the value of the agent's probabilistic beliefs, which justifies the use of such a model in dialogue management, rather than the more traditional logical belief model. Similar experiments are used to evaluate a set of negotiation acts, which are built-in acts that the agent can use to pass information about beliefs that increases the expected utility of domain-level plans. The negotiation planner replaces meta-level dialogue planners, which in the past have been used to generate dialogues using a logical belief model.

Acknowledgement

I would like to thank all those who helped me in producing this thesis. I would like to mention my supervisor, the students and staff of the School of Computer Science and Informatics, the authors of papers referenced in this thesis, and the reviewers of papers associated with it. Free software was important in facilitating my research, therefore I would like to thank and encourage those who contribute to the free software effort. My research was sponsored by Enterprise Ireland.

Chapter 1

Introduction

Planning and plan recognition have been identified as mechanisms for the generation and understanding of dialogues. Founded on speech act theory [6] [65], and Grice's theory of meaning [27], a body of research has developed that views cooperative dialogue as a joint activity of generation of acts by a speaker, and then plan recognition and response by the hearer [10]. It was soon realised that in this process, different belief sets are involved [53]. The speaker's beliefs are used in the generation part, whereas the hearer's beliefs are used in plan recognition, and generation of a cooperative response. To plan a dialogue of many steps, a deeply nested belief model is required, whereby one agent may generate an expectation of the other's dialogue contribution by estimating its beliefs.

1.1 Motivation

While a theory of dialogue planning has developed, the use of deeply nested belief models has not filtered through to the development of dialogue planning systems. Instead, most systems employ strategies that do not change with

the user's beliefs. McTear [49] reports that the popular architectures used today - finite state and frame-based, do not employ a model of the user. However, agent-based architectures, in which the system models its user as an autonomous agent with a belief state and plan rules, have not been shown to have any definite advantage over the simpler architectures, in terms of the quality of the dialogue [41], [3], [69]. The need for a system that produces measurably better dialogues than the architectures in use today is the primary motivation for this thesis.

A secondary motivation stems from the need for a system that can be easily understood by, and easily programmed by dialogue system designers. They should only be required to know something about the specification of dialogue plan rules, rather than about the use of the user model, or the mechanisms that choose dialogue strategies, or the mechanisms that understand the user's plan. These mechanisms turn out to be quite complex, and so their hiding is important. Today, the VoiceXML language [46] provides such a facility for state-based dialogue systems, whereby the designer need only go so far as to specify states and their transitions. The VoiceXML interpreter then uses this specification to control the dialogue.

A tertiary motivation is the need for a freely available design and implementation of an agent-based planner, that can serve as the basis for further research and development of dialogue management systems.

1.2 Objectives

With the motivations in mind of producing an agent-based dialogue system that is both easy for the designer to use, and produces better quality dialogues, the work reported in this thesis sought to achieve:

- The development of a model of dialogue planning, founded on current theories of cooperative dialogue. Such a model is required so that the dialogue acts chosen by the system can be understood by a user whose expectation is based on that model. Symmetrically, the system must be able to understand the user, who generates his dialogue acts using that model.

- The integration with this model of a suitable representation of the system's nested beliefs, since these beliefs determine the dialogue acts that each agent is expected to choose. The nested beliefs were to represent the system's own beliefs, as well as its beliefs about the user.

- Development of a mechanism for planning efficient dialogue, that considers the system's nested beliefs in the choice of a dialogue strategy.

- The implementation of a dialogue manager based on this mechanism, taking as input a set of dialogue plan rules given by the dialogue system designer, and from these automatically generating dialogue acts by planning a dialogue with the user. From the rules alone, it would automatically acquire a nested belief model, improving the efficiency of future dialogues as the model is fitted to the user. The system would be domain independent and useful to designers of dialogue systems without requiring them to understand its inner workings. It is common in designing dialogue systems to separate the components of dialogue management, and generation and recognition of user input and output. Therefore, the planner was not intended to be a complete natural language system. Rather it was intended as the dialogue manager component of such a system, which could equally well be used with alternative dialogue modalities, such as a graphical user interface,

or even in dialogues that mix spoken acts with physical acts that are seen rather than heard. The planner should be made available in the public domain as "PED" (Planner for Efficient Dialogue) to facilitate further research in dialogue management systems.

- Demonstration of the implemented dialogue manager by means of a set of example problems. While the implemented planner was not subjected to a correctness proof, which would show correctness in all circumstances, example problems provide a handful of samples of the planner's behaviour, which can be checked by hand. While the examples are not intended to represent the entire space of dialogue planning problems, they nevertheless provide evidence that a number of important problems can be solved by the planner. Since the planner is written in a high-level programming language, Prolog, a correctness proof of the program code would be almost trivial, because the specification to which the planner is proved to adhere would have been almost identical to the program code.

- Calculation of the efficiency gain that can be achieved by using a probabilistic nested belief model in a dialogue system in the context of the example problems. This gain represents the difference in quality between the best dialogue that can be planned without using a user model, and the quality that the planner achieves in exploiting the user model. If the gain is substantial, then the planner can compete with current dialogue managers. A comparison between traditional logical models of belief, and probabilistic models was also to be made.

- Development of a set of domain-independent dialogue acts which can be used in the generation of negotiation dialogues. The negotiation di-

alogue uses as its subject domain-level plan alternatives that are specified by the planner's ordinary plan rules. These acts are common to many dialogues, and so they are built in to the system, rather than expressed in the planner's input. Using these acts, a negotiation planner was to be developed for producing meta-level dialogues about a domain plan.

1.3 The structure of the thesis

The next chapter, chapter 2, introduces the subject of planning in dialogue systems, to provide a foundation for the planner's design, and to relate the planner to planning problems, and planning systems that have been discussed in the past. It describes the nature of action, planning and intentional agents, and the cooperative process of planning and plan recognition. It is shown that the nested beliefs of the agents, as well as what they say, determine the meaning of their dialogue acts. Since this thesis is about efficient dialogue, game theory is introduced to provide a mechanism for choosing between alternatives of differing value.

Chapter 3 describes the planning model and design of the planner, drawing from the foundations provided in chapter 2. The chapter starts by defining the requirements of the planner, and giving a set of assumptions about the problem setting. Then, the design components are explained, and an example is used as an illustration. The agent's state is described in terms of beliefs, desires, and the dialogue history. Then the set of processes that use and update this state is described - the planner which generates the plan alternatives, the evaluator which evaluates them so one can be chosen, and belief revision, which updates the agent's mental state in response to

observed actions.

Chapter 4 illustrates how the planner can be used to solve two practical dialogue problems. The utility gains that the planner can obtain by using belief revision to adapt the probabilistic belief model to the user are estimated using simulation. The subject of the first example is that of deciding whether to say something that has little dialogue cost, but risks plan failure due to misinterpretation, or whether to use an alternative whose meaning is clearer. The subject of the second example is that of deciding whether to pursue a goal by introducing it to the dialogue. The agent must decide whether it is better for him to take the initiative and risk plan failure, or whether to allow the other agent, who knows whether the plan will fail, to take the initiative instead. Continuing from this planning problem, a demonstration is given of the planner adapting its belief model to a user over the course of several dialogues.

Chapter 5 looks at the use of built-in negotiation acts, and how they can be used to efficiently pass information about the agents' beliefs so that they can improve the expected utility of their domain-level plan. Again a set of demonstrations is given which indicates the utility gain obtained by using a planner that can adapt a probabilistic belief model to its dialogue partner. The demonstrations compare the use of the different negotiation acts in some simple problems in a cookery domain.

Chapter 6 looks at some ideas for future work. Some of the ideas are improvements that can be made to the design and implementation of the planner. Comparisons with other planning algorithms are proposed, and evaluation in a human setting, rather than a simulated setting is discussed.

Chapter 7 concludes the thesis, discussing the objectives, and what has been done to achieve them.

There is an appendix, which serves as a guide to the implemented planner, giving a brief description of the code modules, formally defining the input file syntax, and guiding the reader to the experiment materials that correspond with the demonstrations.

Chapter 2

Planning of Dialogue

2.1 Introduction

In this chapter a wide range of research is described that addresses the topic of dialogue planning, and which forms the foundations of the work presented in the following chapters. Planning is used in dialogue systems in two ways. Most obviously, it is used in deciding what to say. As well, it is used in understanding what has been said, by performing plan recognition in reconstructing a plan that is consistent with the speaker's utterance and the dialogue history. An overview will be given of classical planning, derived from robot planning problems. Where there are two agents who cooperatively act, the topic of plan recognition becomes important, as does the planning of cooperative responses to recognised plans.

The chapter begins by introducing planning systems in Section 2.2. Then, the BDI agent model is discussed in Section 2.3. This is a logical model of the mental state of the planning agent. Speech act theory is described in Section 2.4, which provides a link between planning and language production. The topic of plan recognition is introduced in Section 2.5, as a prerequisite to

understanding of planned dialogue. Meta-level planning, where dialogues are planned that track, modify and support a domain level plan is described in Section 2.6. The combination of planning and plan recognition in producing cooperative responses to a user is described in Section 2.7. A Section on user modelling, Section 2.8, describes the representation, acquisition and maintenance of user models in a dialogue system. In contrast to symbolic planning of dialogue, Section 2.9 describes systems that plan and understand dialogue by ignoring plan structures and using statistical information instead. Section 2.10 describes game theory, which will be used as a quantitative basis for deciding dialogue strategies. Section 2.11 describes approaches to loosely coupled plan decomposition and coordination among agents, and relates these to the problem of dialogue planning. Section 2.12 describes generic or shell systems that use user modelling, which can be programmed with dialogue plan rules to produce a dialogue system. Finally, Section 2.13 describes measures that are used to evaluate dialogue systems.

2.2 Planning

In planning systems, actions are functions from states to states. Actions are chosen by an agent, who must construct an ordered sequence of them, transforming a given initial state to a final state. The states are descriptions of the world, and so are represented by propositions in a logical language. It is assumed that within this world, the only changes that occur are those due to the planning agents. The set of states may not be finite, and so the set of pairs of states required to describe the function over its domain may not be finite, yet the function needs to be expressed as compact rules. In the STRIPS planning system [21], actions are realised as a mechanism

that checks entailment of a precondition proposition by the given state, and then adds and deletes propositions from a set of conjoined propositions using add and delete lists. The states are described using a simplified language of sets of atomic predicates over objects. The planner typically uses a search algorithm with heuristics to obtain a sequence that takes the agent from its observed initial state to its goal state.

For some problems, planning using STRIPS can be impractical, since it can require searching of very long chains of low-level actions. Hierarchical planning [60] addresses this problem by adding decomposition rules to the planning system. This allows aggregations of steps to be represented as well as more basic steps, producing shorter chains and thereby often reducing the search time. For example, a robot who needs to assemble a car would be able to construct a high level plan that checks for available parts and supplies for each assembly task, before proceeding to planning the individual arm movements that constitute one assembly task. This allows infeasible plans to be rejected before the low-level planning is done. Hierarchical planning proceeds by a combination of decomposition chaining and precondition-effect chaining. In particular, decomposition has emerged as simple and effective representation for dialogue planning rules, and forms the foundation for most dialogue planning and plan recognition models. By using decomposition, the language of dialogue act sequences can be expressed using a context-free grammar. For this reason, sets of plan rules are often called "dialogue grammars".

2.3 Belief, desire and intention

When one agent reasons about the plans of other agents, it must consider that the preconditions to actions are evaluated in the context of not its own, but rather the other agent's beliefs. Such beliefs can be about the agent's capabilities with respect to actions, represented as the plan rules that it holds, their effects, or about the environment state. Beliefs are expressed using a modal logic [34], whereby a modal operator B is used to express that an agent believes a proposition. The agent is taken to consider a set of worlds, each of which corresponds with a consistent set of propositions that hold in that world. The intersection of all of the sets of propositions of the possible worlds constitutes the propositions that the agent believes. Those propositions that occur in only some of the sets are considered possible, while those that occur in none of the worlds are disbelieved. The beliefs can be nested, since propositions to which the belief operator has been applied are also propositions. To formally define the semantics of a modal logic, a Kripke model is used [42], which is a set of worlds, and an accessibility relation between worlds. A sentence $B(P)$ holds in a world $w1$ if and only if for every world $w2$ accessible from $w1$, P holds. A number of axioms can be introduced, whose validity depends on certain constraints on the accessibility relation. One useful set of axioms is KD45, given below.

$$\begin{aligned} &K : B(A) \wedge B(A \rightarrow B) \rightarrow B(B) \\ &D : B(A) \rightarrow \neg B(\neg A) \\ &4 : B(A) \rightarrow B(B(A)) \\ &5 : \neg B(A) \rightarrow B(\neg B(A)) \end{aligned} \tag{2.1}$$

The language of belief can be extended so that an agent may have be-

liefs about other agents, allowing the agents to reason about one another's plans. For example, the following sentence might be generated by agent X, expressing that agent Y believes that agent Z believes P.

$$B(Y, B(Z, P)) \tag{2.2}$$

An agent is said to know a proposition only if the proposition can be proved and only if it can be proved that the agent believes the proposition. Agent A might then claim "agent B knows P", which means that both agent A believes P and that agent A believes that agent B believes P.

$$P \wedge B(P) \leftrightarrow K(P) \tag{2.3}$$

A common metaphor for reasoning about agents, for designing agents, and for designing agents that reason about other agents is that of a mental state consisting of beliefs, desires and intentions [17]. This is known as a BDI model or BDI architecture [7]. In Rao and Georgeff's decision model for BDI agents [55], the desires of the agent describe the preferred goal states, in the same sense of a goal as in STRIPS planning. For each of the agent's possible worlds, generated by its beliefs, desires and intentions, there is a time-tree structure that represents the actions of the agent on its edges and states at the nodes. The agent's goals can be any consistent subset of its desires. A subset of the the possible worlds will have corresponding time-trees that are consistent with a goal. These are called the goal-accessible worlds. The agent must choose and commit to one goal. The worlds in which the agent is committed to a goal are the intention-accessible worlds, which are a subset of the goal-accessible worlds. A number of BDI architectures have been proposed, for example IRMA [7] and PRS [24], in which the agent's mental state is composed of beliefs, desires and intentions, and in

which the agent executes a cycle of observation of the environment, update of beliefs, deliberation of over intentions, and execution of an intended plan. The planner developed in this thesis will also use a BDI architecture.

2.4 Speech act theory

Speech act theory [6] [65] treats language as action, so that it can be planned just as any physical action can. In common with physical acts, speech acts have certain preconditions, and can be said to have effects in the environment. In contrast to physical acts, the effects are more in the mental state and subsequent actions of the hearer rather than a direct physical effect. A speech act is a composition of different acts. The locutionary act is the act of saying something, whereby the hearer becomes aware of an utterance. The success of the speech act depends on the success of the locutionary act. Associated with the locutionary act is an illocutionary act. There are different types, of differing "illocutionary force", often identified by different explicit performative verbs, such as "inform" or "request". To define illocutionary acts within the STRIPS planning system, Bruce [8], Perrault and Allen [51] and Allen and Perrault [1] developed a set of schemas. They defined a request act, whose precondition was that the speaker intends the act being requested, and whose effect was that the hearer intended the act. They also defined a family of inform acts, whose preconditions were that the speaker knew the proposition being informed and whose effects were that the hearer knew the proposition. These schemas are repeated here.

```
name:           request
parameter:      P
precondition:   intend(speaker,P)
```

```
effects:        { intend(hearer,P),
                  know(hearer,intend(speaker,P))
                }

name:           inform
parameter:      P
precondition:   know(speaker,P)
effect:         { know(hearer,P),
                  know(hearer,know(speaker,P))
                }

name:           informref
parameter:      TERM,P
precondition:   knowref(speaker,TERM,P)
effect:         { knowref(hearer,TERM,P),
                  know(hearer,knowref(speaker,TERM,P))
                }

name:           informif
parameter:      P
precondition:   knowif(speaker,P)
effect:         { knowif(hearer,P),
                  know(hearer,knowif(speaker,P))
                }
```

While this was a good first cut, there are some problems, the most important of which is that they ignore the autonomy (free will) [78] of the hearer in deciding whether to believe what is informed, and deciding whether

to intend what is requested. This is due to the use of effects lists. Appelt, in his logical formulation of the planning of speech acts, also uses hard-coded effects lists to realise effects [5]. A weaker and more acceptable set of effects is that the hearer only believes that the speaker believes P or that the hearer believes that the speaker intends P. Even these are too strong, since if an agent is allowed to insincerely request and inform, the hearer might not recognise sincere acts as being sincere, and so their effects would not occur. It seems that effects lists need to be done away with altogether, and this approach is successfully taken in this thesis. A second problem is the use of the term "know" instead of "believe" in the preconditions. For example, suppose agent A believes $\neg(P)$, while agent B believes P. While it is reasonable that agent B could inform agent C of P, the plan rules would not allow agent A to generate this expectation, since agent A could not say that agent B knows P. The preconditions should instead only refer to the speaking agent's belief. This principle will be adopted in designing the planner's negotiation acts. A third problem is that the effects ignore the nested beliefs of the agents. For example, an effect of the inform act should be that the speaker believes that the hearer believes P, and so on.

In speech act theory, the perlocutionary effect of an act is the state that the system reaches as a result of the act. While Allen and Perrault ignore the hearer's autonomy and define this using the STRIPS effects list, the perlocutionary effect in fact depends on the following choices of the hearer. For instance, the perlocutionary effect of an inform act depends on the hearer's choice of how to revise his beliefs, and what subsequent action he will take as a result. Grice [27] accommodates the hearer's response when he describes meaning as the "effect the speaker intends by the hearer's recognition of that intent". This description motivates the speaker's use of plan recognition and

response from the hearer's perspective when planning an illocutionary act, rather than computing the effect using STRIPS effects lists. Without considering the hearer's response, the speaker could not know the meaning of his utterance.

2.5 Planning and plan recognition

Plan recognition is the process of one agent inferring another's intention from the evidence of its actions, so that the inferring agent can act on that intention. Usually it is mutually believed that the inferring agent is observing the acting agent [11]. This is called intended recognition. In this case the acting agent chooses its actions with the expectation that its intention will be inferred and acted upon by the inferring agent. He may thereby choose a plan that is impossible to achieve by himself. Where the inferring agent does not believe that the acting agent believes that it is being observed, it must limit its inferences to those intentions that the acting agent would be able to act upon alone. This is called keyhole recognition.

Plan recognition models start with Kautz's [39] theory. He uses a decomposition chaining model for his plan rules, which he translates into logical rules. These rules can be used to produce explanations in disjunctive normal form. Following the same decomposition chaining planning model, Vilain [73] uses a chart parser to produce answers that match Kautz's theory. Statistical parsing [13] could possibly be applied to plan recognition, whereby each decomposition rule for a symbol has an associated probability of being used by the planner. Statistical parsing is used in parsing sentences to disambiguate those that have many parses, and by the same token may be used by the hearer to pursue only the most likely plan hypotheses. This would

be effectively a user modelling approach of the kind that will be taken in this thesis, since the probabilities would reflect the beliefs of the acting agent about their capability to use a plan rule, and about satisfied preconditions of the rule. Charniak and Goldman [14] describe a similar idea, by modelling the planning process with a Bayesian network. Each action is represented by a random variable. The top-level actions of the plans appear as the roots of the network, with different decompositions as their children. Each node in the network is related to its parents by a conditional probability table, which can be trained from data. For a given set of observed actions, the probability of different explanations can be found by following the conditional probability tables. Probabilistic plan recognition is important since in many cases there can be large numbers of unlikely yet possible explanations to a set of actions. For an agent to respond efficiently, it must be able to find out the most likely of those explanations. The same holds in dialogue planning, where with little evidence, the hearer must do the most he can to reduce the set of hypotheses, so that the speaker can be understood. Although the speaker should choose contributions to the dialogue that minimise the hypotheses available to the hearer, probabilistic reasoning allows the hypothesis set to be further reduced. A probabilistic approach to plan recognition is taken in this thesis.

Specifically for dialogue planning, Carberry [10] uses a tree model for plan recognition, called a context model, with parent-child relationships representing decomposition chaining and precondition-effect chaining. Following Grice's description of meaning, her theory encompasses a plan recognition step by the hearer, from which the hearer obtains the speaker's intention, and a continuation of the plan, whereby the hearer adds actions that satisfy the intention. She assumes that dialogues are planned in a focussed way [28]. If an agent is focussed, it will not open a subtree until all of the other subtrees

in the context model have been completed. Focussing is important since by limiting the number of continuations of a plan, it reduces the number of hypotheses that the inferring agent must consider, making plan recognition much easier.

2.6 Meta-level planning

Meta-level planning in dialogue distinguishes between the domain plan, and the dialogue plan whose subject is the domain plan. The domain plan is often, but not always, a plan of physical actions, whereas the dialogue plan is one that establishes the beliefs that are a precondition to some of the domain actions, or establishes the domain-level intention of the speaker so that a correct cooperative response can be given by the hearer. Litman [45] describes a planner that uses a stack of plans, with the domain plan at the bottom, and with each subsequent plan being a meta-level plan of the previous plan. The dialogue plans are constructed from a set of STRIPS schemas, which take the plan at the next level in the stack as parameters. The "continue-plan" schema is for executing the next focussed act. The "track-plan" schema is for using a dialogue act to declare the focussed act to the hearer. This helps the hearer to perform plan recognition, and provide an appropriate cooperative response. The "identify-parameter" schema is used to establish a speaker or hearer belief that is a precondition in the object plan. The "correct-plan" schema is used to introduce a new object plan when another has failed. The "introduce-plan" and "modify-plan" schemas are used to indicate a change of focus on the object plan. These are needed since otherwise, a focussed plan recogniser would discount the plan as breaking the focussing rule.

While Litman's meta-planning schemas are used to follow the structure of a domain-level plan, Ramshaw [54] uses meta-planning to also choose the domain plan. In a situation where there is an agent with limited knowledge of the domain state, and an expert user who has wide domain knowledge, the agent may need to explore several domain plans, consulting the expert about each one, and if a plan should thereby fail, it may need to backtrack and try an alternative way of constructing the plan. Ramshaw defines a "build-plan" action, which can be achieved by several different "build-subplan" actions, with backtracking over these alternatives. Ramshaw's planner backtracks because of failed preconditions. Some preconditions can be satisfied using y/n and wh- queries, named "check-pred-value" and "ask-pred-value". Some preconditions can be satisfied in many ways, for example, there may be ten different ships suitable for the agent's battle plan, each of which may be used to instantiate a parameter to a plan schema. The "ask-fillers" action is one of a family of actions used for this sort of query. Ramshaw suggests that a future direction for his planner is rather than seeking out the first workable plan, to use a quantitative comparison of competing plans in a negotiation dialogue. This suggestion is taken up in this thesis.

Smith and Hipp [69] describe a planner that is quite similar to Ramshaw's backtracking meta-planner, in that it too uses a backtracking search, controlled by a Prolog interpreter, to find a solution for a domain-level plan. Where Ramshaw's planner always uses system initiative, only allowing the expert to respond to queries, their planner allows differing levels of initiative whereby both agents can answer queries, ask queries, give unasked-for information, and introduce goals to different extents. This requires a richer model of the user than one of the passive knowledge-base used by Ramshaw. They use a model that encompasses problem solving capabilities as well as

domain-state knowledge. Problem-solving knowledge is encoded as Prolog rules, with some subgoals being achieved by rules that are within the user model. The system passes control to the user whenever these subgoals are invoked. They even give an example of a clarification where the user is given control for a subgoal, and then for a further subgoal the user passes control back to the system, thereby using a deeply nested user model.

2.7 Plan recognition and cooperative response

Plan recognition is a necessity both for understanding dialogue, and for generating dialogue, and the ability to do so is usually assumed by both the speaker and the hearer. In understanding, plan recognition allows the hearer to work with the speaker's intention, rather than just respond to an utterance. In generation, a speaker must predict the plan recognition process of the hearer, so that he can decide what needs to be said so that the hearer has enough evidence with which to recognise that intention. In planning a dialogue that is intended to be recognised by a hearer, there is a set of conversational maxims [27], which are related to the rationality and efficiency of the speaker's contribution in a cooperative setting. Speakers are expected to conventionally follow these. The maxim of quantity states that the speaker should be as informative as is necessary and no more. The maxim of quality states that the speaker should be truthful, so that he chooses the contribution that should lead to the satisfaction of his intention. The maxim of relation states that the contribution should be related to the current focus of the conversation. The maxim of manner states that the speaker's mode of expression should be easy to understand by the hearer. If a speaker assumes that the hearer expects the speaker to follow the maxims, the speaker

can attempt to apparently flout the maxims. This would force the hearer to search for an unobvious explanation for the speaker's contribution that does satisfy the maxims. For example, flouting the quantity maxim is a way of expressing disinterest in a topic, changing the subject, or of passing the initiative in the dialogue to the hearer.

The use of plan recognition for understanding dialogues was first described by Allen and Perrault [1]. By using planning rules, their system could take an utterance and perform planning in reverse, by matching the consequence of each rule to a given utterance and introducing plan hypotheses called alternatives from the different available antecedents. From these hypotheses, the plan rules were used to search for a continuation of each alternative. The system partitioned the plan rules according to the capabilities of the system and the capabilities of the user, so that "obstacles" could be identified. Obstacles are parts of the plan that cannot be achieved by the user, but can be achieved by the system. The response of the system is then determined by the obstacles for which solutions can be provided.

Allen and Perrault give several examples of how a cooperative response can be provided. In one, an agent asks a question, for which an answer is given, but as a bonus, the hearer applies the planning rules to recognise the speaker's plan, identifies a further obstacle involving lack of information, and provides further information without being asked. In another example, a speaker asks a yes/no question about a property of a referent. Finding that the answer is no, the hearer infers that the speaker may have been looking for a referent for which the property is satisfied, and so the cooperative response is to instead provide a suitable referent. In another example, the planning of an indirect speech act is explained. In this example, a speaker asks a question that is seemingly irrelevant to any plan, such as "can you pass the

salt?", flouting the maxim of relevance, until it is inferred that this question can be answered as a side-effect of another plan that is relevant, namely one in which the speaker wants the salt passed. While Allen and Perrault described the use of plan recognition in understanding an utterance, it must also be used from the speaker's perspective in choosing an utterance. For example, an agent might choose between the questions "what is the time and platform number of the Windsor train?", and "What is the time of the Windsor train?". In choosing the latter, the speaker must reason about the hearer's plan recognition process. The speaker expects that the hearer will recognise his plan and provide the information without being asked.

There are various devices of ambiguity in dialogue whose success depends on plan recognition. For example, a referring expression like "the ball" might refer to the red ball or to the blue ball. Suppose a speaker has just picked up the red ball. He can then say "Shall I pass the ball?" and expect that the hearer will recognise his plan and find that the only ball that can be passed is the red one. Grosz and Sidner [28] describe how this happens by referring to the agent's attentional state. The attentional state is the collection of actions and objects associated with the focus point in the plan structure. If an agent is focussed, its contribution must attach itself to the focus point, and therefore must refer to the attentional state. Anaphora can also be planned in the same way. Where a pronoun may refer to more than one agent, the hearer would be expected to select whichever agent allows a coherent and focussed continuation of the plan to be constructed [67]. Carberry [9] shows that by generating an expectation from a plan, ellipsis can be used to communicate sentence fragments, from which the intent of the speaker can be recovered by the hearer by matching the expectation with the fragment.

Grosz and Sidner [29] have formalised a logical model of the cooperative planning and plan recognition process. They describe the conditions under which an agent can infer that a plan is intended in which both the speaker and the hearer will act. This is called a shared plan. Assuming that when an agent recognises a speaker's intention, he will adopt that intention as well, the conditions are that it is mutually believed that the speaker intends the intention, and it is mutually believed that the actions in the plan are correct. If these conditions hold, then the mutual belief that each agent intends its part of the plan can be established. Shared plans are formalised as a set of inference rules on the mental state of the speaker. Shared plans are in some ways too strong to be useful. There are many situations in which mutual beliefs do not hold about intentions, due to differences of beliefs about the domain state and plan rules, yet the speaker can still form a useful plan. For instance, a dialogue planner may ask for milk in his coffee, but the hearer, not believing that milk is available, may well form a plan to ask him if cream is alright instead. In this example, a shared plan does not exist for the first utterance because the agents do not have mutual beliefs about the correctness of the plan. Such plans will feature regularly in this thesis, since the planner to be described is designed to deal with uncertainty and differences in beliefs.

2.8 User modelling in dialogue systems

User modelling is the representation, acquisition, and maintenance of a model of those components of the mental state of a user that determine his preference over different courses of a dialogue with the system. Typically, these are his beliefs about the state of the environment and the system, his capabilities for action within that environment, and the value he associates with

the achievement of goals [38]. A dialogue system with a good user modelling component will achieve the user's intentions more quickly, since it can infer those intentions without being told. Such a system would never tell the user the same fact twice, nor begin a subplan that the user is incapable of cooperating with, nor ask for too much clarification about the plan that the user is trying to construct. In the context of dialogue planning, the BDI model is the natural place to start for representing a user model. The intentions of the agent are a function of its beliefs and desires. The desires of the agent do not change during the dialogue, since the agent does not choose desires, he only chooses intentions. Intentions are not part of the user model but are determined by the system through the plan recogniser. Beliefs and desires determine many possible intentions, but only some of those possibilities will be consistent with the actions of the agent observed in the dialogue. A BDI user model will then have three components: beliefs, desires and action history, from which hypotheses about the intended plan structure can be inferred.

While much work on dialogue planning and plan recognition makes no distinction between the different beliefs of the agents, it is quite possible, in apprentice-expert dialogues or where each agent brings a complementary set of skills, that they will have differing beliefs about the state of the environment, and differing beliefs about the plan rules as well [53]. An agent may then construct one plan based on two different sets of plan rules. Plan recognition for dialogues of two or more agents must account for these differences.

For the model of dialogue planning based on STRIPS schemas, the basic mechanism of acquisition and maintenance of a BDI user model is through inference of preconditions and effects. Assuming that each agent is capable of recognising that each dialogue act has happened, it can then update those

propositions that were necessary to satisfy the preconditions of the action, and the propositions that were on the effects list of the action [52]. Another means of maintaining a belief model is by using stereotypes. For example, a dialogue system for flight bookings might see dozens of business users every day, who all have a similar belief state, and dozens of tourists every day who all have a similar belief state, which is quite different from that of the business user. Finding out whether the user is a business user or a tourist then allows the system to retrieve an appropriate stereotype model, constructed from previous dialogues, that best represents the user's state. Rich's GRUNDY system was the first to use stereotype models [57].

Nested belief models are those that represent the system's private beliefs, the system's beliefs about the user, the system's beliefs about the user's beliefs about the system and so on. The first model, the system's private beliefs, is referred to as level one, the second, level two, and so on. Beliefs that occur at all levels are referred to as mutual beliefs. Such beliefs are quite common, since if the agents mutually believe they are both observing the dialogue, any inferences drawn from the dialogue will be mutual. Many dialogues have this property, and so the system need only maintain a two-level belief model, since every second level is identical [72]. There are exceptions however. For example, the agents may be talking on a noisy telephone line, and agent A may assume that agent B has not heard what it said, whereas agent B may believe that agent A assumes it has been heard. This would form a discrepancy between level one and level three. Similarly, one agent may leave the room and perform actions that the other cannot observe. Clark and Schaefer [15] call the process of establishing mutual belief "grounding". Failure in communication in spoken dialogue is common, whether through disagreement about what was actually said, or through disagreement about

what can be inferred from what was said. Hearers must therefore follow up what has been said with some evidence that can allow the speaker to update his level three model, his level five model, and so on. For example, after dictating a telephone number, the speaker cannot say that the hearer knows what the speaker uttered. However, if the hearer repeats the number back to the speaker, the speaker can establish this belief. Once the level three belief is established, the speaker can expect that any plan that the hearer pursues that involves this number will succeed. Deeply nested belief models are also necessary when stereotypes are used. For example, one travel agent may offer extra legroom, whereas all the other travel agents in town offer no extra legroom. Therefore the user may believe that the system does not offer extra legroom. This would be a discrepancy between levels 1 and 3 of the nested belief model. As a result of this the system must refer to level 3, and make an extra effort to ground the fact that extra legroom is offered. The planner will therefore allow belief models nested to arbitrary depths.

There is a standard notation, introduced by [40], for describing the occurrence of propositions in a belief model. To say that a proposition P occurs at level 1 of the belief model, SBP would be used, meaning "the system believes P". For level 2, SBUBP would be used, meaning "the system believes that the user believes P". For level 3, SBUBSBP would be used. The notation MBP is used to say that a proposition P is mutual, that is, it occurs at every level in the belief model. It is often the case that a proposition occurs at every second level of the belief model. The expression SBMBUBP is used in this case to say that "the system believes that it is mutually believed that the user believes P".

2.9 Generating and understanding dialogues without planning

While symbolic planning is useful in generating dialogues, it is not as useful in understanding them. Dialogue systems that are programmed with a set of fixed rules cannot parse plans that are outside the limited set of plan structures specified by those rules. Input can fall outside the rules because the user is misconceived or because the rules only cover a fraction of the domain, and so the user may attempt correct, but out-of-scope plans. It would be useful for dialogue systems to learn the sequences of actions that occur in a dialogue from given dialogue corpus data, without any use of prespecified rules. This approach would have a second advantage, in that it performs user modelling as well, being able to find out the patterns of actions that a particular user or stereotype group prefers.

An analogous problem occurs in recognition of not action sequences, but word sequences in sentences. Statistical language models are much better than hand-crafted grammars in predicting word sequences, illustrated by their usefulness in speech recognition systems and in machine translation. This has led to a branch of research in dialogue understanding that uses statistical information about dialogue act sequences to better understand the user's utterance. Each dialogue act has a type and a propositional content. Deciding the type of the act is a classification task to which common machine learning algorithms can be applied, using various features of the previous dialogue acts, and the current utterance [63]. The dialogue act type can then fill one slot in a semantic frame for the utterance. Dialogue act classification has also been used in speech recognition, where a small gain in recognition accuracy can be obtained by applying language models that have been trained

on the expected act type of the given utterance [70].

2.9.1 Deciding dialogue strategies using policies

It is clear that human speakers do not derive a dialogue plan from first principles every time they must think of something to say. Often, situations arise that have been seen before, and the speaker needs only to recall his policy for the situation. For example, flight-booking dialogues would often have states in common. For many planning problems that involve long sequences of actions on problems with limited numbers of states, planning from first principles can be inefficient since the agent must search over several action alternatives for each of many steps in the plan. On the other hand, by using reinforcement learning, a learned policy provides a compact record of the solution to every problem instance. To design a system that uses a policy, the designer must first specify a set of states. For example, in a flight booking system that fills a frame of information from the user, the frame states would form the state space. For each state, the set of actions is defined, and a state transition function specifies the state that must result from applying an action in a state. Different actions could represent different strategies. For example, there might be a system-initiative strategy or a user-initiative strategy, or different confirmation strategies for different states. Learning a policy is a matter of evaluating utility for a state and an action by looking up the utility of the resultant state and adding any reward gained from the current one. By iterating this learning rule many times over the each state in the state space, a table of state action pairs and utility values converges on the optimal policy for the problem by passing back utility values from the outcomes of the dialogue. Using a policy, planning becomes a trivial matter of looking up the table to decide the best action to take in a state.

Using a policy has an advantage in that the true reward of dialogues can be used in training the system. On the other hand, by planning the dialogue, the reward obtained is estimated from the value of the goal and the costs accrued by each of the actions in the plan. However, the downside of using a policy is the cost of exploration to obtain training data. In reinforcement learning, an agent must strike a balance between exploration and exploitation. Using softmax selection [71], an agent can try actions that are not currently optimal, in order that examples can be collected to reinforce that action. As more and more examples are collected, the agent tends to exploit the optimal action rather than explore. Softmax selection is also useful in cutting down the complexity of the state space, which might be equivalent to the combinatorial composition of the states of many beliefs. One good example of using dialogue policies is that of Walker et al [76]. They use the PARADISE evaluation framework to compute the utility of the dialogues.

Using reinforcement learning is an attractive approach to deciding dialogue strategies, and it is important to contrast this approach with the planned approach that will be taken in this thesis, since both can adapt themselves to users by training on their dialogues. To be competitive with reinforcement learning, two qualities are important. First, the planner should be as easy to use as a reinforcement learning system, and in the next chapter this will be shown to be the case. The second quality is its performance, that is, the quality of the dialogues produced given a certain amount of training material. While reinforcement learning is very useful for problems with limited numbers of states, which are well covered by training data, planning is useful where there are many more states, leading to the problem of making a good decision in novel situations. The arguments for and against planning and reinforcement learning in robot planning carry over to dialogue planning,

especially where the dialogue is a non-routine dialogue such as a meta-level negotiation over a robot plan.

The model used for basic reinforcement learning is that of the Markov Decision Process (MDP) in which the state transition function is assumed to be deterministic. Just as in robot planning, where actions and observations are uncertain, dialogue planning must accommodate uncertainty since errors occur in the speech recognition process. For this reason, several researchers have addressed the use of Partially Observable Markov Decision Processes (POMDP) in dialogue planning. In a POMDP, actions have a probability distribution of effects, and states result in a probability distribution of observations. Since the agent does not know which state it is in, reinforcement is more difficult, and POMDPs can be difficult to train. Roy et al [58] , used a POMDP to deal with speech recogniser uncertainty in a speech-controlled robot, showing a significant improvement in performance when uncertainty in the belief state of the robot is accommodated. Zhang et al [79] address the state complexity issue by using a Bayesian Network to map several state variables into one.

2.10 Game theory

Game theory [47] describes the mathematics of individual decisions made by several agents in a shared environment. Each agent has a number of alternatives from which to choose. The utility obtained by each agent depends on the combined choice of all of them and so a matrix is used to specify the utility obtained by each. In general, the agents are self-interested in that there is no particular relation between their individual utility functions. However, in the application of game theory to fully cooperative dialogue it happens

that their utility functions are the same, meaning that they favour the same outcomes. The objective of each agent is to make choices that maximise its expected utility. Where agents make a sequence of decisions, a game tree can be used with a node representing each decision. It is possible to calculate an equivalent matrix for any given game tree. The most interesting games are where agents choose simultaneously, or do not immediately find out the other agents' choices. However, in this thesis, agents take turns to make their choices which are immediately observed, and so all that is required is to choose the one that will maximise the agent's utility.

Game theory will be used in this thesis to provide a quantitative aspect to the dialogue planner's choice. Traditional dialogue plan rules are used to generate the alternatives available to the agent, but these alternatives then form a game in which the agent chooses the alternative with the maximum expected utility.

The area of Bayesian games is particularly relevant. Bayesian games generalise standard games to those of incomplete information. In a standard game, every agent knows everything about the alternatives available to the agents and their utility functions. In a Bayesian game, such information is probabilistically known to the agents. Harsanyi [31] showed that uncertainty about applicability of actions can be modelled using the utility function by just using very negative values for those actions. Then, Bayesian games need only be concerned with uncertainty about the utility function. Each agent has a type, which determines its utility function. Each agent uses a set of nested beliefs about the type of the other agents. These beliefs take the form of probability distributions over types. To calculate the utility of an alternative, the agent needs to evaluate the expected utility over the types. This calculation is much the same as that used by the dialogue planner described

in this thesis. Instead of using beliefs about types, the planner directly models the alternatives available to the agent through STRIPS preconditions, whose satisfaction is determined using beliefs about the domain state.

Gmytrasiewicz and Durfee [25] have developed a method of computing the utility of games using a probabilistic model of belief, which has its foundation in Bayesian games. While the planner presented here was developed initially without knowledge of this work, it has close similarities. They use a "Recursive Modelling Method", which represents the game in canonical form, that is, as a game matrix. At the root of a tree is a complete matrix, specifying all of the alternatives available in the game. At each node, a belief is taken, and a pair of edges are annotated with the probability of each value of the belief. For each edge, a child matrix is constructed, in which alternatives whose preconditions are disabled by the belief have had their row removed from the matrix. By performing a weighted sum over the leaf matrices in the tree, the expected utility of each alternative can be found. While the Recursive Modelling Method is equivalent to the game trees that will be used in this thesis, it does not provide any way of constructing an RMM tree from plan rules, nor is it clearly explained how the child matrices may be obtained from their parents by consulting the nested belief model. The emphasis of the RMM is on applications in military strategy, using gathered intelligence to inform the nested belief model. They have performed some experiments with human subjects, and found good agreement between the strategies chosen by a human player, and by the RMM.

2.11 Cooperative distributed planning

There is some work in the planning field that addresses the distribution of planning control among multiple agents. A number of cooperative distributed planning (CDP) [18] algorithms have been proposed that attempt not just to tackle planning problems that will be executed by a number of coordinated agents, but to distribute the planning process as well. The planning process often needs to be distributed for reasons of speedup through parallelism, fault tolerance, communication bottlenecks, distributed planning knowledge, perception of the environment state that is localised to only some of the agents, and localisation of execution to the place where it was planned. There are generally three planning processes for these algorithms that determine the dialogues that take place between the agents. First, task distribution must find a way of allocating portions of the global plan to individual agents. Then, each agent pursues development of their own subplan. Finally, the agents must communicate their choices so that the subplans can be critiqued and modified. For distributed execution, constraints between subplans can be handled by using synchronization messages that control multi-agent execution.

The earliest CDP planner was Corkhill's [16] distributed version of Sacerdoti's [60] hierarchical planner Noah. This planner used critics which were invoked at each level of abstraction, to incrementally check for coordination of the developing plan. Later, the Partial Global Planning (PGP) [19] system was developed for a distributed vehicle monitoring application. This system could decompose tasks, and then hold negotiation dialogues to identify conflicts and opportunities for coordination by each agent passing their localised view of the global plan.

CDP differs from the planning approach that will be developed in this

thesis. It is more useful when plans can be pursued for the most part independently with only a few coordination points. Furthermore CDP does not have anything to say about the efficiency of coordination dialogues, since it is assumed that communication is relatively inexpensive. Nor can CDP be easily adapted to account for communication cost. This is because the planning approach taken in CDP emphasises the exchange of proposed plans rather than exchange of the beliefs that are used to form those plans. The plan space grows exponentially with the number of beliefs about the domain state and the plan rules, and so the number of plan proposals that must be exchanged is exponential as well. Conversely, exchange of beliefs and is more more efficient from the perspective of communication costs. In addition, the probabilistic approach to planning that will be taken here allows an agent to prune the space of multi-agent plans of the unlikely alternatives, so that the likelihood that a subplan is easy to coordinate is increased. Conversely, CDP makes no distinction among the plans that the other agent would chose to put forward. DSIPE [77] is one exception. It deals with the information overload problem by filtering out the constraints that are unlikely to be relevant to another planner.

2.12 Dialogue management and user modelling systems

Dialogue systems are are most often composed of several modules. The most common organisation is that of a dialogue manager, a speech recognition system, a parsing and semantic analysis system, and a generation and text-to-speech system. The dialogue manager is used to maintain a representation of the dialogue state and to decide a strategy at each turn. The speech recog-

nition, parsing, and semantic analysis systems are used to produce a formal representation of the user's utterance. Often a semantic frame will be used with slots for the act that the user is performing, and for the parameters to this act. For example, the act might be labelled "send-email" with parameters for the recipient and subject of the email. In the case of planned dialogue based on speech acts, a semantic representation should be composed of the propositional content of the sentence, represented using predicate logic, and the illocutionary type associated with the utterance, for example the sentence might be informing or it might be requesting. The same formal representation of the utterance would also be used as input to the utterance generator. A set of rules would be matched against the parsed formal representation to construct and assemble natural language phrases corresponding with the phrases of the formal representation.

Three general types of dialogue manager exist [62] [49]. State based systems are the simplest. The dialogue is modelled as a sequence of states, with the strategies chosen by the user represented by the transitions. State based systems therefore have a hardcoded dialogue strategy and no user model. Another more flexible design for a dialogue manager is one based on a frame representation of the dialogue state. Frame-based systems are useful for the style of dialogue where a number of information items must be elicited from the user. The strategy is more flexible than in state-based systems since the system can dynamically form strategies by checking the slots that still need to be filled. For example if the name and credit-card number are still not known, the system could formulate a single question to elicit both. The most complex design for a dialogue manager is that of an agent based system, which uses an explicit model of the system and the user in terms of of a belief, desire and intention (BDI) architecture (see Section

2.3). These systems choose dialogue strategies by planning.

One of the objectives of this thesis is to create a dialogue planning system that is domain independent, in that it acts as a shell that supports execution of a set of dialogue plan rules, and automatically maintains of the user model that is used in generating the dialogue. No changes should be necessary to the system to support a new set of dialogue plan rules. A number of dialogue systems already exist that have a similar objective. Based on a finite state design, VoiceXML [46] uses an XML description of a finite state machine. For each state, a set of transitions is given to correspond with each of the user inputs. These inputs are specified using a grammar. VoiceXML is intended as an analogue to HTML, so that forms can be filled using voice rather than a web browser. Just as web pages are served from a web server, voiceXML pages are served from a voiceXML server, which runs the automaton, generates speech output and interprets speech input, and returns the information gathered in the dialogue. Another domain independent system that supports dialogue management is the BGP-MS system of Kobsa and Pohl [40]. This constitutes a user modelling shell system that stores and infers user beliefs and goals. It provides a protocol through which an application program can feed beliefs about the user and reports of the user's actions. The system, working on a specified knowledge base can pass interesting inferences to the application system, search for misconceptions, and formulate questioning strategies for the application system which are used to acquire the user model. Using action specifications, inferences are drawn about the preconditions and effects of the dialogue acts that are observed in the application's interaction. A mechanism for resolving inconsistency in the user model has been proposed for the system, since user beliefs can change or be misconceived. Stereotypes can be used, allowing inheritance of user

models from a stereotype model to each of the members of the stereotype. Kass and Finin [37] also developed a user modelling shell, GUMS, which has much the same set of features as BGP-MS.

COLLAGEN [56] is a dialogue planning system that is used in managing a collaborative process between a user and a separate agent. While it does not choose strategies, it instead operates as a mediator between the two dialogue participants, recording the dialogue history and parsing it into a plan structure. The dialogue is modelled using hierarchical plan rules, and assumes that acts are added to the plan in a focussed manner, in a similar fashion to the model employed by Carberry (Section 2.5), and similar to the model that will be used in this thesis. Since COLLAGEN records the structure and changing focus of the dialogue, it can help the user in a number of ways. First, it can display the dialogue acts from which the user can choose at a point in the dialogue, by checking the applicable plan rules at the plan's focus point. The user can stop an incomplete plan so that the focus point can be moved somewhere else. He can return to stopped points later on. A plan can also be abandoned, by backtracking and taking a different alternative at an earlier choice point. Segments can be replayed allowing their reuse in different contexts. COLLAGEN separates the generation of allowable strategies from the discourse model from the choosing of those strategies by the agent. A similar principle will be used in the planner presented in this thesis, where the allowable strategies are first generated, and a separate module is used to choose a strategy from those alternatives.

The TRAINS system [2] is an example of a natural language collaborative planning system. It is a kind of meta-level planning system (see Section 2.6), in that the planned dialogue is one that supports the choice of a domain-level plan. A human planner uses the system to answer questions about

the domain and to evaluate proposed alternatives. The architecture of the TRAINS system is agent-based, and in common with the planner described in this thesis, uses the BDI model to represent the state of the dialogue manager. Using this model the system maintains a set of nested beliefs about the user, so that as the dialogue progresses, a model of the user's domain plan alternatives can be developed, and the system can provide cooperative contributions to the dialogue in the context of these alternatives. TRAINS is a complete natural language system, addressing the challenge of understanding natural user input that is relatively unconstrained due to a mixed-initiative dialogue strategy.

Walker [75] discusses BDI planning using Bratman's IRMA [7] architecture. IRMA is designed to accommodate an agent's resource limitations in performing deliberation every time a change is made in the environment. Walker explains that a substantial fraction of dialogue contains redundancies - information that needs to be communicated only because the hearer's resource limitations prevent him from inferring it himself. A dialogue partner may be limited in working memory or may be limited in inferential capacity. A resource bounded BDI architecture is useful for generating dialogue with redundancies, since it can be used as to model the deliberation process of the hearer.

2.13 Evaluating dialogue systems

The value of a dialogue system is determined by the purpose for which it was built, and there can be a wide range of these. Often, systems are task-based, in that there is a definite goal whose achievement is the system's main purpose. A task-based system is expected to achieve that goal in a

manner that uses few resources. Since dialogue acts only consume the time of the system and the user, the objective is really just the time taken to execute the dialogue, balanced against the reward obtained by achieving the goal. Task completion and execution time are both objective measures, that can be determined by the system itself, allowing for self-training or planning with these measures as the objective. Happily, task-based systems are the kind that are examined in this thesis. Apart from execution of a task, dialogue systems can be built for other purposes. For example, the well known Eliza system that attempts to simulate a psychiatrist, is not related to any task, yet users seem to appreciate the system in producing intelligent dialogue. Similarly, a system to produce weather forecasts would need to be evaluated by a mix of objective measures and human judgement, in producing informative, error free, and pleasant dialogue. Without a task model, it is more difficult to tell why the system is of use to the user.

Unfortunately, human judgement is often the final word in acceptance of a dialogue system, and it is often not perfectly determined by objective measures of the system. Even when there is a relation with objective measures, it can be irrational. For example, a system that exhibits frequent speech recognition errors due to the use of a freer user-initiative dialogue strategy might be perceived as of poor quality, even though the dialogue strategy improves the execution time of the system. As a result, human judgement, which is comparatively expensive, must often be used in combination with objective measures.

One attempt to relate objective measures to human judgement is the PARADISE framework [74]. This framework uses a set of objective measures of the dialogue, such as task completion, execution time, response time, and number of errors. In user trials, a quantitative measure of performance is

obtained from human judges. It is then supposed that this performance quantity is a weighted sum function of the objective measures. Using the set of user trials, a weighted sum of the objectives is equated to the judge's value. The set of weights that minimises error over the trial data is then obtained. It is interesting to look at the weights obtained, and this was done for two dialogue systems. It was found that task completion, response time, and elapsed time for the dialogue generally obtained the greatest weights. These experiments also showed that users irrationally value recognition accuracy over elapsed time, even though they ought to value only task completion and elapsed time.

Where suitable objective measures can be found for a dialogue system, automatic training of the system becomes possible without the need for human judgements. For example, a reinforcement learning system would be able to train on dialogues with users by calculating the measures at the end of each example dialogue. For a planned approach to dialogue, there is a further requirement of the objective measures that they be compositional over the plan structure, in that the measure for a plan is equal to the sum of measures for the acts in the plan. This is because a planner, in contrast to reinforcement learning, searches for plans, rather than reinforces plans that appear in the training data, and so that planner must be able to predict the value of the plans in the search space. This could be a disadvantage, but the measures given in the previous paragraph are compositional in that they are additive over the acts in the plan structure. Task completion is a function of the plan structure that is obtained at the close of the dialogue. Response time is a function of act that the system chooses at its turn. Elapsed time is the sum over the system and user acts of the time taken to execute each dialogue act. This might be easy to predict from measures such as the num-

ber of words used in the utterance that corresponds with the act, or from recorded examples of the act's use in a real dialogue.

2.14 Conclusion

This chapter has described work that is both related to and that directly forms the foundation for models of dialogue planning, and for the dialogue planner that will be developed in the remainder of this thesis. The main areas of emphasis were planning and plan recognition, user modelling, and dialogue management systems. In all of the theories of dialogue planning, and in most of the planning and user modelling systems described in this chapter, there has been an emphasis on planning as something that produces a set of possible plans that can achieve a goal, without comparing the utility of the different alternatives. Plans are treated as merely valid or invalid, and beliefs are modelled using logical rather than statistical models. The only exception is the systems based on reinforcement learning, where different strategies are compared according to a quantitative metric. However, while reinforcement learning can be used for many of the same problems as a dialogue planner, its approach is one of brute force, using only raw statistics to decide between strategies. On the other hand, dialogue planners seek to explain the dialogue by inferring the mental state of the speaker, inferring from the data not just that one strategy is better than another, but determining the belief state that explains the choice of strategy as well. While such a planner, which combines the best of both worlds, is desirable, there is currently no satisfactory planning model or implemented system. In the next chapter, a dialogue planning model, and a design for such a system is developed. By using game theory, and particularly by following Harsanyi's model of the

Bayesian game, the planner retains the existing theory of dialogue planning, but uses the alternative plans that are generated according to this theory as the alternatives in a game. As a result, a planner is obtained that can choose efficient dialogue plans.

Chapter 3

Design of the Planner

3.1 Introduction

This chapter describes the desired functionality and the resulting design of the planner. The chapter begins by setting out requirements for the planner, which are chosen such that the motivations set out in chapter 1 are satisfied. It will be argued that a planning system based on the evaluation of Bayesian game trees satisfies the motivation of producing a planner that produces plans that are more efficient than those planners that have come before. Then some assumptions about the planning model, about the rationality of both the user and the planning agent, and about the cooperative setting of the dialogue are made. The design of the planner is then given. The structure of the planner's state is defined, the algorithm that computes its strategy is described, as are the algorithms that revise the agent's state as the dialogue progresses. A simple example will be used to illustrate these algorithms. The reader might be interested in looking at the Prolog source code for the planner, which is available on the world wide web at http://planeffdia.sourceforge.net/.

3.2 Requirements

In chapter 1 it was stated that current dialogue management systems do not take advantage of a user model in deciding their dialogue strategy, and an objective was set to construct a dialogue system that was demonstrably more efficient than current systems by virtue of the use of a user model, yet not much more difficult for the system designer to use than those existing systems. A user model would typically be based on the BDI model, recording the user's beliefs, desires and the dialogue history so that inferences can be made about the user's likely preferences (see section 2.8). The system should also conform to current theories of planned dialogue, so that the user can understand the system's plan, and symmetrically, so that the system can understand the user's plan.

In order that the planner conforms to the current theory of dialogue planning, a planning model must be developed. One of the most popular ways of specifying dialogue plan structure is by a dialogue grammar, consisting of hierarchical plan rules. In a system that does not use a user model, there is no consideration that the acts in the plan might have preconditions that are evaluated in the context of the mental state of the agent. Instead, each plan structure and resulting dialogue outcome is equally permissible, no matter what the beliefs of the agent are. In some obvious circumstances, such a planner will fail to be efficient. For example, a user model might be used to discount a plan in which an agent will not be able to answer a question. This failure would be due to a failed precondition of the informing part of the questioning plan, which needs to be evaluated in the context of the user's beliefs. A system with no user model would have no way of determining that the plan is bound to fail. This model of hierarchical planning with preconditions is in keeping with the plan construction theory of Carberry

[10], and therefore it is adopted for the planner. Hierarchical planning is powerful enough that any set of finite-state rules traditionally used in finite state planners has an equivalent set of hierarchical rules [43]. From the above example, it is clear that a user model will improve the performance of the planner, and that its presence coincides with the use of preconditions to dialogue acts. Furthermore, the user model should be nested, since the agent's own beliefs about whether a precondition is satisfied may differ from those of the user, and may differ again from the user's beliefs about the system's beliefs. The plan under consideration will therefore be evaluated differently from the perspective of each agent, resulting in a different choice of contribution to the plan for each perspective [53]. Therefore the system needs to be able to evaluate the plan from as many perspectives as there are levels in the belief model.

Different agents can have different beliefs about plan rules as well as about the domain state. This is less the case with routine rules such as a question and answer pair, but more the case for meta-level plans whose subject is a domain plan that is constructed using different rules. As a result the different agents will have different expectations about the course of the meta-level plan. The planner is required to treat plan rules as subject to belief in the same way as the domain state is. Alternative ways of constructing a plan will then be subject to preconditions about the correctness of the plan rule used. These preconditions can be evaluated in the same way as action preconditions. Often in BDI models such beliefs about plan rules are called "capabilities". The use of plan rule beliefs opens the possibility of planning dialogues between an expert teacher and a novice student, or one between cooperative experts who exchange information about their expertise so that each can form a correct plan. With different beliefs about both plan rules and

the domain state, it becomes clear that when the agent takes the perspective of the user (or the perspective of the user's model of the system and so on), each perspective will look upon a different plan structure.

3.2.1 Treating dialogue as a game

One way to solve the problem of these different perspectives of the plan is to view the outcome of the dialogue as a sequence of acts, where at each turn, the perspective of the acting agent is taken. The acting agent's beliefs about the plan rules and the enablement of preconditions should be used to generate the set of available acts that can be taken at the turn. These acts should be the first steps in the plan structures that correspond with the beliefs. Since there can be many permissible sequences of acts due to the many alternative acts at each turn, an appropriate representation of the dialogue possibilities would be a game tree, whose nodes are constructed from the alternating perspectives of the agents participating in the dialogue. The game tree is not really a plan, but rather a representation of the possible outcomes of the iterated process whose steps consist of plan recognition, planning of the first step in the continuation of the recognised plan, and execution of that step. This process is symmetric, and alternates between the two agents. It can therefore be used by the agent to represent an expectation of the ultimate outcome of each of the alternative acts that it can choose from at the current turn. For an agent to generate a game tree, it must plan forward by alternately using its level 1 beliefs, and its level 2 beliefs, since these provide the expectation of its own act and its expectation of the other agent at each turn.

3.2.2 Using a probabilistic belief model

In section 2.3, a logical model of belief was described, where an agent will regard any given proposition as either "believed", "disbelieved", or "possible". Using such a model agent A might come to believe that B believes P, should B inform A of P. For many dialogues such a model is adequate, and indeed most of the work seen in dialogue planning and user modelling described in chapter 2 rests upon a logical model of belief. However, more generally, it is better to know whether a belief is probable rather than merely possible. For example, if an agent were to plan a questioning dialogue, it might risk failure of the plan if the probability of the other agent not knowing the answer were small, but at some point the risk would be become large enough that the cost of trying the plan exceeds its expected reward. On the other hand, the logical model of belief cannot help to make such a judgement. A second advantage of a probabilistic model is that it can be used to estimate the beliefs of a population of users as a stereotype model. For example, if three of five users knew the answer to the question in the past, a reasonable estimate of the current user's knowledge is that he will give the answer in three of five outcomes. Similarly, if just one user sometimes holds a belief and sometimes does not, a probabilistic model is useful to form an expectation of that one user. The system can then make decisions based on the expected beliefs of the user.

A supposition is made now that the use of a probabilistic belief model offers significant gains in the efficiency of the dialogue, when expected utility is the objective. For this reason, a probabilistic model will be developed, and to verify this supposition, demonstration problems in the following chapters will be used to measure the efficiency of the dialogues obtained by a probabilistic planner.

3.2.3 Bayesian games

The planner must be capable of examining the actions that will be taken in different belief outcomes. For example, in a questioning dialogue, the belief outcome of the questioned agent determines the answer it is expected to give. Some representation of the set of belief states, and the plans that occur in each of these states is needed. In section 2.10 the theory of Bayesian games was described. In this theory, a probabilistic model of belief is used to determine the efficient strategy in a game by examining the applicable actions in each belief outcome, and so the Bayesian game model is appropriate. In evaluating a Bayesian game using the maximum expected utility rule, the utility of each strategy is determined in each of the belief outcomes of the user, and a weighted sum of utility is taken over those outcomes according to their probability. In each of the belief outcomes, a different game tree is obtained, since in different belief outcomes, different alternative acts are believed to have satisfied preconditions. The central design element of the planner is therefore an algorithm that can perform computations with Bayesian games.

In a game tree, the permissible alternatives available to an agent correspond with the edges at a node corresponding with that agent's turn in the dialogue. Those permissible alternatives are determined by the rules of the game. In keeping with the requirement that the dialogue is planned using hierarchical plan rules, the second design element becomes clear. A planner is required that can plan a step in a dialogue according to a given set of beliefs about the domain state and about the dialogue plan rules, and for each candidate step, an alternative be supplied to the algorithm that generates the game tree.

3.2.4 Maintaining the user model

An objective set out in chapter one was that the planner be easy to use, with the dialogue system designer needing only to specify the dialogue plan rules. This leads to the requirement that the user model be initialised and maintained automatically by the system. The system should need only to look at the preconditions of the plan rules to know the set of beliefs that need to be held in the user model. Corresponding with this requirement, another main design element is that of a belief revision system, that can observe the dialogue acts chosen by the user and infer from these an explanatory belief state of the user. As dialogue evidence accumulates, it must be compiled into a statistical model of the user's belief state. Over time, as the accuracy of the model improves, so should the efficiency of the dialogue. As well as being used after each turn in the dialogue, belief revision must be used in evaluating game trees in the context of a belief model. For example, a plan in which an agent informs the same fact twice is discounted by using belief revision to update the belief model after the first inform, and then finding out that the second inform has no effect on the outcome of the dialogue, since the fact is already in the belief model. The belief revision system should be derived from those of the user modelling shells described in section 2.12, whose main function was to keep a user model up to date based on dialogue evidence, and using dialogue plan rules. The system will need to go further than these systems however, since they are based on a logical model of belief. Using a probabilistic model, probabilistic rather than logical inference needs to be considered. This entails representation issues as well, since in a probabilistic model, conditional probability using belief networks [50] is used in deduction rather than modus ponens. In contrast with logical models in which belief revision [23] is used to maintain consistency of the belief set, a probabilistic

form of belief revision is required. In a logical model, beliefs are absolute and so their update is normally based on a single observation. On the other hand, in a probabilistic model, a set of observations is used to calculate a probability value. Some mechanism is required to maintain beliefs using sets of observations.

3.3 Assumptions

In this section some assumptions are set out about the game model which is used by the planner to choose efficient strategies. These serve to restrict the required scope of the planner's capabilities, while being reasonable assumptions about the conditions in which a dialogue takes place.

3.3.1 Agenthood

Following the theory of two player games [47], it is assumed that two agents each select a strategy from a number of alternatives available to each. Different combinations of strategies are assigned different utility values with respect to each agent. A mechanism is required whereby each agent chooses the strategy that maximises its expected utility.

The game is organised into turns, in which each agent selects and reveals its strategy to the other. An assumption that simplifies the decision process is that of perfect information, where each agent finds out the strategy chosen on the previous turn before making its choice. This is reasonable since in most dialogues the environment is immediately observable to the other agent, and most agents wait until they have recognised the other's last act before proceeding with their one. Without this assumption, there are games where one agent cannot settle on one particular strategy, since if it did settle either

of them, there would be a worse alternative which the other could choose. The agent is therefore forced to adopt a randomised "mixed-strategy" [47]. However, with the assumption, the choosing agent may use a simpler process of just taking the maximum utility strategy at each node in a game tree.

3.3.2 Cooperation and sincerity

In all of the examples presented in this thesis, the agents use a shared utility function, since the computer acts as a fully cooperative assistant to the user and so holds the same preferences over outcomes. This guarantees that acts are recognised as having sincere intent. For example, an informing act whose precondition is that the speaker believes the told proposition would not usually be chosen if the precondition failed. If it were to be chosen, it would lead the hearer to a state of false belief, which would lead him to choose an irrelevant and therefore usually worse strategy. Since the utility function is shared, this would also be a worse strategy for the speaker. There are devices such as the "white lie", where the false belief happens to lead the hearer to a better strategy. For instance, saying it is cold when it is not is a good way to get someone to close the window. White lies should be plannable in the system, by means of a sincere and an insincere version of the plan rule for the informing act. However, regarding acts as being sincere reduces the number of hypotheses that the hearer must entertain about the speaker's intent, thus reducing the breadth of the game tree without reducing the abilities of the system for any of the problems explored in this thesis.

It is not difficult to generalise the system to be used for problems in which agents are self-interested rather than fully cooperative. It only requires that each agent is given its own utility function, and that agents are allowed to make choices that are insincere by dropping the belief preconditions. One

specialisation of this is the zero-sum game, in which the utility functions of each agent sum to zero. In this instance, it turns out that while the agents must choose some act, it is never of benefit to choose a spoken act. This is because a spoken act would do nothing but reveal the agent's mental state, which would if anything be advantageous to the opponent, and so with opposing utility functions, a disadvantage to the speaker. Such behaviour can be found in games such as poker, where players must act by moving cards, but minimise any other acts that would reveal something about their mental state.

3.4 Design of the planner

In this section a description of the design and operation of the planner is given.

3.4.1 Overview

Figure 3.1 is an overview of the planner. The state of the system is made up of the **beliefs**, and a record of the **dialogue history**, which is simply a list of acts. The choice of these state components was motivated by the structure seen in BDI architectures, where from the beliefs, desires, and dialogue history, each agent's intention can be inferred by plan recognition (see section 2.8). The processes of the system are the **planner**, which constructs the game tree from the act alternatives, and the **evaluator**, which evaluates the tree in the context of the agent's belief state. These processes together are used to choose the most efficient dialogue strategy given the agent's belief model. The **belief revision** process is used to revise the agent's beliefs in the light of executed acts as they are added to the dialogue history.

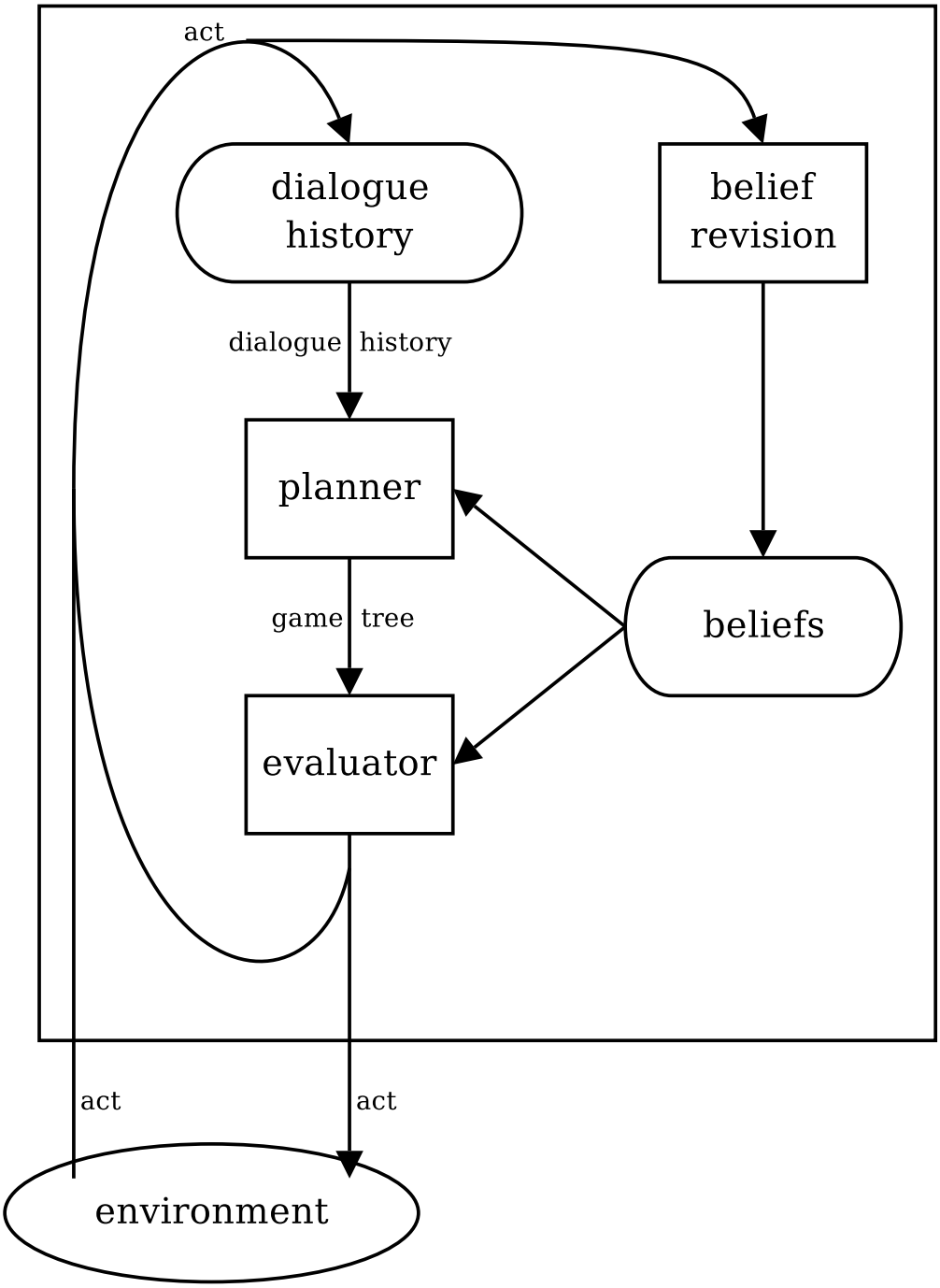

Figure 3.1: Overview of the Planner

3.4.2 Agent state

The system performs a cycle of deciding and executing a strategy for a turn for a given mental state, and then updating that state using belief revision, and updating the dialogue history. On the user's turn, the cycle consists only of invoking the belief revision module and updating the dialogue history. The mental state consists of:

- **Desires** The desires of the agent are represented by intention rules, which give a probability distribution over parent intentions for a given plan. These can be used to describe intentions that are not explained within the plan that the agent is generating, but which are the starting point for a plan. For example, a travel agent may find that most passengers have a fixed intention to have a window seat in an aeroplane, and so do not seek to explain or change that intention within the dialogue. However, when the agent's plan is being recognised, this intention is added as a parent to the plan structure.

  While the intention rules describe a distribution over starting intentions, a utility function gives a value to each plan. It should be used to evaluate the different outcomes at the leaves of the game tree, producing a number, which is then used to compare different alternatives in the game tree. In section 2.13 it was stated that in planned dialogue, a compositional evaluation function is required that can be used on every element of the plan search space. Here, the two most important evaluation measures are taken, task completion and execution time, although others such as speech recognition error and response time are also compositional and can be used in the same way. Since evaluation is compositional, a recursive search of the plan structure is required that

adds up all of the utility contributions of the acts in the plan. For task completion, the presence of the goal node in the complete plan is an indication of success, for which a positive utility is gained. Therefore the designer specifies a reward and goal node for the system. Each dialogue act is assumed to have some constant cost in terms of execution time. Therefore, the designer needs to specify this number for each of the terminals that appear in its plan rules. The system should search the plan structure automatically, using those given numbers. Finally, a weighted sum should be used, as in the PARADISE framework, to obtain an overall utility for the dialogue.

- **Nested belief model** The nested belief model is a numbered list of belief sets, where level 1 represents the acting agent's own beliefs. Level 2 represents its beliefs about the other agent, level 3 its beliefs about the other agent's beliefs about the acting agent, and so on. Only a finite number of levels is required, and this number equals the number of turns required in the dialogue. The utility functions of the agents ought to be nested as well, but for simplicity, they are assumed mutual, and therefore are outside the nested model.

- **Belief set** Each belief set is a set of propositions, paired with probability values. The probability value represents the estimate of the probability that the agent at that level holds that proposition. For example, a belief $p(prop, 0.5)$ at level 3 represents the speaker's estimate of the hearer's estimate of the probability that the speaker holds the belief *prop*. Level 1 is distinguished from all others in that it represents the agent's own beliefs, which are not estimates. They therefore have probabilities of 0 or 1 only. The use of a probabilistic model of belief,

rather than a logical one, follows from Harsanyi's model of the Bayesian game (see section 2.10), which allows for probability distributions over an agent's beliefs. By using a probabilistic model, the agent's estimated belief can be used. Typically, this estimated belief is informed by the evidence of previous dialogue, from which the frequency with which the belief occurred determines the probability value.

- **Propositions** Each proposition is of a particular type. There are domain propositions which represent the state of the environment. These propositions are just atomic propositions of predicate logic. There are plan rule propositions which represent the agent's capabilities - the plan rules that the agent believes are applicable, their preconditions and their effects. There are plan recognition rules which represent the probability that the agent intends a parent act given that a child act has occurred.

3.4.3 Representation and construction of the game tree

To decide a strategy on the systems turn, a game tree is constructed, and then evaluated so that the maximum utility alternative at the root node is used as the act for the current turn. It was stated earlier in the chapter that the game tree should be constructed incrementally, taking the perspective of the acting agent at each node. Preconditions determine the applicability of acts, and so different belief states result in different game trees. The objective of the planner is to determine the best strategy over all of the possible belief states. For example, a plan with two preconditions might have four combinations of belief states (two times two), each with a different probability. The planner must find the value of the each strategy in each of the belief states

and take a weighted sum according to the probability of each state. This follows from the use of the maximum expected utility rule. Instead of using many such game trees, a representation has been chosen in which just one tree is used. Whenever a precondition occurs for an act, a "chance node" should be inserted just before the choice node in the game tree. Chance nodes always have two branches, one for each outcome of the precondition value. In the "true" outcome, the act appears at the following choice node. In the "false" outcome the act does not appear at the following choice node. This representation allows all of the different trees to be conflated into one, sharing the construction and evaluation effort where the different trees have common roots. The game tree can be equally well evaluated using a logical belief model, or the probabilistic model. The logical approach would be in keeping with Pollack's notion [53] of differing belief sets, while retaining the traditional logical model of belief. Instead of taking weighted sums over chance nodes, possible outcomes would be obtained, by checking whether the proposition at the chance node is believed (in which case only the "true" branch is possible) , disbelieved (in which case only the "false" branch is possible), or neither (in which case both branches are possible). This style of reasoning is particularly useful for the risk averse agent who would like to know the worst possible outcomes for the alternative available to him, given a set of nested beliefs. For example, if lives were at stake, a dialogue could be planned whose effect in revising the belief model would be to prune away the chance nodes at which both branches were previously possibilities, hopefully leaving a tree with no outcomes of plan failure. The probabilistic, rather than the possibilistic approach to game tree evaluation is taken for the design presented in this thesis.

To construct a choice node in the game tree, the plan rules should be

invoked as follows. The planner process should take as input the dialogue history and the belief model of the agent, and grow a game tree incrementally by planning forwards by one step at each game tree node, providing a strategy set which forms the node's branches. To do this, the planner should consult the plan rules of the level in the belief model of the acting agent. This is because each agent may have different capabilities. For example the system may believe that it is possible to make a pavlova with strawberries, but not expect the user to hold this plan rule as a capability. The dialogue history, concatenated with the sequence of acts that forms the path from the root of the game tree should parsed (using a standard parser for phrase structure grammars) according to the hierarchical plan rules. Then, a set of alternative strategies should be generated to continue the recognised plan by one step. This is done by planning forward in a focussed manner until a leaf act is obtained. To plan in a focussed manner the parse tree should be searched in left to right order for a node that does not have a full complement of children. Forward planning proceeds from the first such node. This process is the same as that of Carberry [10] (see section 2.5). If the parse tree is full (closed), a parent should be added to its root and forward planning should proceed along the path to this parent's second child. To do this the intention rules should be used (section 3.4.2). There may be many different ways both of parsing and planning forward. The alternative strategies should be gathered from each of these. To construct the subsequent choice node in the game tree, the next level of the belief model should be consulted, and using its plan rules, the act sequence reparsed from the perspective of the acting agent, and the next act produced. Since there may be different sets of plan rules, a different parse must be done at each level [53]. A game tree constructed from the perspective of the agent at level 1 would therefore alternate between

consulting level 1 and level 2 of the belief model in progressing through the game tree.

Chance nodes occur in two ways. First, if planning forward produces a branch at a choice node whose act has a precondition, a chance node ensures that the branch can only be added in the **true** branch of the chance node. Second, if a full plan tree is found, a parent must be added to the root. There may be many candidate parents, whose occurrence follows a probability distribution. Using the intention beliefs (section 3.4.2), a chance node should be generated to differentiate the different intention states that the acting agent may be in.

Implementing the planner module

The planner module (see figure 3.1) should be implemented as a recursive procedure that constructs the game tree a choice node at a time. For each choice node, the planner should be called. The planner should use a parser, taking plan rules from the given level of the belief model. An algorithm is required to search the given parse tree, in top down, left to right order, for the first node that does not have a full complement of children. If no such node exists, the tree is closed, and the intention rules should be used to attach a parent to the root of the tree. Parents should be repeatedly added until the search algorithm finds that the tree is open. The search algorithm, on finding the first open node, should apply the plan rules, matching with the node and its current children. One more child should be added to right of the current children. This new node should then be repeatedly decomposed at the leftmost child only, until a terminal node is obtained. This terminal node represents an alternative to be used as an edge at the choice node. The parsing and decomposition process should backtrack, so that all of the

alternative solutions are obtained. Each solution should then be used as an edge at the choice node. In adding nodes to the parse tree, the planner might encounter intention rules and preconditions. Each of these should be used to generate chance nodes. Along with the set of alternatives, information about these should be returned, and used to insert chance nodes in the game tree, ahead of the choice node. To ensure that chance nodes only occur once in the game tree, propositions and their values that have already been addressed by a chance node should be passed in the recursive call to the procedure that generates the game tree. This is because the evaluation function will always evaluate a second occurrence of a chance node as true in the true branch of the first occurrence, or false in the false branch of the first occurrence.

3.4.4 Example problem

To aid in describing the operation of the planner, a simple example is used. In this example, a goal act **ask-pair** is used by the first agent to ask a question. The first agent executes the first act in the decomposition, **ask**, and expects the second agent to parse the plan and produce the response **reply**. Reply can be further decomposed to either of tell-true or tell-false. **tell-true** has a precondition that the agent believes the proposition, whereas **tell-false** has the precondition that the agent believes the negation of the proposition. The STRIPS rules for this problem are as follows:

```
name:           ask-pair
parameter:      P
precondition:   {}
effects:        {}
decompostion:   { ask(P); reply(P) }
```

```
name:           reply
parameter:      P
precondition:   {}
effects:        {}
decomposition:  { tell-true(P) },
                { tell-false(P) }

name:           tell-true
parameter:      P
precondition:   bel(P)
effects:        {}
decomposition:  {}

name:           tell-false
parameter:      P
precondition:   bel(not(P))
effects:        {}
decomposition:  {}
```

To build the game tree (figure 3.2), the agent starts with the goal of **ask-pair**. It decomposes this in a focussed fashion, using the level 1 plan rules, finishing with a terminal node ask. Since this is the only way to decompose **ask-pair**, it forms the only strategy at the root node of the game tree (see figure 3.2). To generate the second level of the game tree, the planner starts with the node **ask**. Since this constitutes a full parse tree, it adds a parent, using the intention rules at level 2. The probability distribution of the candidate parents is obtained. There is in fact only one

candidate, **ask-pair**. This is connected to the root of the tree, forming a tree of two nodes, with **ask-pair** as the parent and **ask** as the child. Now using the beliefs at level 2, leftmost searching obtains ask-pair as the first focussed non-full node. **reply** is then added. Since this is a non-terminal act, it is decomposed further. There are two children that can be added - **tell-true** and **tell-false**. Since each has a precondition whose value is uncertain, the planner must insert a chance node before the choice node. The acting agent's belief set spawns a pair of belief sets, each corresponding with the outcome of its belief about the proposition at the chance node. As result, the planner produces the alternatives **tell-true** and **tell-false**. These spawned belief sets are propagated through the following subtrees since the acting agent's belief needs only be checked once. This prevents further chance nodes being inserted.

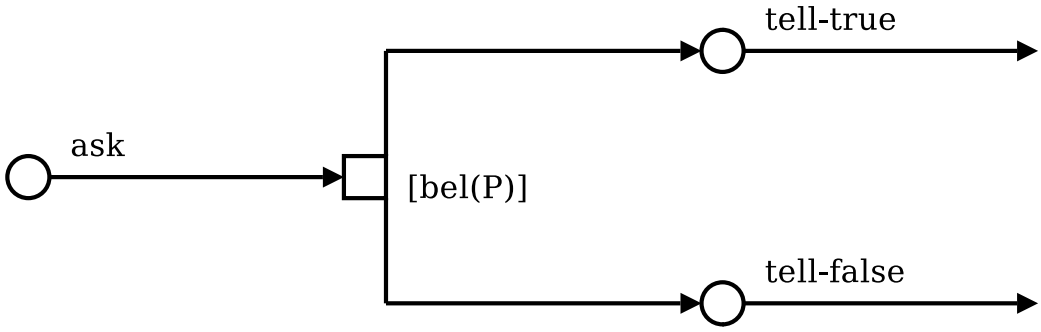

Figure 3.2: Game tree for a question and answer

3.4.5 Evaluation of the game tree

The second stage of deciding a strategy is the evaluation of the game tree in the context of the agent's belief state. Evaluation is based on the mini-max algorithm commonly used to evaluate game trees, since this algorithm returns the strategy of maximum expected utility [59]. This algorithm is

a recursive function which, for any given node, finds the minimax strategy, with respect to the other agent's utility function, for each of its children, and returns the one of maximum utility, using the agent's own utility function. While minimax was originally used in evaluating zero-sum games such as chess, where it takes the maximum of the minimum strategy selected by the opponent, it is used here to select the maximum strategy of the each agent from the that agent's perspective. Since agents have different beliefs and different utility functions an alternating perspective must be taken.

There are chance nodes in the game tree which must be considered. In principle, the game tree should be evaluated using minimax in every belief state outcome, and a weighted sum taken, according to the expected utility rule. In each possible belief state, the chance nodes can be removed from the tree according to the value of the state. This leaves an ordinary game tree, with no chance nodes, that can be evaluated using plain minimax. A weighted sum of utility is taken over these trees, according to the probability of each belief state. One way to do this is to check each possible belief state, finding its corresponding utility and probability. In practice, a better approach is for the evaluator to traverse the game tree, and each time a chance node is encountered, a pair of belief models is generated from the current one. In the first of these, the belief addressed by the chance node is set to true (a probability of one). In the second of these, it is set to false (a probability of zero). Each belief model is then propagated down one of the branches of the chance node. A weighted sum is taken at the chance node. Rather than generating all of the outcomes from the start, this mechanism generates the outcomes on demand as chance nodes are encountered.

To implement the evaluation module, a modified form of the standard minimax algorithm should be used [59]. This algorithm returns the best

"play" of the game tree, that is, a path representing the maximum utility strategies that each player is expected to choose in the course of the game. The standard algorithm is a recursive function that returns the best play from a game tree by obtaining the best play from the following choice node, and then evaluating each of the strategies available at the current choice node in the context of the following play. The difference between the standard minimax algorithm and the one that must be used here is that the agents disagree about the best play. Each has different beliefs, and so rather than compute one global best play, a best play must be obtained from the perspective of each agent, so that its best strategy can be obtained. To do this, the central minimax function should be first modified to return not a path as the best play, but a tree, with branching at nodes that represent the questionable beliefs of the two agents. This represents the best play path, but over all outcomes of the belief state. An "expected-utility" function should be used to evaluate such a play, by calculating the expected utility over the leaf nodes of the play tree, taken in the context of a belief model. To compute the best play, the minimax function should, for each of the alternatives at the root of the game tree, call minimax at level two of the belief model to obtain the best act of the second agent at the second level of the game tree. At the third level of the game tree, minimax should be called at level one of the belief model to obtain the best act for the first agent, and so on. Therefore minimax is called recursively, alternating between levels one and two of the belief models to obtain the best play following each of the alternatives at the root. Once best plays have been obtained for each of the alternatives at the root, the evaluation function is applied to choose between them, and the one chosen, along with the play that follows it, is returned as the minimax play.

The expected-utility function terminates at the leaves of the game tree,

where a function should be used to calculate the utility of the parsed dialogue history by adding up the contributions from each of the acts in the plan (see section 2.13).

Correctness

To indicate that the evaluation algorithm is correct, but without a rigorous proof, an outline of a correctness proof will be given. Consider that since the evaluation algorithm is recursive, a induction proof on the size of the game tree is suitable. For the base case, where there is one chance node, evaluation proceeds by evaluation of the complete plans that terminate the game tree. The one with the maximum expected utility is returned as the minimax choice. Since this is just a matter of comparing a set of alternatives, it is clear that it is correct. For the induction step, suppose that every game tree of n levels is evaluated correctly. If this is the case, then for a tree of $n + 1$ levels, all of the recursive calls to minimax that are made in choosing edges for the play tree must choose the maximum utility alternative. Since the play tree is correct, and since the evaluation of the play tree is only a matter of taking a weighted sum over belief state outcomes, the algorithm is a correct one.

Different game trees for different belief sets

In constructing the best play for an agent, the game tree from which the play is derived must be constructed using the plan rules drawn from its belief set. Therefore, at each call to minimax, a new game tree should be constructed that is based at the level of the planning agent. Unfortunately, this is computationally expensive since there are many calls to minimax in a typical game tree. In fact game tree construction is much more expensive

than evaluation, and it was found in implementation that recomputation of the game tree is not feasible on an ordinary computer. For the current design, the system only constructs one game tree. This is acceptable since in all of the examples given in this thesis, beliefs about plan rules are mutual - it is only beliefs about the domain state that differ.

Evaluation using a logical belief model

Evaluation of the game tree using a logical rather than a probabilistic belief model requires few changes to the design of the system. If the objective of the system is to choose the alternative with the best worst outcome (maximin), the weighted sum that is used with the probabilistic model can be replaced with a function that checks which of the three values of "believed", "disbelieved", and "possible" is taken at each chance node. Where the node proposition is disbelieved, the maximin value of the false branch is taken, where the proposition is believed, the maximin value of the true node is taken, and where the proposition is neither believed nor disbelieved, the minimum of the maximin values of both branches must be taken. The utility values obtained at the leaves of the game tree can either be continuous, or some ordinal representation of plan success and failure, for example 1 for success and 0 for failure. It is possible to transform a logical belief model to a probabilistic one so that it can be used with the current design for the belief model. One way is to use probability values of 1, 1/2, and 0 to represent the logical states of believed, neither, and disbelieved. These values can then be checked when the maximin evaluation function is applied at a chance node.

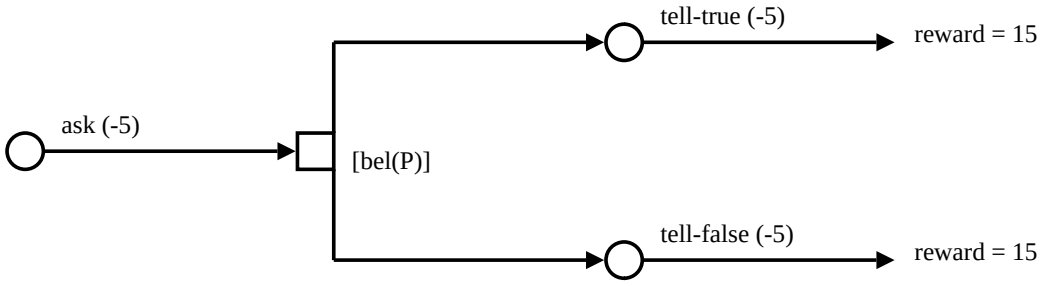

Figure 3.3: Evaluation example for a question and answer game tree

3.4.6 Evaluation example

To demonstrate the evaluation of a game tree, the example used in section 3.4.4 is used (see figure 3.3). Suppose that **ask-pair** has a positive utility of 15 units, and that each of **ask**, **tell-true** and **tell-false** has a negative utility of 5. To start, a belief state is initialised. Then, passing the ask branch, 5 is subtracted from the utility value. The chance node for the second agent is evaluated at level 2. Suppose this evaluates to 0.4. In the top branch, the proposition in the belief model for agent 2 is updated at level 2 to a value of 1. Next the minimax choice of agent 2 is found, in the context of the agent 2 belief model. This is trivially **tell-true**. A negative utility of 5 is subtracted here. Since a leaf is reached, the positive utility for goals achieved is computed. This amounts to 15. In the bottom branch, the proposition in the belief model for agent 2 is updated at level 2 to a value of 0, and the subtree is evaluated in the same way as the top branch. The top branch returns a total of 10, the lower branch returns 10 and their weighted sum is 10. Subtracting 5 for the initial ask act, the total utility for the tree is 5 units.

3.4.7 Belief revision

Plan recognition is used to infer the agent's intention given the dialogue history, so that possible continuations of a plan can be inferred. To differentiate the possible continuations, the beliefs of the agent must also be inferred, and the possible continuations of the plan evaluated in the context of these beliefs. The belief revision process is used to update the agent's beliefs from the evidence of executed acts. After each turn in the dialogue, the belief revision process is called. It is also used in the evaluation process, where as the game tree is traversed, the belief set used to evaluate a chance node must be updated to reflect the acts in the path that leads to it. Belief revision [23] involves the agent adding propositions to its belief set. The new set may be inconsistent, requiring that the agent search for some set that best satisfies the evidence given by the existing set and the new proposition together. In a logical model of belief, one procedure might be to drop as few propositions from the set as possible to obtain consistency. This would work quite well for a set such as $\{A \Rightarrow B, A\}$ with the revision $\neg B$. The inconsistent set would then be $\{A, A \Rightarrow B, \neg B\}$ and a minimal contraction of this set would obtain $\{A, \neg B\}$. In a probabilistic representation such as a belief network [50], statistical information about the co-occurrence of beliefs in the observed dialogues might be used to infer the causal relationship between the beliefs, and thus construct the causal structure of the belief network. As more data arrives, the causal structure might be recomputed. For a given current dialogue, the inferred network could then be evaluated using the node values obtained, which would provide updated values for the other beliefs in the network. There appears to be plenty of interest in the subject of inferring belief network structure, but it is also more than a trivial problem [32], and will not be tackled here. One way around the problem is to just assume

that there is no causal relationship between beliefs. By doing so, each belief can be updated independently and directly from the dialogue evidence. Such an assumption works quite well, especially for the examples that will be presented in this thesis, since for many of the preconditions found in the dialogue plan rules, there is no causal relationship. Therefore, this is the approach that is taken. Scenarios where this assumption does not hold as well are discussed in Section 6.3.3.

The basic revisions that should be made by the agent are those drawn from the preconditions and effects of the act. If an act was observed, then it must have been the case that its preconditions held, and it must be the case that its effects now hold. Therefore the preconditions must be added to the observers beliefs, followed by the addition of the effects. The order is important since effects can undo preconditions. The decomposition rule that the agent used to choose the act should also be added as a capability belief, but since there may be many suitable rules, it is not clear at the moment how to do this. Grammar induction techniques might be useful for this problem. The preconditions and effects of physical acts are added to the belief model at all levels, but for acts whose preconditions refer to the beliefs of the actor, only the level of the acting agent, to avoid having to resolve conflicts between the agents' beliefs. For example the physical act of handing a ticket has an unquestionable effect of giving possession of the ticket to the receiver, but the spoken act of claiming that a ticket is available has a precondition that refers to the beliefs of the speaker and so should be updated only at level 2 of the hearer's belief model. An assumption of full observability of all actions by both of the agents is made. This ensures that any revisions of beliefs that are made as a consequence of these actions are mutual. Therefore each proposition is adopted as a mutual alternating

belief. For example, if the user executes an act with a belief precondition, levels 2, 4, 6, 8 and so on are updated in the system's belief model. As a result of this lazy approach to belief revision, the system cannot cope with misconception dialogues [48] in which the agent must revise its own beliefs. For example, an agent may attempt a plan with a failed belief precondition. This should prompt the hearer to attempt to convince the speaker that the believed proposition does not hold. As a result, the speaker could try an alternative plan that is enabled by the revised base beliefs.

As an example of belief revision, consider the question and answer pair used in section 3.4.4. In executing this dialogue, the agent should respond to the user's answer by updating the belief model at levels 2, 4, 6 and so on.

As well as revising beliefs, the system should revise the intention rules. Each rule specifies a probability distribution of parent intentions for a plan tree, which should be updated from frequency counts in the dialogue data. This is not done by inference of preconditions or effects, but rather by counting the occurrence of acts in the plan tree. One difficulty with doing this is that there could be several candidate plans that explain an act sequence. If the probability of each plan can be obtained, an appropriate distribution of probability mass could be given to the parents in the intention rule. This probability might be found by checking for occurrences of each plan over all outcomes in the game tree. Due to the difficulty of implementing this, the revision of intention rules is left to future work. For the moment, the first parse is taken, and every parent in the parse tree is counted in updating the intention rules.

Dry-land algorithm

A less simple technique for belief revision that turns out to be necessary for some of the examples is one of searching for explanatory belief states when an agent executes an act. For example, an agent may choose not to say that it does not have a driver's license in a job interview, and although choosing not to tell has no direct preconditions, the interviewer should infer that the candidate probably does not have a license. There might be many candidate explanations. As another example, an agent may ask for some fruit. This may be explained by the agent's intention to make a fruit-salad. Equally well, the agent may intend to paint a still-life. There may be many supporting beliefs to these acts, such as all of the beliefs that satisfy the preconditions that support its subacts, which should be reinforced. Beliefs that support the agent's other alternatives may by the same principle be weakened. There might be many explanations, but the dry-land algorithm looks for the most simple one, searching the space of belief models for the most probable one given the previous belief model, in which the agent chooses the act. This is a case of Bayesian updating. Consider that a prior distribution over belief states is known, $P(B)$. The planner can determine whether an act will be chosen in each of these states, giving a value for $P(A|B)$. This value will be 1 or 0. To obtain the updated belief state, $P(B|A)$ is required. This is obtained using Bayes rule as follows:

$$P(B|A) = [P(A|B).P(B)/P(A)] \tag{3.1}$$

Since the most probable belief state is required, the P(A) factor cancels out when comparing $P(B|A)$ for pairs of belief states.

A straightforward way to calculate the dry-land belief state is to randomly sample the space of belief models, returning the most probable one in which

the act is chosen. "Most probable" is defined heuristically as the euclidean distance when the n probability values of the beliefs in the belief model are interpreted as a point in n-space. There will be illustrations of this algorithm later in the examples.

Bayesian updating has also been described by Gmytrasiewicz [26], who shows how beliefs are updated as choices are observed in Bayesian games.

Mutuality

The planner often works with deeply nested belief models. For a plan that is n steps deep, n levels are required. For example in a four step plan minimax would alternate between levels 1 and 2 in obtaining the best play at the root. The recursive call to minimax would then require alternation between levels 2 and 3 to obtain the second agent's choice. The next recursive call uses levels 3 and 4, and the final call is for a leaf choice node requiring only level 4. Although deep nesting seems necessary, for most of the examples, every second level is almost identical. This is a result of the mutuality of belief revision. Since both agents observe each other's acts, each revision occurs at every second level, which keeps every second level identical. For this reason, many of the starting states of the examples were specified compactly using two belief sets, which are then duplicated to produce a deeply nested model. However the planner must use a fully expanded belief model so that more difficult dialogues can be handled in which mutual beliefs are not adopted in belief revision, or where mutuality is not present in the starting state of the belief model. For example, a travel agent system might have a belief set at level 1 that is quite different from the belief set at level 3, since it is expected that the user has formed a stereotype model based on the general population of travel agents, rather than this particular one.

The assumption of mutuality may be more controversial once the scope of the planner goes beyond spoken dialogue, to encompass planning and recognition of acts in a physical world. Physical acts may not be immediately observed, with information about the order of execution lost, and the effects of acts overwritten by later acts. Agents may disregard turn-taking and act before observing all of the acts of the other agent. Some perceptual model is required. For example, a robot equipped with a sonar device would make a sequence of partial observations over time, since some effects will have been overwritten and some effects are hidden from immediate view. Where the dialogue planner generates plan hypotheses that are consistent with a history list of dialogue acts, the robot should generate hypotheses that are consistent with the sequence of effect observations. Since observation is part of the agent's cycle of sensing, planning and acting, expectations can be generated about observations as well as about actions. For example, a robot may plan to move into another room, and in generating the continuation of its plan, generate an expectation that another robot will fail to observe the effects of its actions.

Mutuality has a pleasant effect on the complexity of the evaluation function. If the belief model is identical at every second level, the evaluation function need not be applied from many different perspectives. Generally, a call to minimax using levels 1 and 2 will result in a recursive call to minimax that uses levels 2 and 3 and so on, but if mutuality is present, each of these calls would return the same best play. If mutuality of plan rules is present, only one game tree is required as well. As a result, this one game tree can be evaluated in one pass, rather than many. This may be one reason why dialogue participants rarely make the computational effort to consider more than a two level belief model [72].

3.4.8 Stereotypes

Stereotypes are used to represent a belief model not of a single agent, but of a particular class of similar agents. The stereotype is used in choosing a strategy that has the best expected outcome over the population that it represents, this being what can be expected for a particular member of the class. Where it is known that an agent is drawn from a stereotype class, but less is known about the agent's particular mental state, computation of an expectation using the stereotype is appropriate.

Stereotypes are most useful when a system is used only once by a particular user, yet the belief states of the user population form tight clusters from which classes can be derived, and when it is easy for the system to acquire many samples of data from the population. Stereotype acquisition is a matter of finding the belief model which maximises the probability of the belief states of the members of the stereotype class, given that belief model. This is a case of "maximum likelihood estimation", where a model is required that predicts the data in the distribution that was observed.

Only one stereotype is used here, representing all users, avoiding the problem of retrieving the correct stereotype for a given user. Stereotypes can also be used to represent agents whose beliefs vary from time to time, or are influenced by outside factors that cannot be modelled. The stereotype then is representative of the expected beliefs of the single agent, given a number of samples of that agent's dialogues.

In some cases where the system is using a stereotype model of a user, each user drawn from the stereotype interacts with the system only once. An example would be a once-off travel booking. In this instance the stereotype model is only one level deep, since the user has no experience of the system, and therefore does not form a dynamic stereotype over time. On the other

hand, if the same system interacts with the same user many times, the system must adapt the model at the second and subsequent levels, as each develops a dynamic stereotype of the other.

Acquisition of stereotype models is straightforward. A set of example dialogues is taken, and the belief revision mechanism is used to update the beliefs. For each belief in the belief state, a mean value over the example dialogues is obtained to compute the stereotype value. From this mean value, the correct distribution of belief states of the stereotype members can be recovered. For example, if 6 out of 10 agents believe grass is green, the distribution for 10 agents given a mean value of 0.6 is 6 out of 10. Happily, the mean value can be used directly by the evaluator, as the belief probability value, to find the expected utility over the stereotype members for a game tree.

A variation of using the mean value is to use a decaying average of evidence from recent dialogues. For example, each revision of the stereotype might use a weighted sum of ninety-five parts of the previous stereotype value, with five parts of the belief state produced by the current dialogue.

The system normally uses implicit acquisition, where it passively observes the dialogues and obtains information from belief revision. There is no use of explicit acquisition dialogue, such as direct questions, although such questions and the value of information they provide could be easily computed within the system, just by writing some plan rules for the explicit questions that occur when a new user is introduced, and evaluating the game tree that results. Explicit acquisition is discussed more in the "future work" chapter.

Stereotype error

One less than obvious but important factor in stereotype models is error. For example, a stereotype belief variable might take its value from ten dialogues, in five of which the value was true and in five of which the value was false. While the stereotype value would then be 0.5, the true value would be normally distributed around 0.5. This has the consequence that decisions that depend on the value of this variable may in some cases be in error. For example, there may be what will be termed a **decision surface** between two alternatives that occurs at the value of 0.4 for the variable. The decision surface is a surface in belief space across which the maximum utility alternative changes. Although the variable is estimated at 0.5, the actual value falls below 0.4 with a certain probability, resulting in a mixture rather than just one of the alternatives being taken. To deal with this problem, a sampling system has been implemented which randomly varies each value to simulate the error. Using 1000 samples, the system returns the alternative of maximum utility over the 1000 samples. 1000 samples is enough to ensure statistical significance of the decision, unless the alternatives are very close in utility. An example of use of the sampler will be given in the next chapter.

3.4.9 Complexity

Computational complexity is a subject that must be addressed, since game trees grow exponentially with the number of steps in the dialogue, so that their construction and evaluation is not effectively computable. Here are some candidate mechanisms to deal with this problem:

- **Focussing** In keeping with Carberry [10], the planner only admits plan hypotheses that are focussed (see Section 2.5 for an explanation

of focussing). This greatly reduces the branching of the game tree, yet the unfocused plans are usually so unlikely that there is little error in pruning them.

- **Alpha-beta pruning** Where the agents have opposing utility functions, alpha-beta pruning [59] can be used to prune away branches of the game tree by considering that the utility of branches so far evaluated in the game tree bounds the utility that the agent will accept for the unexplored branches. The agent at the previous level can use the bound to immediately discount an entire branch. However, agents with opposing utility functions rarely use spoken dialogue, so this sort of pruning would rarely be useful.

- **Abstraction** Since planning is performed by hierarchical decomposition, planning can be done at any level of abstraction. By choosing a higher level of abstraction, the planner can look further ahead, but use fewer steps in doing so.

- **Heuristic search** The agent can take estimates of utility from a higher level of abstraction, or from a breadth-first partially developed game tree at the current level of abstraction. With these estimates, the agent should advance the leading branches of the game tree, since the trailing branches are unlikely to be chosen. The branch-and-bound algorithm can also be used to prune paths without sacrificing optimality, by pruning those paths that can never lead to a plan that is better than the best one found so far. Heuristic search may be problematic however. This is because the agents have different beliefs and utility values, and so one agent cannot prune alternatives that are only poor from its perspective. One way to deal with this might be to prune only those

branches that are poor from the perspective of both agents. This ought to be effective in the cooperative setting where despite differences of belief, there is often agreement about the poorer alternatives.

- **Probability mass search** The utility function is computed using weighted sums. Therefore, error that is propagated through the game tree in computing the utility is also weighted. Therefore, the subtrees with greater weight deserve deeper exploration. This heuristic has been implemented by using a mass value that is propagated through the tree from the root. The mass is divided according to the weights at the chance nodes. Once the mass falls below a threshold, pruning begins. The effect of this pruning is to limit the chance nodes in the tree to a beam, with weighted random selection of branches at chance nodes once the threshold value has been reached.

- **Recombination** Plans can decompose in many different ways, since each action can have a number of decomposition rules. This leads to many paths in the game tree for alternative subdialogues that serve some root goal. However, once the subdialogue is finished with, the goal's parent is expanded in the same way, no matter which of the paths was taken in the subdialogue. Paths can therefore recombine at the end of subdialogues. This could be useful in generating the game tree, but would not reduce the complexity in evaluating it since beliefs are revised differently for each of the paths. Often though, such beliefs might only be used within one subdialogue. For example, in booking a flight, a subdialogue about whether the flier wants a window seat would refer to beliefs that are only relevant to dialogues about sitting in an aeroplane. In such cases, dialogue planning obtains the important

property of computational tractability, with the number of edges in the game tree being linear with the number of decomposition rules in the plan library. The proof of this property is by induction on the plan library. Assume first that plan libraries do not allow decompositions to be recursive in that they refer to their own parents. This is acceptable for most problems. Addition of a new decomposition rule to a plan library then results in the addition of at most one extra edge at the corresponding node in the game tree. It then requires one more edge to recombine this node with its sibling nodes to close the subdialogue. Therefore the size of the game tree is linear with the size of the plan library. Figure 3.4 illustrates a plan library, and its corresponding recombined game tree. Implementation of the recombination planner is left to future work.

- **Alternating mutual belief** In many dialogues, it is mutually believed that the dialogue is observed by both agents, and that appropriate belief revision inferences are made. Therefore whenever a belief is revised, it can be assumed to be revised at every second level of the belief model, producing an alternating mutual belief. For example, if agent 1 declares that the sky is blue, it should expect that agent 2 has observed this declaration and so a revision is appropriate at level 3. This argument can be extended to level 5, 7 and so on. It is often the case that the agents begin a dialogue with mutually believed stereotypes. For instance one may be a 'customer' and the other may be a 'travel agent'. As a result, at the beginning and throughout the dialogue, every second level is identical. This fact makes construction and evaluation of the game tree more efficient, since only one tree is required and it can be evaluated in one pass. Previously, it had to be

evaluated many times from the different perspectives of the different levels of the belief model.

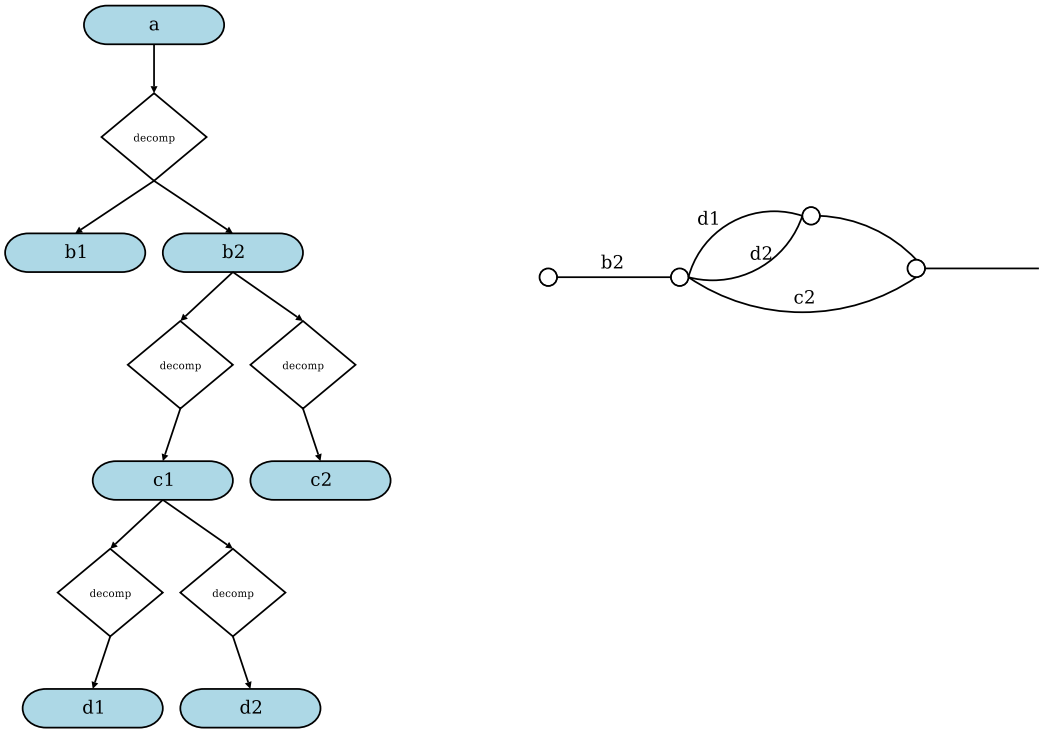

Figure 3.4: Recombination example (a) Plan library (b) Corresponding game tree with recombination

Only one of the complexity strategies has been implemented, the probability mass search, but there are no results describing its effectiveness. It is promising in that the implementation is straightforward, and that it allows enough samples of the possible dialogue outcomes to be taken to make an effective decision while keeping the complexity of the chance nodes linear with respect to the dialogue length. Unfortunately there is no general way to prune the choice nodes. Although it seems more difficult to implement, and requires certain properties of the plan rules, recombination might be the next most promising candidate since it changes the complexity of the game

tree, both choice nodes and chance nodes, without threatening to degrade the performance of the system.

It might be argued that a system that performs computations with game trees would be slow to respond in real time, and therefore irritating to the user. However, for unwieldy game trees used in routine dialogues, strategies can be precomputed by the planner, and updated at regular intervals as the stereotype model develops, perhaps once a day. The strategies would be represented by a stored game tree pruned to the system's best play at each system choice node. Such a tree could be consulted almost instantaneously.

3.4.10 Multilogues

A multilogue is a discourse with contributions from more than two agents, in contrast to a dialogue, in which there are always two agents. Multilogues are not considered in this thesis, nor is the planner capable of planning them, but a sketch of multilogue planning is given here. To plan a multilogue, the game tree is much the same as that used in the two-party case, with each agent adding a level using its plan rules. Consider a three-party multilogue. Assuming that the agents take orderly turns, agent 1 would plan level 1, agent 2 level 2, agent 3 level 3, agent 1 level 4 and so on. Evaluation of the game tree is somewhat different however. Since there are three agents, each agent maintains a model of not one other, but two other agents. This means that the nested belief model is a binary tree rather than a list. To compute the best play for the game tree in a two party dialogue, the evaluating agent would alternate between calling minimax at levels 1 and 2 as a path is followed in the game tree. In the three-party case, the agent alternates between its base beliefs, then the first of the nested models, then the second of the nested models, and then back to its base beliefs. In multilogues, turn taking

may not pass in a circle, since speakers can address particular hearers. For the example of a question addressed to one hearer, and for the purpose of dry-land belief revision, the hearer who has been addressed should come to believe that the speaker wants its answer, whereas the hearers who have not been addressed should come to believe that the speaker wants the addressed hearer's answer, rather than theirs. This would not prevent an unaddressed hearer, who believes that the addressed hearer has the wrong answer, from attempting to take the floor. An example in a two-party case of agents holding the floor by considering the utility value of their own and the expected contribution of the other agent in a dialogue will be given later, in chapter 5.

3.4.11 Conclusion

The description of the planner given in this chapter has been of an informal nature, and particularly, there has been no formal design specification from which to derive an implementation for the planner. This is mitigated by the choice of a high-level programming language, Prolog, for the implementation of the planner. For example, the Prolog program specifies the minimax and evaluation functions using recursive rules that are as useful for a design specification as they are as an implementation. There is a guide to the implementation given in appendix A.

Some functional requirements were set out at the beginning of this chapter. One was that the planner should be based on current theories of dialogue planning, and in basing the plan recognition and focussed forward planning of the system the theories of Carberry [10] and Pollack [53], this has been achieved. Another requirement was that the system should be easy to use. This ease of use stems from the ease with which the starting mental state of

the system can be given by the designer. He needs only to give the following: a utility function to evaluate plan trees, and initial beliefs about plan rules. Once this is done, the choice of strategy, management of the dialogue, and initialisation and maintenance of the user model through belief revision is automatic. Perhaps the most important functional requirement was that the system offer some utility gain over a system that cannot use a user model at all, and over a system that uses the traditional logical, rather than a probabilistic belief model. There must be good reason to depart from the well accepted logical model, and so this final requirement is the main topic of the next chapter, in which two examples are used to establish that such a utility gain can be obtained. These examples will also illustrate the operation of the algorithms that have been described in this chapter.

Chapter 4

Evaluation

4.1 Introduction

In this chapter, the achievement of the objectives set out in Chapter 1 is shown, namely that the implemented planner is easy to use for the dialogue system designer and that it is more efficient than other planners. The achievement of the first condition will become evident as two examples are presented for which simple sets of plan rules can be used.

The second condition will be partially answered in this chapter, by showing how much more efficient the planner is than those that either use a logical, rather than a probabilistic belief model, and those that use no user model whatsoever. However, in designing a planner that adapts itself to a model of the user, reinforcement learning must be considered as a competitor (see Section 2.9.1). Reinforcement learning systems use data about dialogues with users to reinforce dialogue strategies, and so they can be said to adapt to the user. Due to the difficulty of implementing both the planner and a reinforcement learning system, this has been deferred to future work, but is an important comparison to make. It is expected that in situations where

there is little training material, the planner would perform better than a reinforcement learning system, especially where the plans are novel. Such plans require the intelligent application of planning knowledge, and reinforcement learning fails in this respect, relying instead on the brute force of training data. Novel dialogue planning might occur in, for example, a meta-level dialogue about a complex domain-level plan. There will be further discussion of this question in the future work chapter, Chapter 6.

In summary, the chapter will describe a simulation method for evaluating the system, and compare this method with the preferable, but impractical one of direct evaluation with users. The evaluation makes use of two examples that are typical of the kind of problem where strategies create a decision surface, that is, the choice between them depends on the belief state of the agent. A user model is of benefit only for problems that have a decision surface.

4.2 Implementation

One approach to implementation would be to write a design specification based on the discussion in the last chapter and write a corresponding program. Instead of this, the planner has been implemented as a Prolog program, which serves as a directly executable specification. Each of the modules described in the last chapter has a direct representation in Prolog. A description of the program is given in the appendix.

4.3 Evaluation method

The approach taken to evaluate the planner is one of dialogue simulation, rather than trials with real users. Each approach has different advantages. User trials are typically much better than simulation since they provide concrete evidence of the performance of the system in its intended setting. Unfortunately, they are relatively laborious to produce, taking hours of time to produce perhaps a dozen dialogues. To produce statistically significant results hundreds of trials may be necessary. Using a stereotype belief model, the planner is supposed to adapt to random distributions of belief states. The problem then arises of obtaining many different subjects so that a suitable distribution of stereotype values is obtained. Ideally, several runs of training and testing should be used, with each run involving many dialogues and each producing a different stereotype value - different settings would have different characteristics, with different distributions of belief states among the subjects. With human trials it would be difficult enough to investigate just one dialogue problem with one distribution of users, producing just one stereotype value. The advantage of simulation is that it allows dialogues to be generated for many different stereotype states, allowing coverage of all sorts of user populations and different problems. Many of the experiments conducted for this thesis examined around one hundred different stereotype belief states to obtain sufficiently detailed results. Such detail would have been impossible with human trials. The main disadvantage of simulation is that the simulation model of the user could be incorrect, leading to incorrect results. On the other hand, this is not a problem in human trials, once enough data is available to ensure statistical significance. Simulation treats the human user as an ideal decision-maker. It is well known that the performance of most humans is less than ideal [20, 68, 75], and so in using simulations, there

is a missed opportunity to discover and accommodate the human decision-making process. There is no assurance that the characteristics discovered about the planning problems transfer to the human setting, nor that the efficiency claims about the planner transfer. On the other hand, experiments have been conducted to investigate human communicative choices in game-theoretic problems, for example, exchange of information in a war exercise [25]. The human choices were found to be close to the ideals chosen by the computer. The simulation approach also has some popularity in developing dialogue systems, being used by [30] and [36] in studies of initiative in dialogue, and by [64] in evaluating different reinforcement learning systems on a simulated user. The simulation method is quite easy to use, since it uses the planner to compute the strategies of both the user and the system. This being so, the game tree generated by the planner can be used to represent the simulation outcomes, because the planner already simulates the user to generate the game tree. The game tree can then be used to explore the dialogue outcomes obtained from different belief states.

The main differences between a simulated dialogue and one with a human participant relate to the bounded reasoning resources of the user. Some of the game trees used in the examples are quite deep, and the dominance of different strategies is closely related to the probability values in the belief model. A human decision maker could not be expected to compute the game tree with such depth or precision. A second problem is that human users tend to fit themselves to an interface by reinforcing routine responses over time. If the system responds to a changing belief model by changing its strategy, there may be a period thereafter where the user performs badly as he tries to learn a new policy to fit the new strategy. For example, most users would be upset if the menu options or dialogue sequence available to

them on a graphical user interface were to change from the that which they have become used to.

The planner is required to show an advantage over systems that have no user model. It is clear that a system with no user model will not change its strategy as the user or population of users changes. Therefore, the only way to design such a system is to make a good choice of strategy at design-time, and fix it for the lifetime of the system. For example, if there is a choice between two strategies, the planner is compared with the best of these as the system's fixed strategy. It is difficult though to quantify the gain, since if the belief model does not change much during the lifetime of the system, then the system does not need to change its strategy and so there is no gain over one of the fixed-strategy planners. It is only when the belief model drifts across the decision surface that a gain is obtained. Without user data the path that the belief model takes cannot be known, and so the gain obtained over the system's lifetime is not clear.

One of the stated objectives of the system is to provide an efficiency gain over current dialogue systems using a finite state or frame-based dialogue manager. It is not hard to translate the plan rules of such planners into equivalent hierarchical plan rules of the sort used by the fixed-strategy planner. Each parent in a hierarchical rule can be used to represent the state, the first child the output of the system, and the second child the next state. Conversely, the fixed-strategy form of the examples given in this chapter have a finite state equivalent, since each parent can be written as a state, with alternative decompositions represented by alternative edges and next-states. The preconditions must of course be ignored in performing this transformation. Due to this equivalence, it can be said that a comparison with finite-state systems is being made.

4.4 Amenable problems

The kind of dialogue problems to which the planner can be applied are those that first of all have an element of risk generated by preconditions whose satisfaction is uncertain to the deciding agent. For example, a speaker might decide whether to ask a mechanic to fix his car or whether to fix it himself. He might be sure of the outcome should he perform the repair himself, but for the mechanic to make the repair, there would be some risk due to preconditions like having the right tools, or knowing how to fix the car. As well as having risk, there must be a decision surface between the alternatives of the decision. For example, if the worst outcome for allowing the mechanic to make the repair is a utility of 10, the best outcome a utility of 20, but the outcome for the speaker repairing the car himself is 25, the planner is of no benefit because 25 does not lie between 10 and 20 and no decision surface is formed. On the other hand, if the utility lay between 10 and 20, say 15, the planner would be useful, since without a nested belief model, the best alternative is unclear. In this case the planner would at best obtain a utility gain of 5 units over a planner with no user model, but the utility gain would never be negative. While a logical model would be more useful for this problem than no user model at all, it would be of little use where the estimate of the mechanic having the right tools is based on many samples of past experience rather than absolute knowledge of this particular mechanic.

In Section 4.5 and Section 4.6, two problems will be discussed, which have risky alternatives between which there is a decision surface. These problems particularly focus on taking initiative in dialogues, where an agent must decide whether to open a dialogue, and how much to invest in communicating its intention or contributing to the plan should it do so. Example 1 deals with initiative taken in grounding the speaker's intention, where the speaker can

choose between strategies of different levels of risk of misunderstanding. The less risky strategies have a higher cost. The hearer can also take initiative in grounding the intention by planning a clarification subdialogue, which reduces risk. Example 2 deals with initiative in introducing a goal to a dialogue. In this problem actions have preconditions which can fail, which determines whether the speaker should introduce it, or take the risk free alternative of allowing the hearer to introduce the goal if his preconditions are satisfied. What is common to both of these problems is that the agent can choose between a less risky alternative, whose utility is more or less constant with respect to a belief, and one that is more risky, having a utility that varies with the belief. Because of this variation, a decision surface is formed.

It may be argued that there are few problems that exhibit a decision surface, and therefore not much need for the planner. However, one need only look to work on reinforcement learning (Section 2.9.1) to know that inference of a model of the user from dialogue data makes a difference to the strategy taken in a dialogue. Although reinforcement learning is not always used to adapt to one or a group of users, but rather to the characteristics of the dialogue itself (such as the execution cost of different strategies), some of the justifications for that work are inherited by the planner.

That decision surfaces do occur regularly in everyday dialogue is evidenced by the human tendency to find out about and commit to memory the state of the world and the mental state of the actors found in it. If few decisions depended on knowing whether preconditions were satisfied, people could go through life spending very little effort on learning about the world. For example, a mechanic would find a decision surface in choosing to ask his colleague to mend an exhaust and doing it himself, with beliefs about available tools and parts, his colleague's skills and availability, and the likely cause

of the problem all potentially causing a decision surface for the alternatives.

The planner, whose belief model is probabilistic, should be compared with planners that use a logical belief model, as well as planners that use no belief model at all. It will be shown that at some intermediate point of probability for a belief, between 0 and 1, the strategy of the agent should change, and since a logical belief model does not record probabilities, it is impossible to use it to decide a strategy based on maximum expected utility. Moreover, the difference in utility between strategies on one side of the decision surface can strongly favour one strategy, whereas on the other side of the surface, strongly favour the other strategy. Therefore, the agent needs to maintain its state on a continuum of belief states rather than at the extreme points of this continuum.

4.5 Example 1: Risking misinterpretation

This Section will present the first example dialogue planning problem. Often in dialogues, communicative choices come with different levels of risk. In a paper by Carletta [12], it was explained that by making a low risk communicative choice, the speaker must make more effort, but with the benefit that the hearer is less likely to make a misinterpretation that would lead to failure of the plan. On the other hand, the speaker can choose a risky alternative, which requires less effort, but in turn has the risk of costly recovery. Carletta explains a number of such choices such as specificity of referring expressions and indication of focus shifts. There are also corresponding recovery strategies, such as replanning, clarification subdialogue, and repetition of the plan. Sometimes both agents have the opportunity to initiate a recovery strategy. There are many other risky devices that can be used in dialogue. At the se-

mantic level, examples are anaphora and ellipsis, syntactic ambiguity, or use of words and phrases with ambiguous senses. At the pragmatic level, there can be incomplete evidence given by the speaker to unambiguously identify his dialogue plan, or insufficient amount of clarification used by the hearer in eliciting it. Such choices are commonly made when agents try to establish mutual beliefs (known as 'grounding') [15]. Hearers must continually decide whether to open clarifications when it is not certain that the speaker's belief has been established as mutual. As an example of establishing mutual belief, consider giving someone a phone number over a noisy telephone line. The noise creates risk in the outcomes of the agents' communicative choices. It may take several exchanges of confirmations before the hearer is happy that it has the right number, that the other agent believes it has the right number and so on.

In this section, a quantitative approach is taken in planning to take risks in dialogue, complementing Carletta's paper which gives details of particular types of risk-taking, but does not explain how risk-taking might relate to dialogue planning. Only one example is used here, that of the planning of a referring expression to describe an object **car-spanner**, but other types of risk taking would be planned in the same way. There is a choice between a low risk referring expression, which cannot fail to correctly identify the object, and a high-risk referring expression, which requires less effort, but risks misinterpretation by the hearer as **bike-spanner** instead of **car-spanner** The low-risk expression is:

```
"I need the two-inch hexagonal double-jointed wrench so I
can fix the wheel nut"
```

The high-risk expression is:

"I need the spanner"

In choosing the high-risk expression, the speaker is declining the initiative in the objective of grounding his intention. Instead, the hearer has a choice of using a clarification dialogue with which it takes the grounding initiative instead. The risk varies according to the state of the belief model. If the belief model causes the agent's candidate intentions to be recognised as equally likely, the risk is high. However, if one of those intentions were to be more likely than the other, the risk is low and so the risky alternative would be a better choice.

The effect of stereotype acquisition will be demonstrated in this example, showing a performance improvement with each dialogue run, as more and more data accumulates. To this end, the sampler as described in Section 3.4.8 is used.

4.5.1 Plan library

The plan library for the problem is described diagrammatically in figure 4.1. The notation used in this diagram is intended to represent the decomposition rules that form the capabilities of the agent. The "decomp" diamond shape represents the relation "Act A may be decomposed to acts A1....An". The colour coding is intended to indicate the agent who executes each of the acts. Blue represents agent 1 and pink represents agent 2. To apply these rules, the agent might start with fix-car for example, and follow a chain of decomposition to find an act to execute. In this example, ask-car-spanner would be chosen. The agent then has a choice between three decompositions resulting in one of the leaf acts ask-car-unambiguous and ask-ambiguous.

The belief model used for this problem is 5 levels deep, corresponding with planning to a depth of 5 steps. Each level is initialised with the same

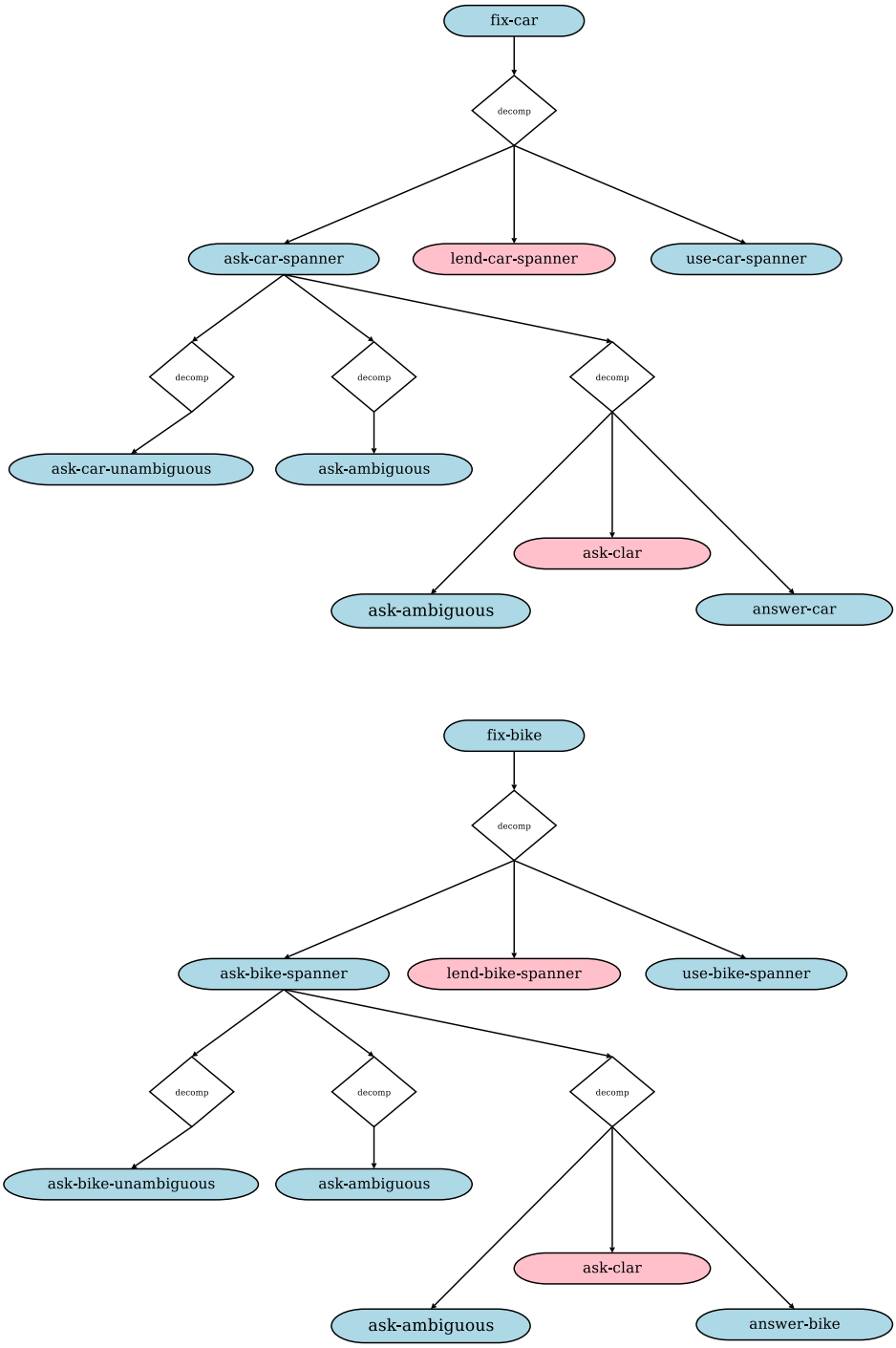

Figure 4.1: Plan library for the risk problem

belief set, since their is no dispute between the agents over the plan rules for the problem. The code used by the planner now follows. Notice that all of the decomposition rules from figure 4.1 are included, as well as some intention rules. Recall that these rules are used for inferring parents in full subtrees during plan recognition. Of particular interest is ask-ambiguous, which being risky, has two possible parents. Notice the correspondence between this specification language and the definition of the agent state given in Section 3.4.2

```
[
  p(decomp(fix-car,
    [ask-car-spanner,lend-car-spanner,use-car-spanner]),1),

  p(decomp(ask-car-spanner,
    [ask-ambiguous,clarify-car]),1),

  p(decomp(ask-car-spanner,
    [ask-car-unambiguous]),1),

  p(decomp(ask-car-spanner,
    [ask-ambiguous]),1),

  p(decomp(clarify-car,
    [ask-clar,answer-car]),1),

  p(decomp(fix-bike,
    [ask-bike-spanner,lend-bike-spanner,use-bike-spanner]),1),
```

```
p(decomp(ask-bike-spanner,
  [ask-ambiguous,clarify-bike]),1),

p(decomp(ask-bike-spanner,
  [ask-bike-unambiguous]),1),

p(decomp(ask-bike-spanner,
  [ask-ambiguous]),1),

p(decomp(clarify-bike,
  [ask-clar,answer-bike]),1),


p(intend(fix-car,
  [ask-car-spanner]),1),

p(intend(ask-car-spanner,
  [ask-ambiguous]),0.5),

p(intend(ask-car-spanner,
  [ask-car-unambiguous]),1),


p(intend(fix-bike,
  [ask-bike-spanner]),1),

p(intend(ask-bike-spanner,
```

```
    [ask-ambiguous]),0.5),

  p(intend(ask-bike-spanner,
    [ask-bike-unambiguous]),1)
]
```

The utility function for this problem is given in the code fragment below. For simplicity, utility functions have been implemented as mutual, and so are specified outside the nested belief model. The values chosen are estimated, rather than directly based on empirical data. A reward of 100 is given if the correct spanner is passed. This is reduced to 80 if the wrong spanner is passed since it would cost 20 units to replan and execute a dialogue in which the agent asks again. If the first agent replans the dialogue, then the hearing agent must accommodate the second attempt by discounting the intention state in which the speaker intended the spanner that he was given on the first attempt. This revision uses the dry-land algorithm. As a result of this revision, the speaker need only ask for the spanner again, and the hearer ought to realise that he has passed the wrong spanner in the first instance. Since the dialogue is guaranteed to succeed at the second attempt, there is a constant 10 for asking and 10 for giving. Rather than build a game tree deep enough for both attempts, the reward of 80 rather than 100 was placed at the leaf corresponding with the failure of the dialogue on the first attempt.

While the utility values for the acts were estimates, informal checks were made to ensure that reasonable variations of these values did not result in more than proportionate changes to the utility values of the alternatives available to the agent. It was found that utility functions retained their general shape and decision surfaces remained approximately in place. These checks did not systematically vary the utility values, nor were the results

recorded, but they provide some evidence that the results are reasonable. Checks were performed for each of the demonstrations in this thesis.

Notice that the utility function is compositional, in keeping with the discussion in Section 2.13, where it was claimed that the value of a dialogue could be worked out using a sum of the reward for task completion and the sum of the costs of the dialogue acts.

```
utility(ask-ambiguous,-5).
utility(ask-car-unambiguous,-10).
utility(ask-bike-unambiguous,-10).
utility(lend-car-spanner,-10).
utility(lend-bike-spanner,-10).
utility(ask-clar,-3).
utility(answer-bike,-1).
utility(answer-car,-1).

utility(use-car-spanner,0).
utility(use-bike-spanner,0).

reward(Plan,100) :- plan_contains(Plan,fix-car),
                    plan_contains(Plan,use-car-spanner), !.

reward(Plan,100) :- plan_contains(Plan,fix-bike),
                    plan_contains(Plan,use-bike-spanner), !.

reward(Plan,80).
```

4.5.2 Demonstrations

In the series of demonstrations that follow, the game tree is taken one piece at a time, showing how the strategy of the system varies with the belief model, and finally comparing the system with each of the fixed strategies for the problem. Two parameters are important. First, the intention rule at level 3 is varied to represent the level of risk in asking for the car-spanner. This is because the second agent is expected to use the rules at this level in recognising the first agent's plan, and in particular, to decide which spanner to lend to him. This variable is called p. Second, since the sampler is employed in the demonstrations (see Section 3.4.8), the stereotype error with which the agent has estimated p is also varied. This is represented by n, the number of dialogues on which the belief model has been trained. A range of values, 2, 8, 32, and 128, was used for n. In the demonstrations, the first agent enters the dialogue intending to use the **car-spanner**.

The game tree

Figure 4.2 describes the game tree that is generated by the planner from the plan library. At the root node, the first agent takes grounding initiative in choosing a non-risky strategy, or declines the initiative by taking a risky strategy. The non-risky strategy leads to the always successful outcome described by the lower branch. The risky strategy leads to a choice by the second agent about whether to clarify the first agent's intention. If it does clarify, there are two possible responses from the first agent, leading to the conclusion of the dialogue. If it does not clarify, it must choose between responses, each of which risks failure.

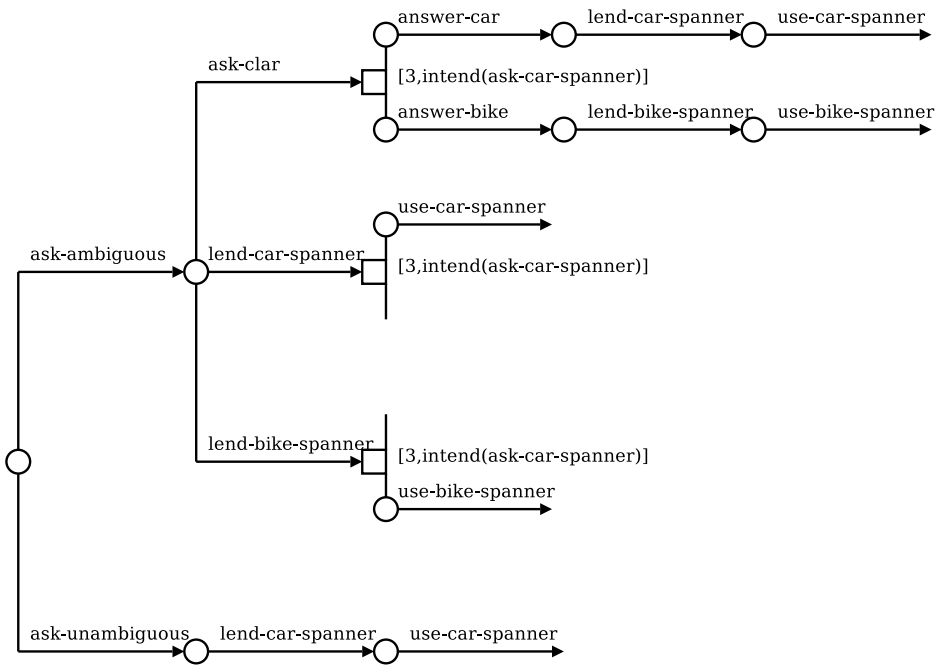

Figure 4.2: Game tree for example 1

Belief revision

During the dialogue the belief state of the first agent changes due to belief revision. To illustrate the development of the belief state consider an example initial state. Notice that there are five levels since the game tree has a depth of 5.

```
level 1:
{
p(intend(ask-car-spanner),1)
}

level 2:
{
}
```

```
level 3:
{
p(intend(ask-car-spanner),0.7)
p(intend(ask-bike-spanner),0.3)
}

level 4:
{
}

level 5:
{
p(intend(ask-car-spanner),0.7)
p(intend(ask-bike-spanner),0.3)
}
```

Now consider the alternatives available in the game tree (figure 4.2). ask-ambiguous has no effect on the mental state since there are two alternative parent intentions to this act. On the other hand, there is only one parent available for ask-unambiguous and so agent 2 may revise its belief about agent 1's intention. This is an alternating mutual revision and so level 3 and level 5 are revised. The same effect is obtained by using the answer-car and answer-bike acts since these two have a single possible parent intention. The following is the resulting belief model:

```
level 1:
{
```

```
p(intend(ask-car-spanner),1)
}

level 2:
{
}

level 3:
{
p(intend(ask-car-spanner),1)
}

level 4:
{
}

level 5:
{
p(intend(ask-car-spanner),1)
}
```

Demonstration 1: The second agent's response

In this demonstration, the second agent's response to the risky strategy is analysed, but for now, the clarification alternative is ignored. The piece of the game tree that is examined is described in figure 4.3. For this game tree, the evaluator is expected to produce for the upper branch:

$$\begin{aligned} & -10 + p.(0 + 100) + (1 - p).80 \\ =& 20p + 70 \end{aligned} \tag{4.1}$$

For the lower branch, the evaluator is expected to produce:

$$\begin{aligned} & -10 + p.80 + (1 - p).100 \\ =& -20p + 90 \end{aligned} \tag{4.2}$$

These formulas can be verified by looking at the utility function in Section 4.5.1. To find a decision surface between the two alternatives, the points in the belief space at which the alternatives have equal utility are found. There is one solution in this case.

$$20p + 70 = -20p + 90 \tag{4.3}$$

$$\Rightarrow 40p = 20 \tag{4.4}$$

$$\Rightarrow p = 0.5 \tag{4.5}$$

Therefore, it can be expected that the second agent will lend the car spanner when $p > 0.5$, and will lend the bike spanner when $p < 0.5$. Figure 4.4 is a plot of the planner's output, which conforms to expectations. This is the plot for $n = 8$ which is identical to the plots for the other training levels, 2, 32 and 128.

Demonstration 2: Planning the first move

In this demonstration the decision of the initiating agent on the first move is examined. Once again, the clarification subtree has been omitted for the moment. The game tree is described in figure 4.5.

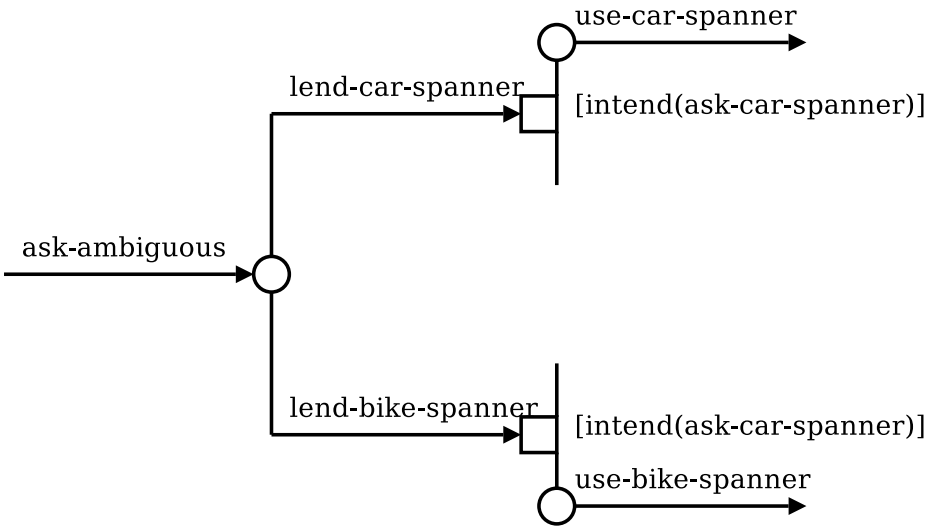

Figure 4.3: Game tree for the second agent's response

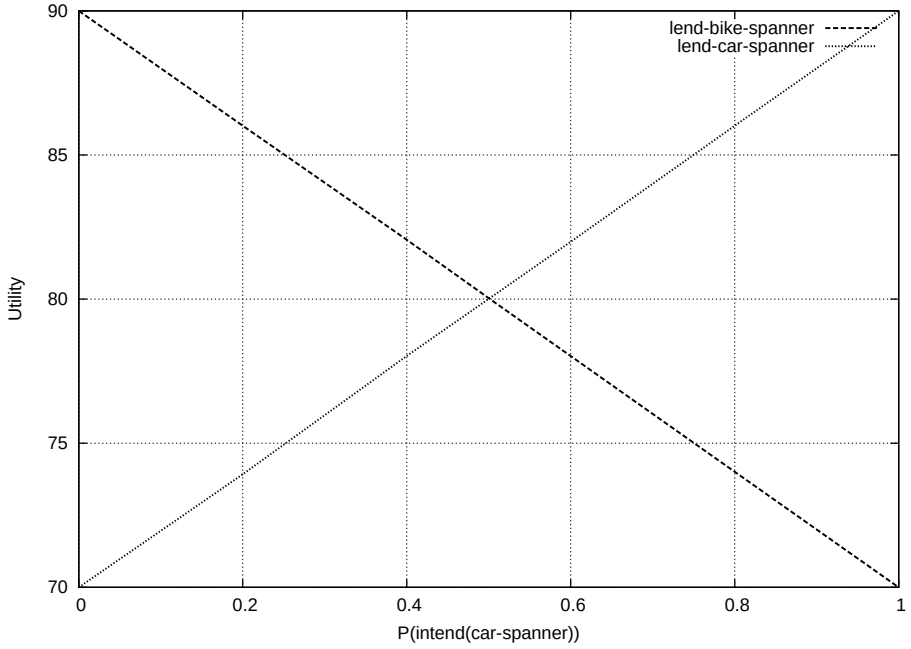

Figure 4.4: Utility of strategies against P(intend(car-spanner))

There are two alternatives available - ask- ambiguous and ask- unambiguous. For evaluating the tree (see Section 2.13), the first agent must evaluate

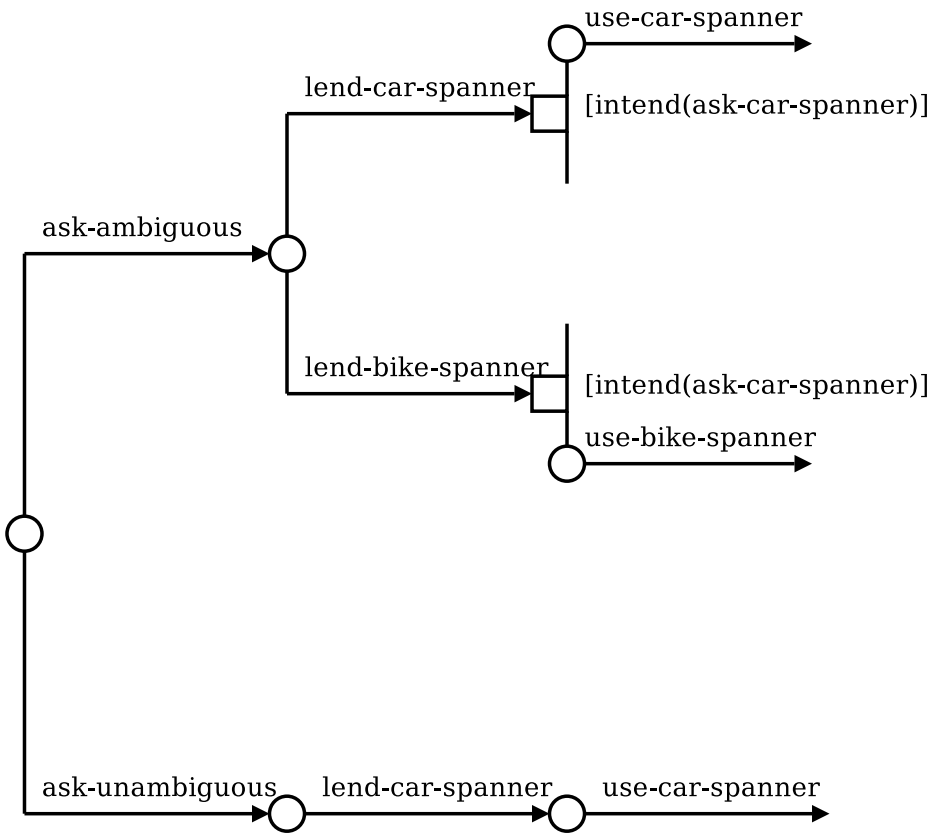

Figure 4.5: Game tree for the first agents choice without clarification

the second agent's decision at level 2, and then evaluate the chosen branch of the tree using level 1. In particular, from the first agent's perspective, the chance node at the third move is evaluated at level 1, whereas when the first agent takes the second agent's perspective, it is evaluated at level 3.

ask-unambiguous has a straightforward utility value:

$$\begin{aligned} & u(au) + u(lcs) + u(ucs) + reward \\ =& -10 - 10 + 0 + 100 \\ =& 80 \end{aligned} \tag{4.6}$$

On the other hand, ask-ambiguous is more complicated, because the re-

sponding agent's decision changes with the decision surface at $p = 0.5$. There are thus two cases:

if $p < 0.5$

$$\begin{aligned} & u(aa) + u(lbs) + reward \\ =& -5 - 10 + 80 \\ =& 65 \end{aligned} \tag{4.7}$$

if $p > 0.5$

$$\begin{aligned} & u(aa) + u(lcs) + reward \\ =& -5 - 10 - 100 \\ =& 85 \end{aligned} \tag{4.8}$$

Therefore, when $p < 0.5$ the non-risky alternative (80) is better than the risky alternative (65). On the other hand, if $p > 0.5$, the risky alternative (85) is better than the non-risky alternative (80) . It might then be expected that the utility curve is a step function. It is not quite, because estimation error comes in to play. Near the decision surface, estimation error causes the responding agent's decision to spill across the surface. For instance, if p is estimated at 0.55, it is quite possible that the actual value is 0.45, resulting in a different choice and a different utility value. Using four degrees of estimation error, corresponding with 2, 8, 32, and 128 samples, the sampling version of the planner obtained the curves in figure 4.6

The spill is clearly evident in the plot, and has a significant effect on the decision of the agent since the step is so abrupt. For a naive agent, who has only experienced two samples, the decision surface is not at 0.5 but past 0.7. As the number of samples increases, the curve approaches the

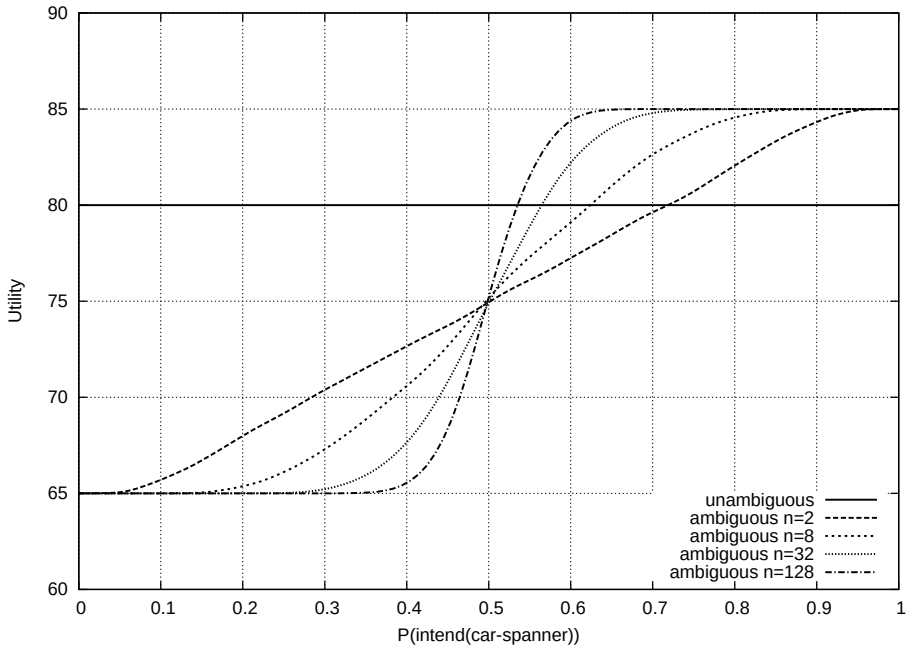

Figure 4.6: Utility of strategies against P(intend(car-spanner))

step function form that was expected of an agent with no estimation error. This demonstration confirms that error estimation, as well as probabilistic reasoning, plays an important role in dialogue decisions. Error estimation also has some effect on the utility curve. Notice that the area under the curve is somewhat greater as n becomes larger. This increased area provides extra impetus for the agent to engage in a dialogue, since what is learned during it improves the agent's performance in later dialogues. The most straightforward way to compute the impetus would be to compute deeper game trees, so that the current dialogue, and the later ones, are accounted for within one game tree. To do so may require algorithms that limit the combinatorial explosion of the game tree, outlined in Section 3.4.9. For example, many instances of the spanner dialogue could be joined to produce a long chain. This approach could also be used to evaluate explicit user model

acquisition questions, which can be used when the user is introduced to the system, but whose value is obtained slowly over the course of future dialogues. In general, it makes sense for the planner to plan not just the immediate dialogue, but to construct very deep game trees, so that the value of a belief that is learned about now can be checked by also making evaluations using that belief in the rest of the game tree. Further discussion of this idea is left to the future work section in Chapter 6.

Demonstration 3: Clarification

This demonstration returns to look at the second agent's response to the risky strategy. This time, the clarification subtree is added as a third alternative, illustrated in figure 4.7.

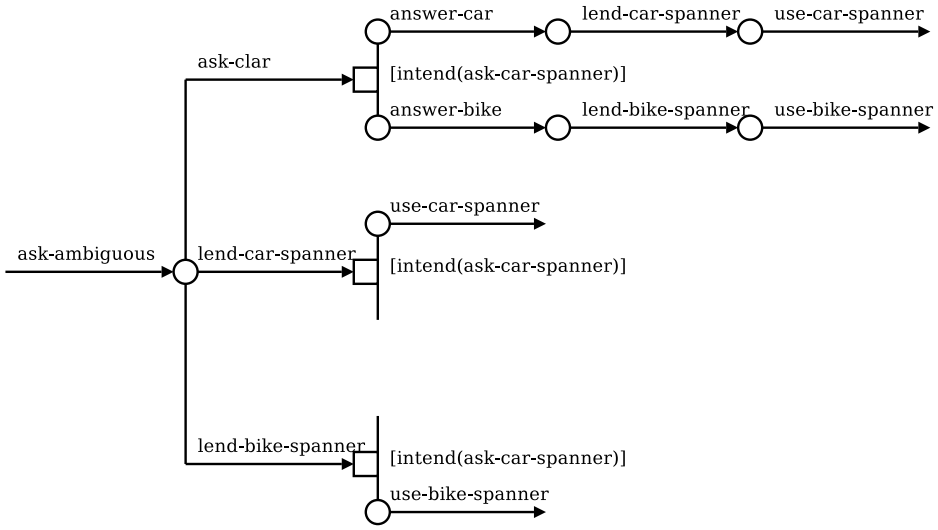

Figure 4.7: Game tree for second agent's response with clarification

The utility of the lower branches, lend-car-spanner and lend-bike-spanner is that same as before. Additionally, the utility of the clarify branch is:

$$
\begin{aligned}
& u(ask - c) \\
& + p.(u(ans - c) + u(lcs) + u(ucs) + reward) \\
& + (1 - p).(u(ans - b) + u(lbs) + u(ubs) + reward)
\end{aligned}
$$

$$
\begin{aligned}
= & -3 \\
& + p.(-1 - 10 + 100) \\
& + (1 - p).(-1 - 10 + 100)
\end{aligned}
\tag{4.9}
$$

$$
=86
$$

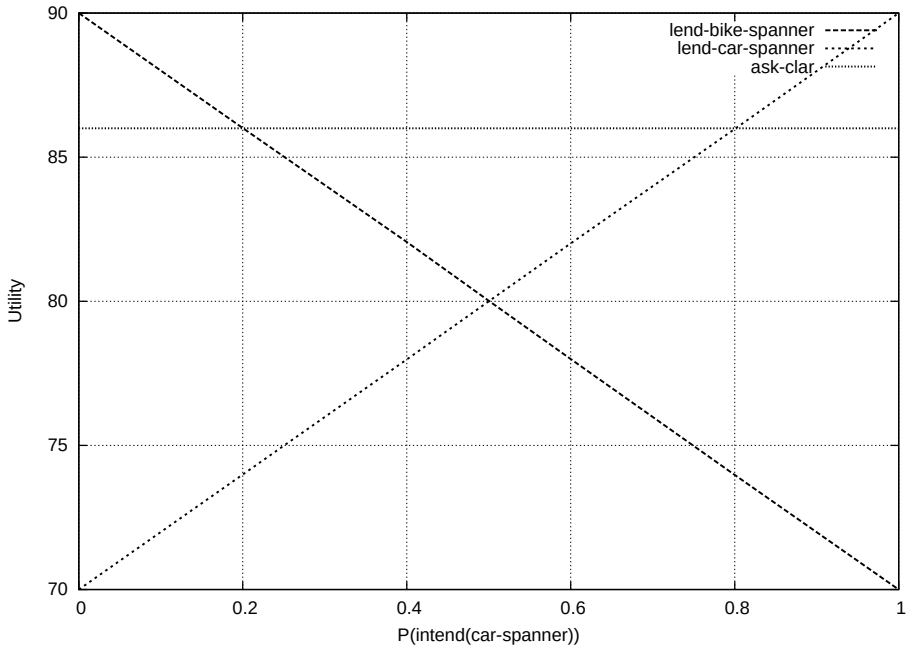

Figure 4.8: Utility of strategies against P(intend(car-spanner))

The plot for the planner's output is given in figure 4.8. Notice that in the middle region for p, between 0.2 and 0.8, clarify is the best strategy, but that

at the extremes, there is less risk, and so clarification is not appropriate.

Demonstration 4: The complete tree

In this section, the performance of the system using the complete game tree is described, repeated here in figure 4.9. Now that the clarify alternative has been introduced to the tree, it would be expected that the first agent could more readily drop the grounding initiative on the first move, and expect the second agent to pick it up at the second move.

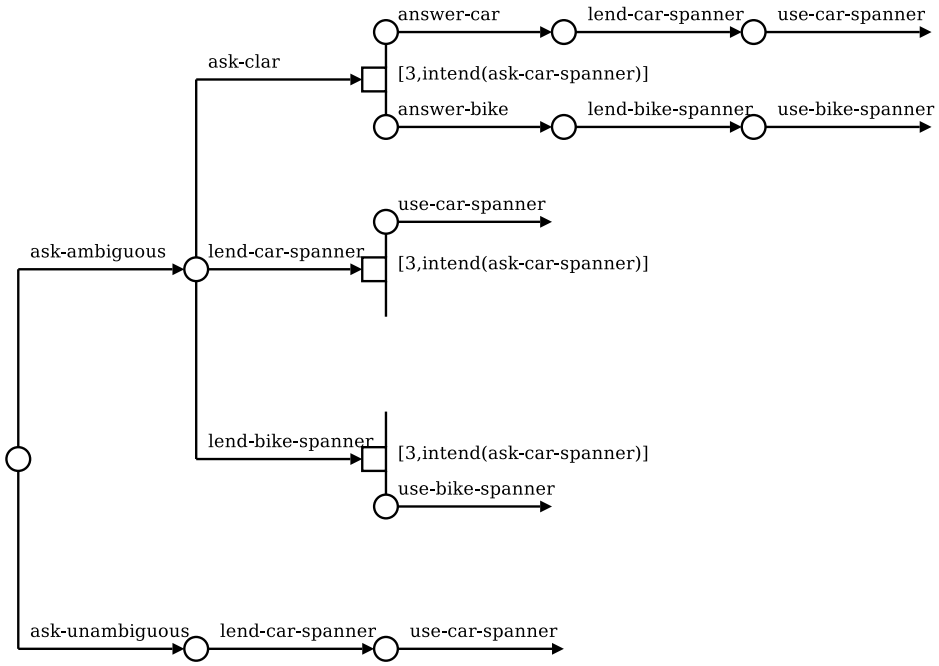

Figure 4.9: Complete game tree

As before, at the first move, the unambiguous alternative yields 80. The risky alternative now has three cases:

if $p < 0.2$

$$\begin{aligned} & u(aa) + u(lbs) + reward \\ =& -5 - 10 + 80 \\ =& 65 \end{aligned} \tag{4.10}$$

if $p > 0.8$

$$\begin{aligned} & u(aa) + u(lcs) + reward \\ =& -5 - 10 - 100 \\ =& 85 \end{aligned} \tag{4.11}$$

if $p \leq 0.2$ and $p \geq 0.8$

$$\begin{aligned} & u(aa) + u(clarify) \\ =& -5 + 86 \\ =& 81 \end{aligned} \tag{4.12}$$

Therefore, the planner is expected to produce a curve with a constant 65 in the left interval, a constant 85 in the right interval, and a constant 81 in the middle interval. The unambiguous strategy is at 80, and so for this configuration of the problem, the agent should drop the initiative in the middle interval and allow the other agent to pick it up at the next move. It is not hard to alter the utility function so that the agent should instead take the initiative in the middle interval. This is because the grounding effort is more or less the same, no matter who takes the initiative. Figure 4.10 is a plot of the planner's output, clearly showing the three intervals, and the close competition for initiative in the middle interval. Spill is once again evident here, with the system dropping the initiative for the best part of the middle interval for low values of n, but taking it up as n becomes high.

Figure 4.10 also provides an illustration of the need for a probabilistic belief model, rather than a logical one. A logical model must take one of the

two extremes on the probability scale, whereas a probabilistic model can take any value on this scale. Therefore, the decision surface that occurs around the point 0.2 would not be recognised using a logical model, even though the relative utility of the strategies varies greatly across this decision surface.

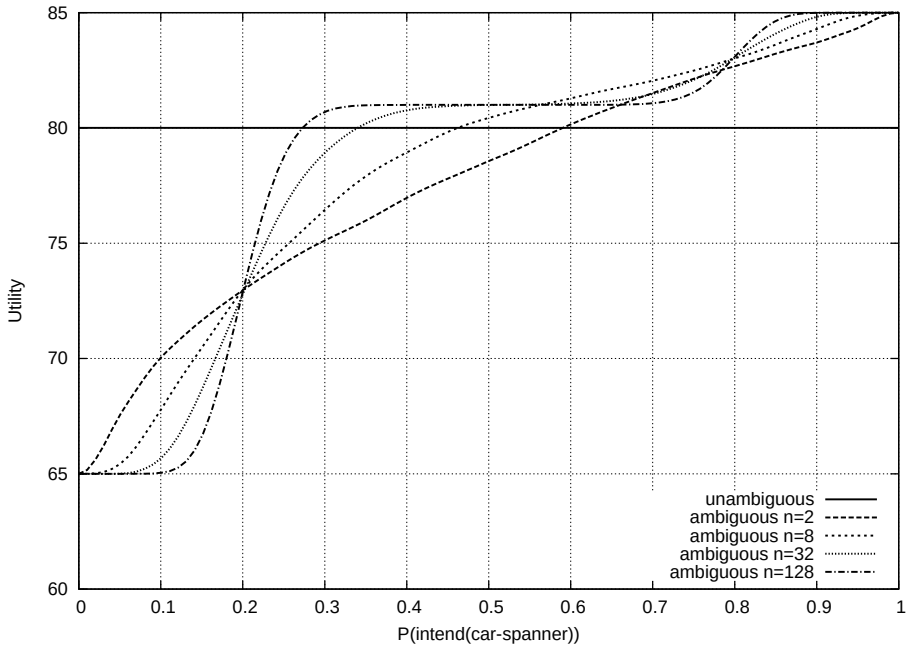

Figure 4.10: Utility of strategies against P(intend(car-spanner)) for four levels of error

It is interesting to compare the utility of the risky strategy in the complete game tree with that in the game tree without the clarification subtree. This comparison is plotted in figure 4.11, for $n = 8$. Notice that using clarification provides greater utility for the most part, except for the region to the right, where clarification is chosen in error by the second agent, since the first agent intends a car-spanner, and had clarification not occurred, the second agent would have taken its best guess instead, which would have turned out to also be the car-spanner.

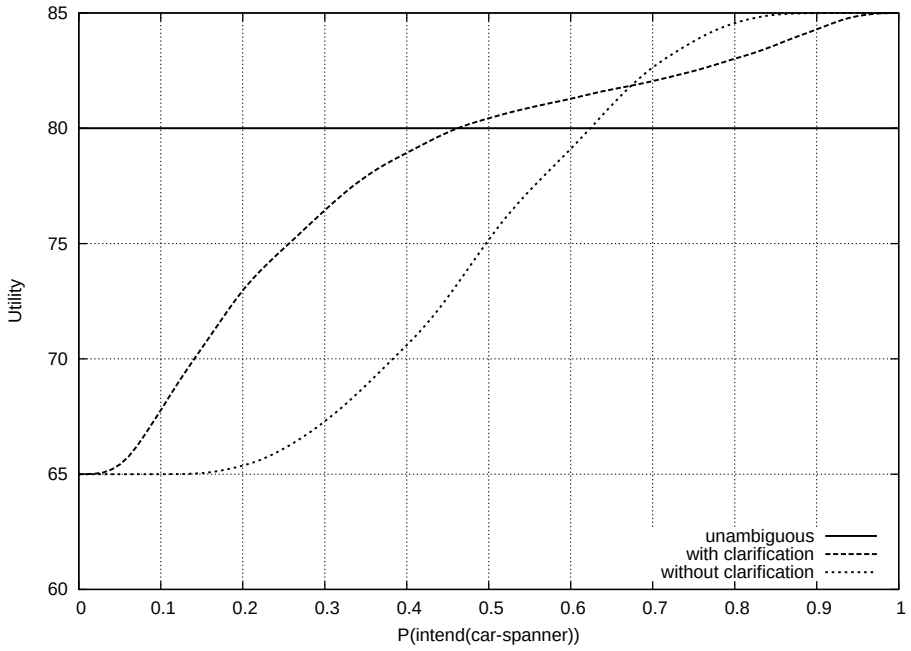

Figure 4.11: Utility of strategies against P(intend(car-spanner))

Demonstration 5: Comparison with fixed strategy

In this section the complete game tree described in figure 4.5 is used to demonstrate the gain in efficiency of the planner over one that does not adapt its strategy to a probabilistic belief model. The probability that the first agent intends to have a car-spanner is varied along the x-axis, and the second agent's belief about this intention is given an equal value. This is reasonable since it would be expected that the hearer would have learned that value through belief revision over the course of previous dialogues. To bind the planner to a fixed strategy, the game tree was pruned at the root node, forcing the first move to be fixed, but not restricting the rest of the moves.

In the left hand plot of figure 4.12, the utility gain is plotted for the fixed risky strategy. This graph was obtained by evaluating the pruned and

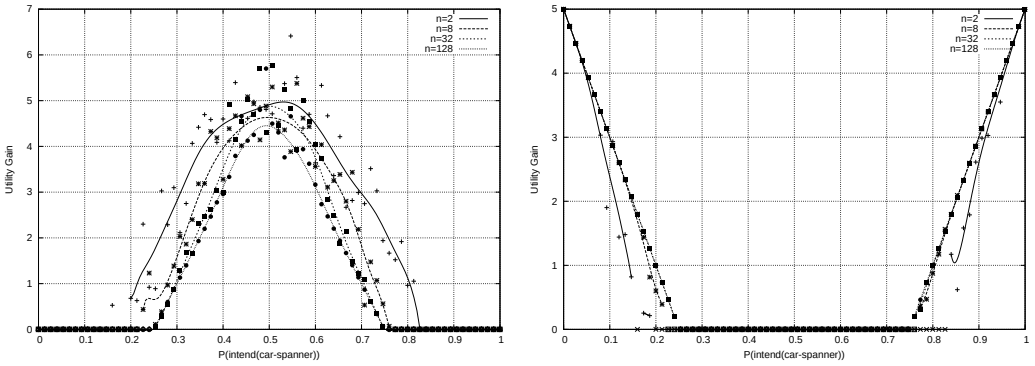

Figure 4.12: Utility of strategies against P(intend(car-spanner)) for different fixed strategies

unpruned game trees and taking the difference in utility. Notice that a gain is obtained by the use of an unambiguous strategy in the central region, since this strategy is more effective when there is risk in determining the agent's intention. Away from the centre, there is no gain (and no loss) since both the pruned and unpruned trees yield the ask-ambiguous strategy. On the right, the utility gain is plotted against the fixed non-risky strategy, where a gain is obtained by correctly taking a risk when the intention is more clear. In the centre of the graph, both trees take the ask-unambiguous strategy. These results show that the probabilistic planner obtains a utility gain of as much as 5 units over both of the fixed-strategy planners over a significant region of the belief space. This compares well with the maximum dialogue length, which is 20 units. There is also some difference in gain between different levels of sample error, with a high degree of error having a slightly negative effect on the performance of the probabilistic system. If the value for p(intend-car-spanner) varies over the lifetime of the system, a considerable gain is obtained by the planner over a fixed strategy system.

Now the performance gain plots are shown for the complete game tree,

including the clarification subtree, given in figure 4.9.

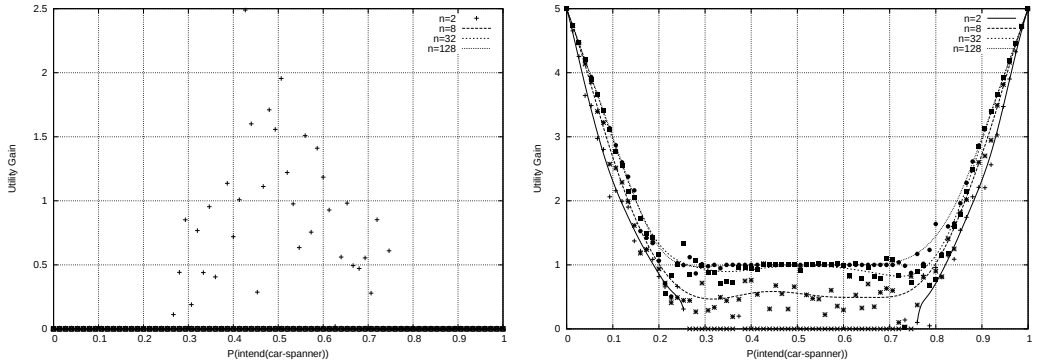

Figure 4.13: Utility of strategies against P(intend(car-spanner)) for different fixed strategies

In the left hand plot of figure 4.13, the utility gain is plotted for the risky strategy. Notice that little if any utility gain is obtained by the planner. This is because in the middle region, where the risky strategy is expected to fail, the responding agent picks up the initiative on the next move, who chooses a clarification subdialogue. The scattering of points just above the x axis for n=2 is explained by the high sample error, which pushes the utility of the risky strategy slightly below that of the non-risky strategy. The conclusion drawn from these results is that as long as one agent uses probabilistic reasoning, there is no need for the other one to. Particularly, a fixed-strategy dialogue system will perform just as well with a human partner, as long as the human partner can be relied upon to pick up the initiative. If he cannot be relied upon to do this, the gains are as in figure 4.12, and probabilistic reasoning makes a considerable difference to the planner's performance.

4.6 Example 2: Goal introduction problem

This section describes a practical problem that can be encountered in customer service dialogues. The problem is that of deciding to add a goal to the dialogue plan. Like the first example it is a problem of initiative, since either the first agent or the second agent could be the one to add the goal. The problem is one that is familiar to patrons of fast-food restaurants, where the customer initiates the dialogue by placing an order. In response, the agent might add new goals, like offering fries or a drink. This would be a valuable initiative if the customer were not aware that such things were available for sale, but would buy them if they were. On the other hand, it would be less valuable if most customers knew what was for sale, and rarely wanted anything other than a hamburger. Another example is at an airport check-in, where a customer may not believe that a window seat is available, and so may not waste time asking for one. On the other hand, the agent may believe that most customers do not want a window seat and so may not waste time offering one. This problem is again one of choosing between the riskier alternative of introducing a goal that might fail, and the less risky one of passing the initiative to the other agent. It is also a symmetrical problem since both agents make risky decisions in introducing the goal.

A plan library has been constructed for the window seat problem. This is described in figure 4.14. Notice that the first agent can begin with a goal of getting a window seat, or of getting any seat. The second agent can respond by taking the initiative and offering one directly, or it can drop the initiative and move with a pleasant chat about something else. There is a precondition to offering that the agent believes that it has an available seat. If the second agent takes the latter alternative, the first agent then has the option to pick up the initiative by asking for a window seat, or drop it and move instead

with a pleasant chat about something else. If it asks, then, if the second agent's precondition of having a seat is satisfied, it will give the first agent a window seat.

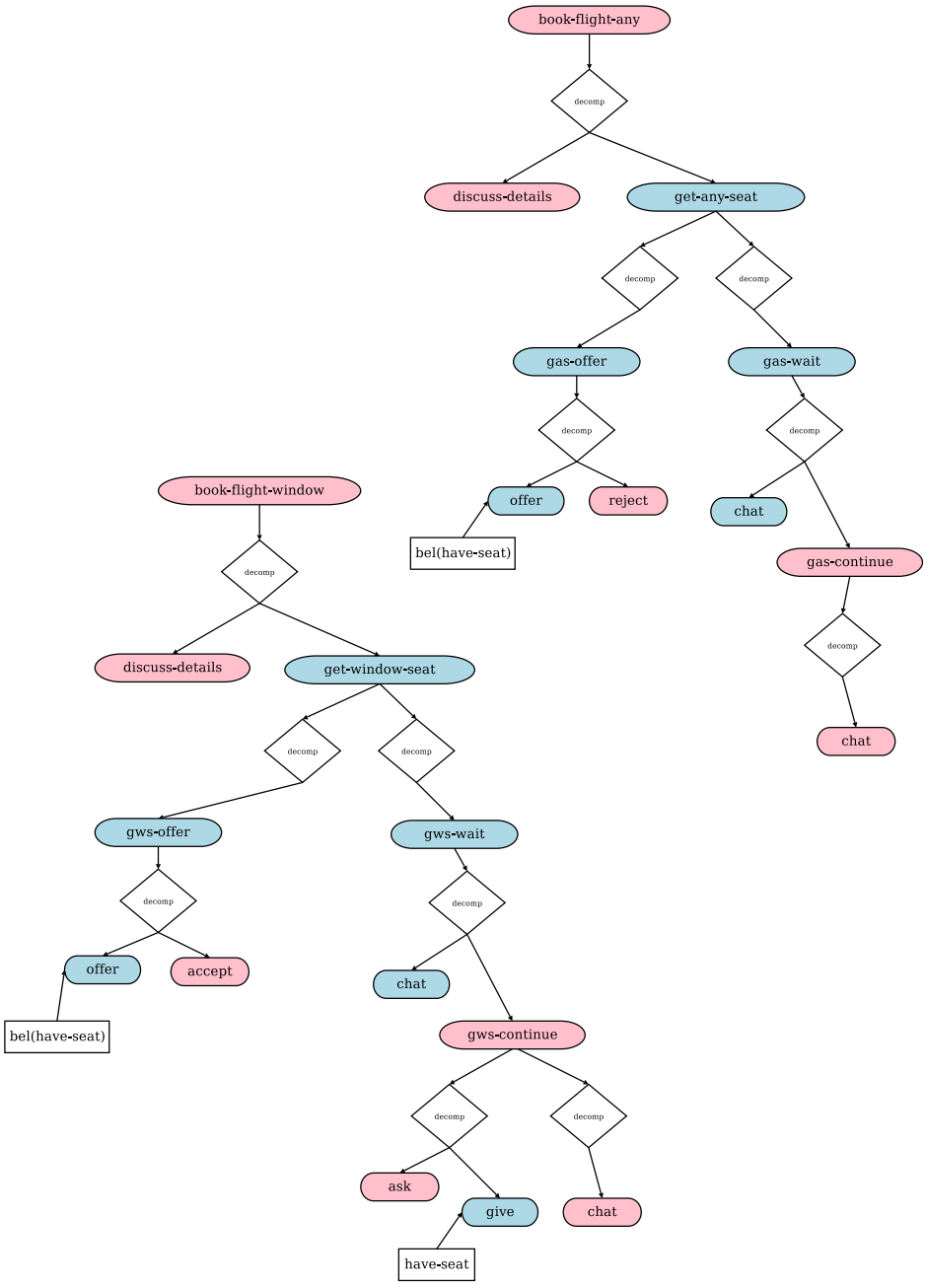

Figure 4.14: Plan library for goal introduction problem

From this plan library, the planner produces the game tree described in figure 4.15. Notice that in this tree the **bel(have-seat)** precondition causes the introduction of a chance node that determines whether the agent can **offer**. The **have-seat** precondition determines whether it can **give**. **ask** and **accept** only occur if the agent intended **book-flight-window**. Therefore a chance node is introduced before these acts. This chance node is generated by the parent intention rules.

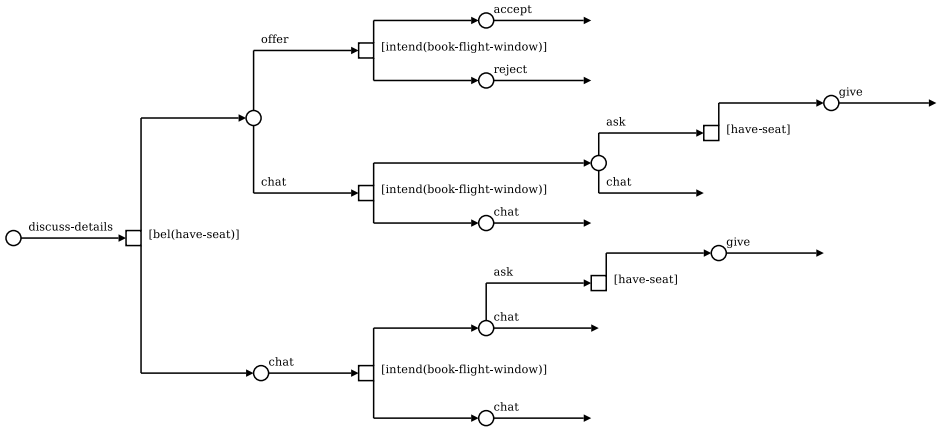

Figure 4.15: Game tree for the goal introduction problem

4.6.1 Demonstrations

The demonstrations for this problem follow a pattern similar to that of example 1. Sampling was not used, and so a belief model with no error is presumed. Unlike example 1, this problem has two variables, the user's intend(book-flight-window) and the agent's bel(have-seat). These beliefs occur at different levels of nesting. For example, bel(have-seat) needs to be evaluated at level 2 corresponding with its occurrence the second level of the game tree. it also is evaluated at level 4, corresponding with its occurrence the fourth level of the game tree. For the purpose of simplifying the

demonstration by reducing the number of variables, it is assumed that beliefs are mutual, which renders level 2 and level 4 beliefs equal, and reduces the problem to one of two variables. This is a reasonable assumption in the usual circumstances that acts are always observable to both agents and so all the belief revisions that develop their belief sets make mutual updates (see Section 3.4.7). The following demonstrations present and explain the output of the planner.

Demonstration 1: Lower branch

This demonstration explores the form of the utility function in the lower branch of the overall tree, from the second agent's perspective. There is in fact only one choice node in the subtree that has more than one alternative, and for it a decision surface is expected between ask and chat. The only source of variation in each of the alternatives is the have-seat node, and so the following solution is found for the decision surface:

$$\begin{aligned} & u(ask) = u(chat) \\ \Rightarrow & -5 + 85 + 10q = 87 \\ \Rightarrow & q = 0.6 \end{aligned} \tag{4.13}$$

Along the intend(book-flight-window) axis, the utility function is expected to be linear since there is a weighted sum over the two invariant sub-branches. The plot from the planner's output is shown in figure 4.16. The decision surface is just visible.

To further verify the results, the corner points of the belief space were each checked, and were found to be consistent with the planner's output. intend(book-flight-window) is in the x position, and bel(have-seat) is in the y position.

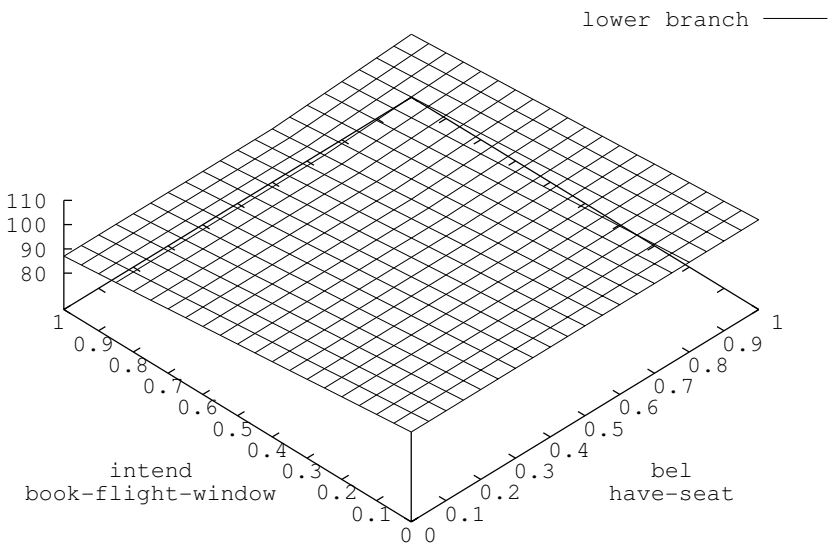

Figure 4.16: Utility of strategies against P(intend(book-flight-window) and P(bel(have-seat))

- At 0,0, the user doesn't intend to have a seat, chat is expected to be chosen with an outcome of 100 for the reward plus a negative cost of $1 + 1$ for the chats, totalling 102
- At 0,1 the user doesn't intend to have a seat, so again chat is chosen giving 102
- At 1,0, the user intends to have a seat but doesn't ask since he believes there is none available, giving $85 + 1 + 1 = 87$
- At 1,1 the user intends to have a seat and does ask, giving 100 for success $+1$ for chat -5 for ask and -5 for give, totalling 91

Demonstration 2: Upper branch

In the upper branch of the game tree, there are two alternatives available at the second step of the plan. The offer branch is expected to be linear with respect to intend(book-flight-window). In fact the line is flat since each branch has equal utility. It is expected to be constant with respect to bel(have-seat). Therefore the offer branch is constant with respect to both variables.

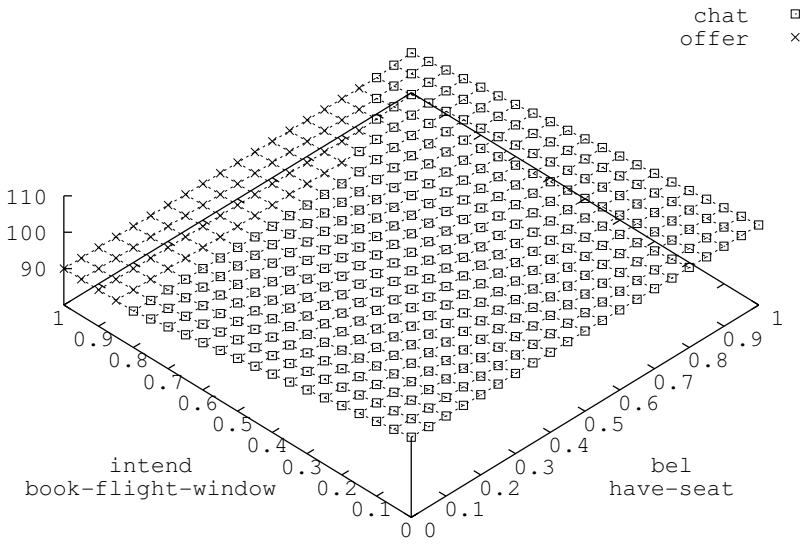

Figure 4.17: Utility of strategies against P(intend(book-flight-window) and P(bel(have-seat))

The chat branch is followed by a chance node, and so its utility is a weighted sum of two sub curves. The chat sub curve is a constant 101. In the other branch, the agent has a choice between chatting and asking. Chatting gives a constant 86 whereas asking is linear between 80 and 90. Therefore there is a decision surface at 0.6. The plot is given in figure 4.17,

with different point styles for the chat and offer alternatives, highlighting their decision surface.

As further verification, the corner points were checked, taking intend(book-flight-window) in the x position, and bel(have-seat) in the y position.

- At 0,0, 102 is obtained, since the lower branch is taken
- At 0,1, 102 is obtained, since the lower branch is taken
- At 1,0, 87 is obtained, since if the user asks and doesn't get 79 is obtained. but chat obtains 87, so chat is chosen
- At 1,1, 91 is obtained, since the user asks and gets and so the system chooses chat

Demonstration 3: Overall utility

The utility plot for the whole game tree is plotted in figure 4.18.

Demonstration 4: Distribution of initiative

This demonstration illustrates the distribution of initiative over the belief space. In the following, the frequency with which a seat was offered was plotted. It was expected that the agent would offer a seat if the intention value was high, but that when the belief about having a seat value was also high, that the agent would override this decision since the other agent can be expected to take up the initiative instead. The system behaved as expected, seen in figure 4.19.

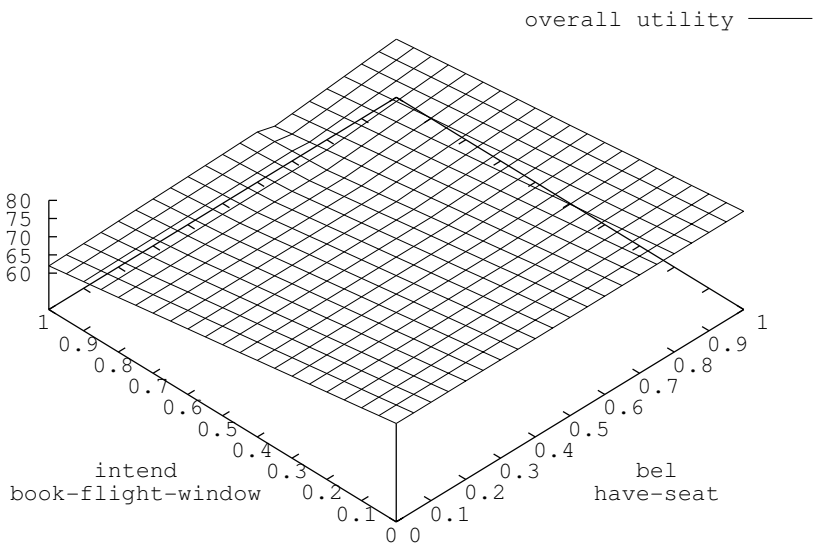

Figure 4.18: Utility of strategies against P(intend(book-flight-window) and P(bel(have-seat))

Then, the frequency with which a seat was asked for was plotted. This was expected to happen when bel(have-seat) was highly valued by the user. This is shown in figure 4.20

Finally, the total of the contributions of initiative from each of the agents is given in figure 4.21.

Demonstration 5: Efficiency

The system was compared with a fixed strategy planner, using a method very similar to that used in example 1. Taking just the upper branch of the game tree, a comparison was made for the decision between offer and chat that is made by the agent. The subtree was pruned to produce two fixed-strategy trees. In figure 4.22 (a) the gain that the planner obtains over the

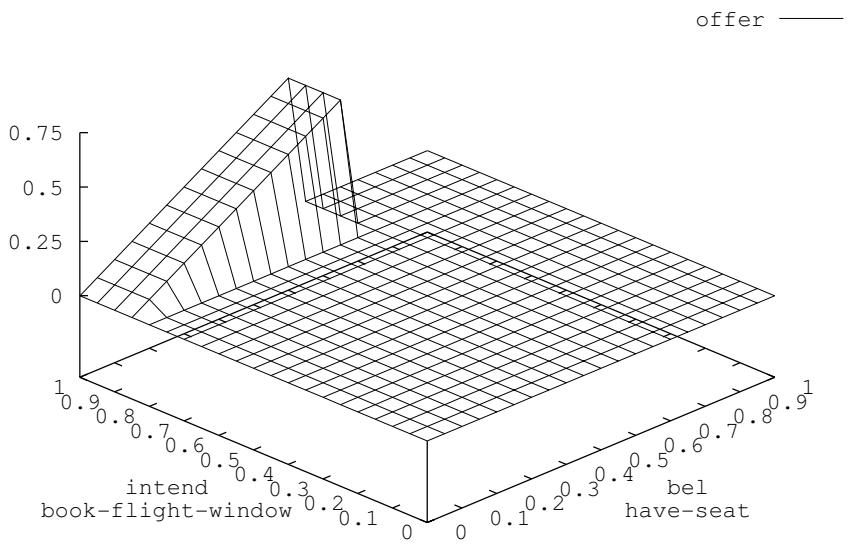

Figure 4.19: Density of initiative against P(intend(book-flight-window) and P(bel(have-seat))

fixed strategy of chat is plotted. In figure 4.22 (b) the gain that the planner obtains over the fixed strategy of offer is plotted. The squared point style indicates where a gain is obtained. Against the chat strategy, the planner obtains a small gain of up to 1.8, in a small region of the belief space. This is expected since in this region, it is almost certain that a window seat is intended, yet the user believes that there is none available. Against the offer strategy, a larger gain of as much as 12.0 is obtained. The maximum length of the dialogue is 15 units, corresponding with the longest path in the game tree, and so for belief models which cross the decision surface during their lifetime, there are some significant gains to be made.

A second configuration of the problem was explored, since in the first configuration, the amount of initiative over the belief space is perhaps un-

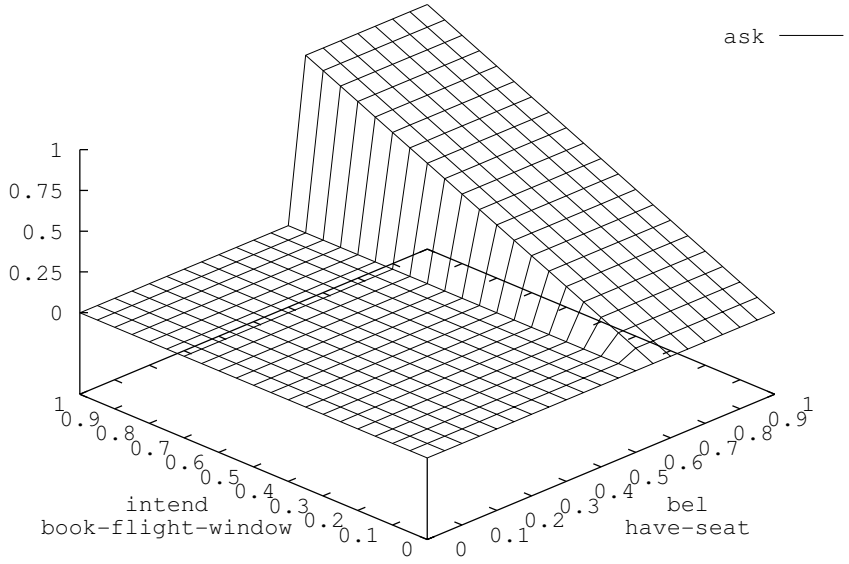

Figure 4.20: Density of initiative against P(intend(book-flight-window) and P(bel(have-seat))

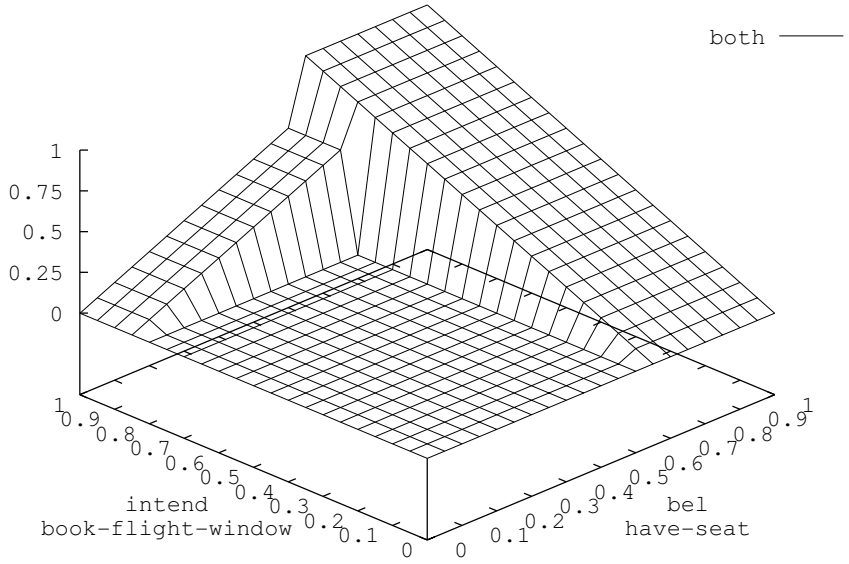

Figure 4.21: Density of initiative against P(intend(book-flight-window) and P(bel(have-seat))

realistically low (see figure 4.21). The plot shows that the agent only offers window seats when it believes it very likely that the user wants one. In the second configuration, the reward for an agent who wants a window seat but ends up not obtaining one has been dropped from 85 to 65 to encourage initiative. As a result, the initiative distribution shown in figure 4.23, was obtained, where the agent only fails to offer a window seat when it is very sure that the user does not intend to have one.

The efficiency of the planner for this configuration is a little better than that obtained for the first configuration (figure 4.24). Against the chat strategy, a maximum gain of 4.60 was obtained, and against the offer strategy, a maximum gain of 12.0 was obtained. This is a good fraction of the total dialogue length of 15.

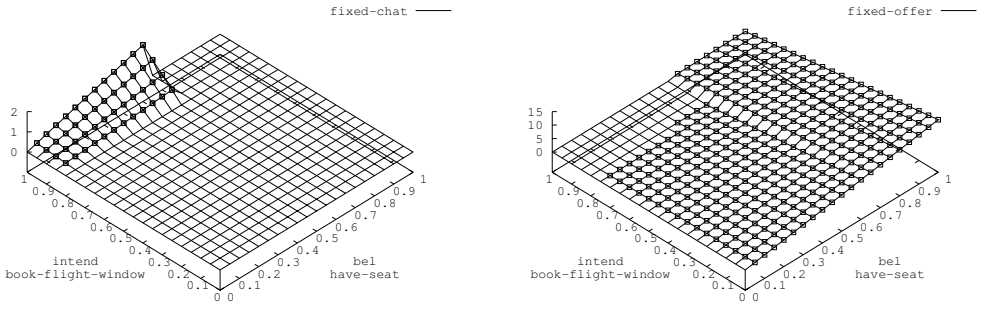

Figure 4.22: Comparison of planner with each fixed strategy against P(intend(book-flight-window) and P(bel(have-seat))

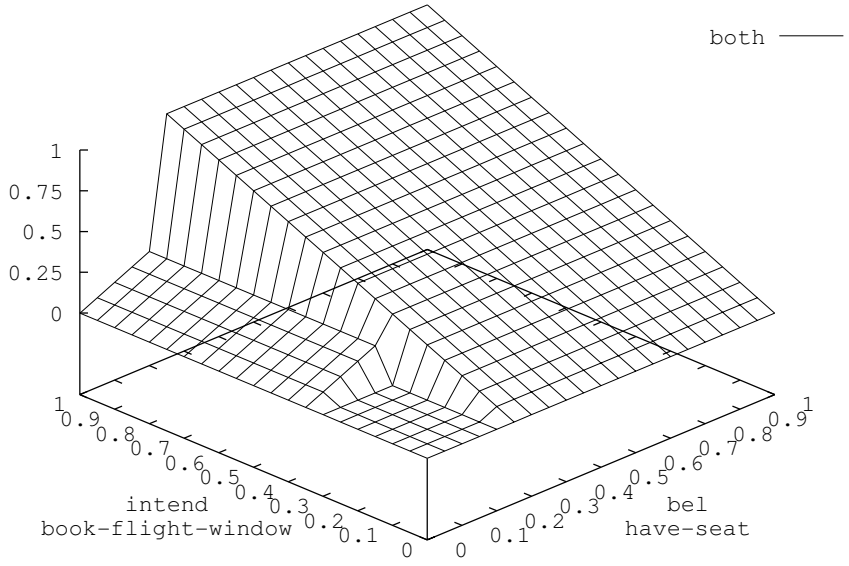

Figure 4.23: Density of initiative against P(intend(book-flight-window) and P(bel(have-seat))

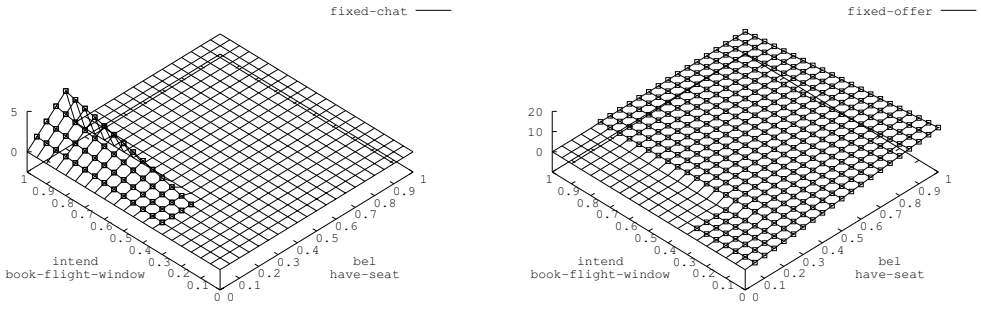

Figure 4.24: Comparison of planner with each fixed strategy against P(intend(book-flight-window) and P(bel(have-seat))

4.7 Belief model acquisition

In this section, user model acquisition is demonstrated by showing how the belief revision component of the system is used to adapt a stereotype model to a sequence of dialogues between the system and a user. Consider again the window seat example of the last section. In figure 4.25 the system must decide whether or not to offer a window seat. This depends on its model of the user's intention to have a seat, and on its model of the user's belief about whether the system has a window seat available. It is assumed in this example that there is one user, and that there is a sequence of dialogues. A stereotype model is employed to capture a user, who on some occasions wants a window seat, and on others does not. Similarly, it is assumed that the user treats the system as varying from day to day, with availability of a window seat randomly determined according to a certain probability. Therefore the user too employs a stereotype, and the system must make an estimate of this stereotype. Over the course of a sequence of dialogues, the stereotype model comes to estimate the expected belief state. In this example, stereotypes that are nested to all levels are used, since the agents mutually believe that each is modelling the other using a stereotype model.

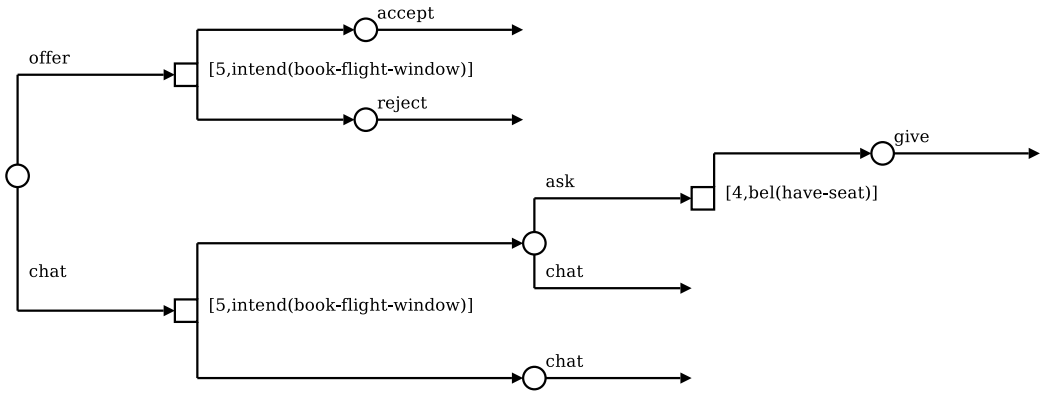

Figure 4.25: Game tree for agent's choice between offer and chat

act	precondition
offer	have-seat
give	have-seat
dontgive	not(have-seat)
ask	intend(book-flight-window)
accept	intend(book-flight-window)
reject	intend(book-flight-any)

Table 4.1: Preconditions used by the belief revision mechanism

The system was set to use a "decaying" average to compute the stereotype. That is, the stereotype was a 90%/10% mix of the previous and revised value, with the revised value obtained by starting with the previous value and performing belief revision on it using the current dialogue. This ensured that the most recent evidence had a greater weighting. Appropriate preconditions were set up for the plan rules to ensure that the system could make all of the necessary inferences about beliefs and intentions. These rules are given in table 4.1

The "dry-land" algorithm is particularly helpful in this example, for the dialogue [discuss-details,chat,chat] (see figure 4.15). The final chat has no precondition and so ordinary belief revision does nothing. However, it can be explained by both of the user not intending a window seat and the user not believing that one is available. The dry-land algorithm adopts the simplest combination of these. It is also useful at the offer/chat decision. For example, if the system were to choose chat, the system would expect the user to update his model with an explanation that the system believes that either the user does not want a window seat or that the user believes the system has a

window seat and will therefore take the initiative himself. That the dry-land algorithm is useful twice in this example indicates that it may be significantly important in other types of dialogue, but these are yet to be investigated.

Some demonstrations are presented now to show how the stereotype model adapts over the course of a sequence of dialogues. Demonstration 1 looks at the updating of the system's model of the user's model, in response to the system's act on the first turn. Demonstration 2 looks at the system's revision of its own model in response to the user's act on the second turn. Demonstration 3 looks at the sequence of belief states that result from a sequence of dialogues.

4.7.1 Demonstration 1

This demonstration looks at the revision of the system's model of the user's stereotype model of the system, in response to the first act in the dialogue. First, the offer alternative was explored. Recall figure 4.17 which shows the decision surface between offer and chat. To force the system to choose offer, bel(have-seat) was set mutually to 0.2, whereas intend(window-seat) was set mutually to 0.8. The belief revision mechanism revises beliefs at all levels, and therefore every second level remains identical through the course of the dialogues. The user's belief revision mechanism would be expected to update bel(have-seat) since it is a precondition to offer. Also it would be expected that the "dry-land" algorithm would maintain the user's beliefs about the system by pushing the belief state towards the centre of the offer region. It turns out that in the user's model of the system, intend(window-seat) increases to 0.833 in this dialogue, approaching the approximately 0.9 centre. bel(have-seat) approaches the approximately 0.4 centre by increasing to 0.311. Then, the precondition is observed, and the corresponding belief is

set to 1, by ordinary belief revision.

The chat alternative was also explored. To force the system to choose chat, bel(have-seat) was set to 0.8, whereas the intend(window-seat) was set to 0.1. It would be expected that these beliefs would be pushed towards the centre of the region in the user's model. That is what happened. intend(window-seat) moved to 0.16, and bel(have-seat) moved to 0.66.

4.7.2 Demonstration 2

This demonstration illustrates the system's revision of its stereotype model of the user, in response to the user on the second move. Each of two cases is described in the following two sections, corresponding with whether the system chose offer on the first move or chat.

Upper branch

In the upper branch of the game tree, "offer" is chosen by the system, and the user can respond with "accept" or "reject". For the dialogue [offer,accept], it becomes mutual that bel(have-seat), and that intend(book-flight-window), since these are preconditions. Therefore both of these should increase in the updated decaying-average model, possibly crossing the decision surface so that the system will not take the initiative the next time around. For [offer,reject] bel(have-seat) should increase, whereas intend(window-seat) should decrease. The dry-land algorithm is redundant in this case, since both bel(have-seat) and intend(window-seat) are updated by precondition inference.

An example point was taken, [bel(have-seat),intend(window-seat)] = [0.5,0.9], and the belief model changed to :

bel(have-seat) = 1 was mutual and intend(window-seat) = 1 was mutual for [offer,accept]

and

bel(have-seat) = 1 was mutual for and intend(any-seat) = 1 was mutual for [offer,reject].

Lower branch

In the lower branch of the game tree, "chat" is chosen by the system. If the user responds with "chat", the state is expected to drift towards bel(have-seat) being false and intend(window-seat) being false, since these are both explanations for the user deciding to chat. This change would be made by the dry-land algorithm. If the user responds with ask, the system responds with give (since its base belief is that it does have a window seat), then both beliefs are set to 1 by precondition inference, with the decaying average possibly moving across the decision surface. Starting at the point [0.5,0.5], the user chose chat, resulting in belief revision by the dry-land algorithm to: p(intend(window-seat)) = 0.486 for the system's belief about the user and p(bel(have-seat)) = 0.321 for the system's belief about the user's belief about the system.

4.7.3 Demonstration 3

In this demonstration, a sequence of dialogues is processed by the system and the sequence of resulting belief states is recorded. Although acquisition happens at all levels of the belief state, beliefs remain mutual, so just the system's belief about whether the user intends a window seat, and the system's belief about whether the user believes the system has a window seat are recorded. The system starts in a state of [bel(have-seat),intend(window-

dialogue	choice	bel(have-seat)	intend(window-seat)
		0.500	0.900
1	offer	0.550	0.810
2	offer	0.595	0.729
3	chat	0.596	0.711
4	chat	0.586	0.703
5	chat	0.571	0.685
6	chat	0.573	0.670
7	chat	0.555	0.671
8	chat	0.551	0.664

Table 4.2: Belief model states for a sequence of dialogues

seat)] = [0.5, 0.9], with the consequence that offering is dominant. The system's base belief was that it has a window seat. Contrary to the system's belief model, the stereotypical user does not want a window seat, and so as each user drawn from the stereotype refuses the offered seat, the belief state eventually crosses the decision surface, as the intention drops from precondition inference and the system begins to decline the initiative. Notice that the have-seat belief increases even though the user declines the initiative, since it is a precondition to offering. Once the system stops offering, the user continues to decline the initiative, and the system responds employing the dry-land algorithm to revise downwards both the user's intention to have a window seat and the user's belief that one is available.

4.8 Related work on initiative and simulation evaluation

Guinn [30] developed an initiative planning system based on Smith and Hipp's (Section 2.6) meta-level planner. Smith and Hipp's planner employed a user model containing plan rules which were applied to construct the user's piece of the dialogue plan. Alternative initiative settings were available in choosing these. Guinn added a probabilistic model of success to the plan rules, using "factors" associated with the alternative plans available to the agents. With different beliefs about the factors, the agents would produce different estimates of the probability of success of different plans, and therefore there would be conflicts over whether to take the initiative for a plan. The agents could use negotiation dialogues to communicate factor values so that conflicts would be resolved. A simulation evaluation technique was used to show the performance improvement obtained in dialogues where factors are evaluated. This was a similar technique to the one used for the two examples in this chapter, where beliefs were assigned to agents and the resulting dialogues measured.

4.9 Conclusion

The purpose of this chapter was to evaluate and demonstrate the planner. In Chapter 1, objectives were set to develop a dialogue planner suitable for use in a dialogue management system that would be easy to use, and at the same time, demonstrate an efficiency advantage over systems that do not take advantage of a user model. The two examples in this chapter demonstrate ease of use. For further evidence, an example of an input file for example

1 is given in the appendix. Therefore, dialogue problems can be specified just as well as for a state based system or for a system that uses a phrase structured dialogue grammar. The question of whether the system is as efficient as other candidates has been partially answered. The two examples showed that a significant efficiency gain can be obtained in some dialogues by exploiting a user model. The only question that remains is whether this system is competitive among dialogue systems that exploit a user model. For those user modelling systems that use a logical, rather than a probabilistic model ([5, 37, 51, 10, 38, 54, 69, 2, 72]) , the answer is yes, since these systems cannot distinguish the negation of the difference in utility that occurs between alternatives across a decision surface. The other category of systems that use an (implicit) user model is that of reinforcement learning systems. While reinforcement learning is very useful for routine dialogues ([64], [76], [58]), it requires plenty of training data. In a routine situation, there would be little point in using planned dialogue. On the other hand, a planning system can make intelligent use of training data in novel situations. The revision of a belief for one plan can have a positive effect on another, novel plan, once the two plans share that belief as a precondition. In particular, negotiations over complex domain plans may require choices of negotiation acts that depend on a large state space of intentions and beliefs for that domain-level plan. The application of the planner to this sort of problem will be discussed in the next chapter. The efficiency of the planner compared with that of a reinforcement learning system has yet to be shown. In the future work chapter plans for experiments that compare each approach will be given.

This chapter has fallen short in not performing at least some evaluation in a human rather than a simulated setting. While simulation is very flexible

and gives detail results, it seems that even a small experiment in a human setting would produce valuable evidence about the planners expected performance in such a setting. More discussion of such evaluation will be given in the future work section.

Chapter 5

Planning of Negotiation Dialogue

5.1 Introduction

In negotiation dialogue information is exchanged between agents that supports their decision about a domain-level plan.The term 'negotiation' is usually taken to mean a discussion that is used to obtain an agreement, often between self-interested parties. Here it is used to mean discussion between fully cooperative parties as well, and the discussion encompasses communication of proposals to agree upon as well as the communication of information that is used to better understand what the best proposal might be. The meta-level planners described in Section 2.6 are used to plan such dialogues, but those planners do not consider the efficiency of the negotiation acts. Instead, they search over the space of domain plans, generating negotiation subdialogues whenever questionable preconditions are encountered. They use logical belief models, and therefore consider and generate dialogue about all potentially valid plans equally, no matter how good those plans might be.

Using these planners, one plan whose preconditions are extremely improbable would be just as much a candidate for discussion as one whose preconditions are almost certain to be satisfied. As an extreme example, such planners would consider negotiation about a plan in which a brain surgeon wears a blindfold, considering that it is unlikely yet still possible that the operation will be a success. There may be many such unreasonable candidates, but few reasonable ones. Such planners would therefore fail to negotiate a good domain-level alternative in a reasonable amount of time.

In contrast, an efficient planner for negotiation dialogues will be described in this chapter which discounts for discussion all but the domain plans that have a reasonable chance of success, and only passes information that will significantly improve the quality of those plans. The planner uses a set of negotiation acts, which use as their subject the beliefs and intentions of the speaking agent. Each of the acts has a pragmatic definition in which their meaning is described in terms of preconditions on the agent's mental state, and so can be planned using ordinary plan rules, in the same way as the domain-level plan. These acts constitute a repertoire of acts that are common to dialogue no matter what the subject, for example, a question-answer pair is a very common subplan in all sorts of dialogues. While they can be specified and planned in much the same way as domain-level acts, they should be built in to the planner rather than given in the input file. They should be automatically plannable too, with the agent being free to mix negotiation subdialogues as appropriate with those that are derived from the domain plan rules given by the user.

The negotiation planner is motivated by the sort of problem where two or more agents must coordinate their actions in some plan. For example, a pair of experts may wish to collaborate in routing trains in a transportation

network [2], or a team of robots may want to coordinate themselves in constructing a car. In these problems, there is often a "team talk" that precedes execution of that plan, where the team members must establish who is capable of what, which resources are available and what the state of the environment is, how the goal will be decomposed, and how the subtasks will be ordered.

Constructing a negotiation planner requires only minor modification to the existing domain level planner, adding just the facility to add negotiation acts to the game tree. The evaluation module, and the belief revision module remain identical to those described in chapter 3, and the language used in the input files to describe the domain level plan rules is the same as that used previously to describe dialogue plan rules.

In Section 5.2, the principles on which the value of the negotiation acts is based are explained using a simple example. Then, in Section 5.3, formal pragmatic definitions are given for the negotiation acts, using STRIPS plan rules. Section 5.4 argues that these are an appropriate set of acts. Then, Section 5.5 presents examples of each of the negotiation acts, demonstrating that each is a necessary member of the agent's repertoire, and showing how the utility of negotiation acts is sensitive to the probability values of the belief model. The set of negotiation acts is extended in Section 5.6 to include insincere acts as a complement to the sincere acts in settings of self-interest. Finally, the work described in this chapter is compared with similar previous work by Gmytrasiewicz and Durfee [25].

5.2 Value of information

The negotiation planner is based on the idea of value of information, which is that an agent's expected utility can be increased by gaining information of the other agent's beliefs. This idea comes from decision theory [44]. For example, given a game tree with alternatives a and b at a choice node, and a belief with probability value p, the utility of each of a and b would be some function of p, since the belief may be used to evaluate a chance node in the subtrees of a and b. If (and only if) there is a decision surface between a and b in the belief space defined by p, the deciding agent would be interested in the value p, since it affects his choice between a and b. There may be many such beliefs to be resolved, perhaps leading to a long negotiation dialogue. The negotiation ends when the cost of the remaining negotiation dialogue outweighs the benefit that can be obtained from the information it provides. At this point the agents should begin executing the domain-level plan. It is possible for negotiation to interrupt the domain-level plan once again. If an agent takes an unlikely alternative in the domain-level plan, chance nodes that were previously too remote, due to their weighting, to outweigh the cost of the negotiation dialogue may become more interesting.

5.2.1 Value of information example

As a concrete example of value of information, consider a problem of designing a kitchen assistant robot who must negotiate efficiently with a user about the various alternatives in jointly preparing a meal. Figure 5.1 illustrates the plan library for this problem, which is duplicated to populate each level of the belief model. The problem is one where agent 1 must decide whether to use some eggs in making an omelette. To make the decision it must find out

whether agent 2 has fruit, since if agent 1 uses up the eggs, and agent 2 has fruit, his decision would cause agent 2's make-pavlova plan to fail.

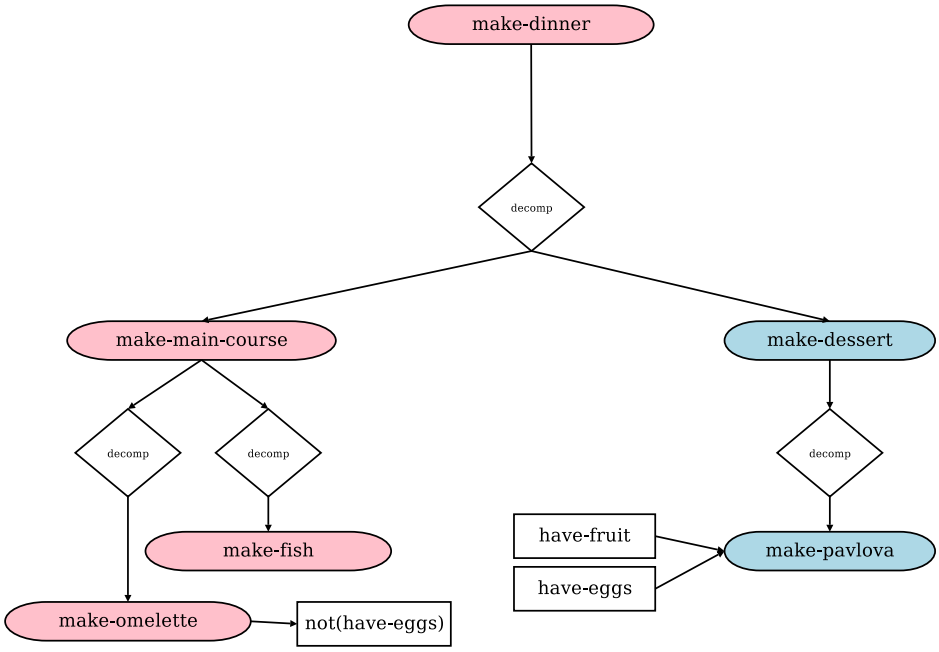

Figure 5.1: Plan library for a value of information example

The utility function for the problem is defined by:

```
utility(make-omelette,200).
utility(make-fish,150).
utility(make-pavlova,100).
```

The game tree generated for this problem is illustrated in figure 5.2.

This game tree matches the general form of a value of information problem, in that there are two alternatives, whose utility is a function of some belief, and there is a decision surface in the belief space corresponding to that belief. In particular, the utility for the upper branch of the game tree is given by:

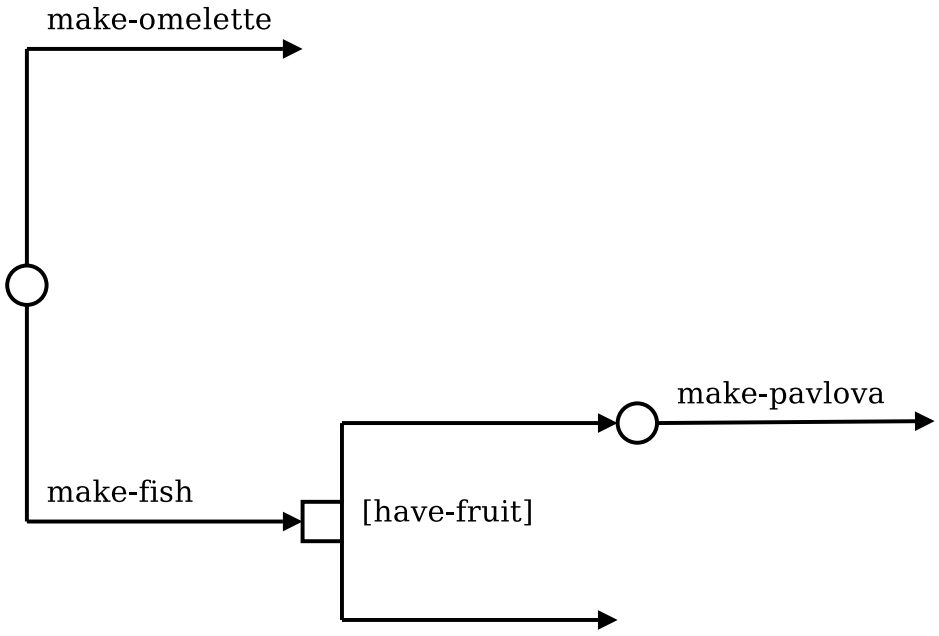

Figure 5.2: Game tree for the value of information example

$$200 \tag{5.1}$$

For the lower branch, the utility function is

$$150 + p.100 + (1 - p).0 \tag{5.2}$$

At the **true** extreme of the belief, the agent chooses the lower branch. At the **false** extreme of the belief, the agent chooses the upper branch. A decision surface occurs where the two functions have an equal value, at 0.5. If the information about the belief is given to the agent, the expected utility value is a weighted sum of these two extremes, according to the prior value of the belief. This is:

$$p.200 + (1 - p).250 \tag{5.3}$$

If functions 5.1, 5.2 and 5.3 are plotted together, the result is as shown in figure 5.3. The area enclosed in the triangle indicates the utility gained from obtaining information. The gain is a maximum in the middle of the range, where the belief is least certain.

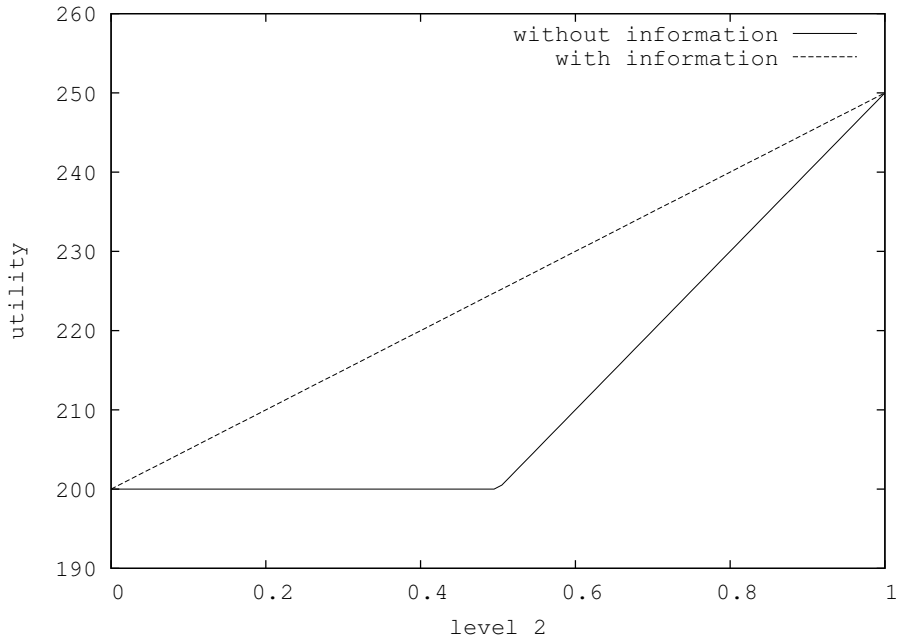

Figure 5.3: Comparison of expected utilities before and after information is obtained

5.3 Negotiation acts and their pragmatic definition

There are a number of negotiation acts that can reveal an agent's belief state. Selection of an appropriate repertoire of acts was motivated by a number of

desirable properties:

- **Adequacy** The repertoire should be strong enough that it can, if necessary, be used to communicate all of each agent's beliefs, leading to a perfectly correct belief model, and therefore the most efficient plan possible given the agents' capabilities and beliefs about the environment. If this were not possible, there would be something lacking in the expressive power of the repertoire.

- **Efficiency** Adequacy does not guarantee efficiency, but the acts should communicate as much as possible with the least dialogue cost.

- **Necessity** The repertoire should not contain unnecessary members. This property might be proved by finding example problems in which one member dominates all of the others. The purpose of this property is to guarantee that the repertoire of acts is divided into basic, orthogonal acts, with no redundancy of expressive ability, therefore obtaining the simplest possible set of acts.

- **Realism** The repertoire should correspond with the repertoire of negotiation acts seen in corpora of natural language collaborative planning dialogues, so that the expectation of the user's contribution to the dialogue is correct. This allows the planner to generate correct expectations about subsequent turns when the act for the current turn is being chosen. Being able to predict the user's act also implies being able to understand his utterance. The acts chosen by the planner must also be realistic so that the user can generate similar expectations and understand the system.

- **Simplicity** The negotiation acts should refer to simple expressions of

the elements of the agent's mental state. Since the agent's decision rests on the evaluation of game trees using a belief set, the only available candidates are descriptions of beliefs and descriptions of the intentions that come from the evaluation of the game tree. Acts other than these could only express some combination of the elements of the mental state, offering no efficiency gain over the elemental acts. The acts should also have a simple pragmatic form, being definable by STRIPS rules, and therefore requiring no more than the basic forms of belief revision that are already available to the planner.

The selected set of negotiation acts is now specified using STRIPS plan rules. By using STRIPS rules the repertoire can be formally specified and programmed within the plan-rule language of the domain-level planner. The repertoire was chosen by enumerating the different STRIPS rules that could be used for an act. Only two were found, one whose precondition refers to a belief of the agent, and one that refers to its intention. These respectively correspond with informing and proposing types of act. The control structures were also investigated, leading to different ways of assembling subdialogues with these acts. For example, in a question and answer pair, the questioner may intend that the hearer unconditionally answers his question, in which case one control structure is used, or he may intend that the hearer choose between answering his question and something else. The control structures are expressed using different rules of decomposition for the subdialogues. Each of the acts that emerged from this investigation was checked for adherence to the properties listed above.

- **pass** allows the agent to pass the turn without giving any information. It is defined by the following plan rule:

```
name:             pass
parameter:        {}
precondition:     {}
effects:          {}
decomposition:    {}
```

Since pass has no preconditions and effects, it has no direct effect, through the belief revision process (Section 3.4.7), on the belief state of the speaker and hearer. However, the dry-land component of the belief revision process may change the belief state, in explaining why the agent chose to say nothing.

- **tell-true** is used to simply give information that a proposition is valued as **true**. It is defined by the following plan rule:

```
name:             tell-true
parameter:        P
precondition:     bel(P)
effects:          {}
decomposition:    {}
```

Since tell-true has a precondition that the speaker believes the proposition, the belief revision process will update the belief at level three, five, and so on.

- **tell-false** is the complement to tell-true. It is defined by:

```
name:             tell-false
parameter:        P
```

```
precondition:     bel(not(P))
effects:          {}
decomposition:    {}
```

- **tell** can be decomposed to both of **tell-true** and **tell-false**, allowing an agent to plan a tell, without knowing at plan-time the belief state of the speaker of the tell.

```
name:             tell
parameter:        P
precondition:     {}
effects:          {}
decomposition:    { [tell-true(P)], [tell-false(P)] }
```

- **propose** is used by an agent to tell the intention that it has formed as a result of evaluating the game tree. That intention can be about just the next act or it can be tree representing the best play for the game tree. It is defined by:

```
name:             propose
parameter:        P
precondition:     {intend(P)}
effects:          {}
decomposition:    {}
```

Understanding a **propose** requires the hearer to apply the dry-land algorithm. The hearer uses the algorithm to search for the simplest belief state in which the speaker would choose the proposed alternative in the domain-level plan. This state must also be one in which the

speaker would choose to propose. There is a simpler pragmatic sense for propose, and that is for the hearer to merely prune the game tree at the choice node of the proposed alternative. While it is partly effective, it does not allow the hearer to take advantage by revising beliefs using the dry-land algorithm.

- **request** is like propose, in that it declares the agent's preference, but it obliges the hearer to adopt the requested intention. It is realised by an act **request-pair**, whose decomposition has two steps. In the first step, agent 1 makes the request. In the second step, agent 2 executes the requested act.

```
name:             request-pair(P)
parameter:        P
precondition:     {intend(P)}
effects:          {}
decomposition:    { [request(P), P ]}

name:             request(P)
parameter:        P
precondition:     {}
effects:          {}
decomposition:    {}
```

The hearer of a request revises his beliefs in a way that is similar to that for a propose. He searches for a state in which the speaker would both choose the requested alternative in the domain-level plan, and in which the speaker would choose a request in the negotiation plan.

- **ask**

  ask is defined as either a propose to tell, or a request to tell:

```
name:           ask-forced
parameter:      P
precondition:   {}
effects:        {}
decomposition:  { [request(tell(P))] }

name:           ask-autonomous
parameter:      P
precondition:   {}
effects:        {}
decomposition:  { [propose(tell(P))] }
```

Finally, some plan rules are required that construct a sequence of negotiation acts, and append the domain-level plan. These are:

```
name:           negotiation-plan
parameter:      {}
precondition:   {}
effects:        {}
decomposition:  { [domain-plan],
                  [negotiation-act, negotiation-plan] }

name:           negotiation-act
parameter:      {}
precondition:   {}
effects:        {}
```

```
decomposition:  { [request],
                  [propose],
                  [tell],
                  [pass] }
```

Lacking any empirical data, it was impossible to find suitable cost values for each of the acts. Instead, estimates were used, with each act valued at ten units, except pass, which due to its empty propositional content, was given a value of four. It was assumed, as it was in chapter 4 (see Sections 4.5.1 and 2.13), that these costs are additive over the dialogue.

```
utility(pass,-4).
utility(tell-true(_),-10).
utility(tell-false(_),-10).
utility(propose(_),-10).
utility(request(_),-10).
```

Since these values are estimated, they may not reflect the real performance of the planner very well. However, it was found that the results obtained in the examples of this chapter do not vary much in character given small variations to the estimates.

5.4 Realism

One of the desirable properties for the set of negotiation acts is realism. One way to demonstrate realism is to compare the repertoire with that of a standard agent communication language, such as FIPA [22]. FIPA has a communicative act library, whose repertoire of negotiation acts is similar to

the acts given here. Each is given a pragmatic specification using preconditions and effects. The tell act has the effect that the hearer believes the told proposition, and the effect of a propose is only that the hearer believes that the speaker intends the proposed act. These are quite similar to the schemas developed by Allen and Perrault [51] [1] and those of Appelt [5], which were discussed in Section 2.4.

A second way to demonstrate realism is to look to a corpus of human negotiation dialogue to see whether the acts that appear in the corpus are well represented. The TRAINS corpus [33] is a good candidate for this, where a human planner converses with a planning assistant in a meta-level plan to decide how to route some trains in a transportation network. The problem differs from that considered here in that only the user acts in the domain-level plan. However, the expert may still have beliefs and preferences with respect to that plan. Without making too much effort to develop annotation rules or to process large amounts of the corpus, a small sample of ten dialogues was taken so that an rough idea of the distribution of dialogue acts could be established.

The TRAINS dialogues are populated mainly with **ask** as the naive user consults the system, who is an expert about the domain state. Since the user is the decision-maker, the initiative tends to stay with him, and so it is rare that the system takes initiative in providing unasked-for **tell**s. **propose** makes up about half of his dialogue. **pass** occurs only occasionally. The system's acts were mainly **tell**s, as responses to **ask**s. **tell** was occasionally used as a response to a **propose**, but never used otherwise. **ask** was rarely used by the system, since the system is the domain expert. In the cases that it was used, it was for the purpose of clarifying the problem description. In all, there were no instances in these ten dialogues of acts that could not be

readily classified using the repertoire developed here, apart from greetings.

5.5 Demonstrations

In the following three sections, some example problems are used to demonstrate the planner, illustrating each of the negotiation acts. It is clear already that most of the desirable properties hold for the repertoire. They are "adequate" since the tell acts alone can communicate the total belief state of the agents, they are "efficient" and "simple" since they correspond directly with expressions of intention and belief. Section 5.4 has shown that they correspond with the acts users would be familiar with. The "necessity" property will become evident in the following sections as the acts are compared with one another in different demonstration problems to show that each dominates in some problem. These demonstration problems will also be used in the same way as the examples of chapter 4, to show how important it is that a probabilistic belief model is used in deciding negotiation strategies, rather than a logical one, and to show the utility gain that is obtained by using such a model.

5.5.1 Demonstration 1: Telling and proposing

In this demonstration, the desirable property of the necessity of the propose act is shown. When an agent proposes an alternative that is unexpected by the other agent, a lot of information is communicated. The cost of convincing the hearer that it prefers a proposed alternative using a combination of tell acts could be greater than that of the propose. The approximate relation between the likelihood of the proposed alternative and the equivalent number of tells is given by information theory [66]. The cost in terms of number of

tells is related to the information content of the propose, which is calculated from its probability by:

$$H = log(1/p) \tag{5.4}$$

For example, a propose that requires four beliefs to be held as preconditions, each of which has a probability of 0.5 in the hearer's mind, would correspond with a probability of one in two raised to the fourth power, or one in sixteen. This represents four bits of information, or four tell acts. On the other hand, an alternative whose probability of being chosen depends on the probability of just one belief, would at worst be as good as using a tell to communicate that belief.

As an example, agent 1 must choose whether to make fish or to make an omelette. If agent 2 has both sugar and fruit, it will save the eggs that would have been used for the omelette, and instead make fish allowing agent 2 to follow with a pavlova. This is quite an unlikely alternative though since it must have both sugar and fruit, whose belief values are both set at 0.5. The plan library for this problem is given in figure 5.4 and the corresponding game tree is given in figure 5.5.

The utility values for the problem were set at 50 for make-fish, 100 for make-omelette, and 100 for make-pavlova.

Negotiation acts were added to the agent's repertoire one by one to demonstrate the utility gain offered by each. To start, there were no negotiation acts, and so the game tree just consisted of the domain-level tree, with a value of 100 for make-omelette. Next, the pass and tell acts were added. This produced the negotiation game tree in the top part of figure 5.6. This tree shows the best play only for the agent, so that each choice node is pruned down to only one alternative. Notice that in response to a pass,

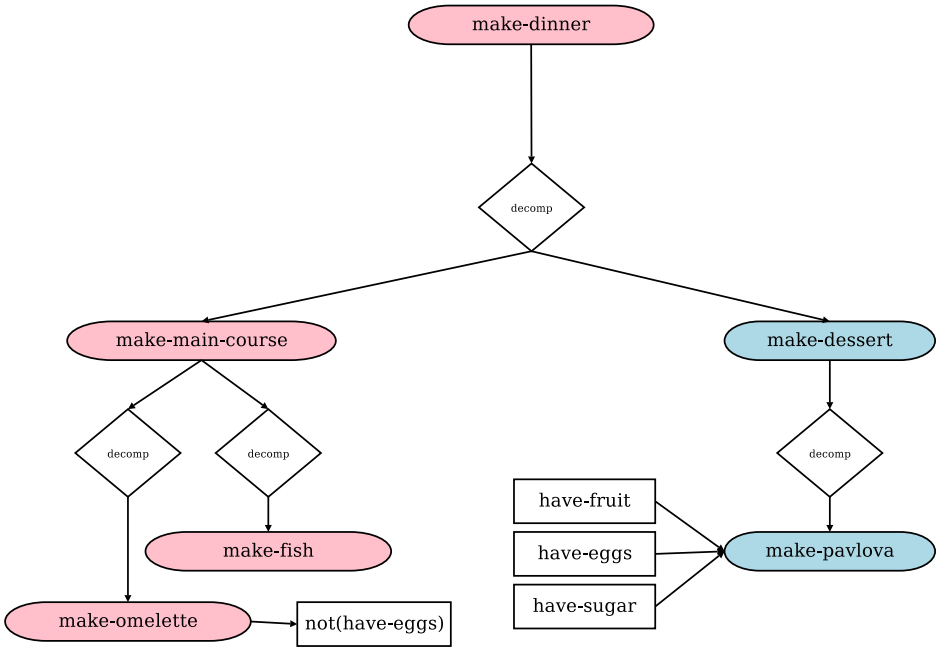

Figure 5.4: Plan library for propose example

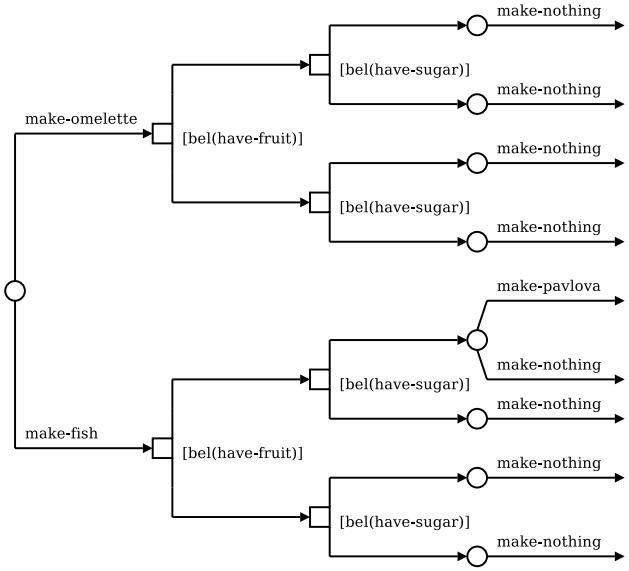

Figure 5.5: Game tree for the domain-level plan

agent 2 uses a pair of tells in the **true,true** branch of the game tree. This subdialogue is efficient, and since it happens in one quarter of instances, the

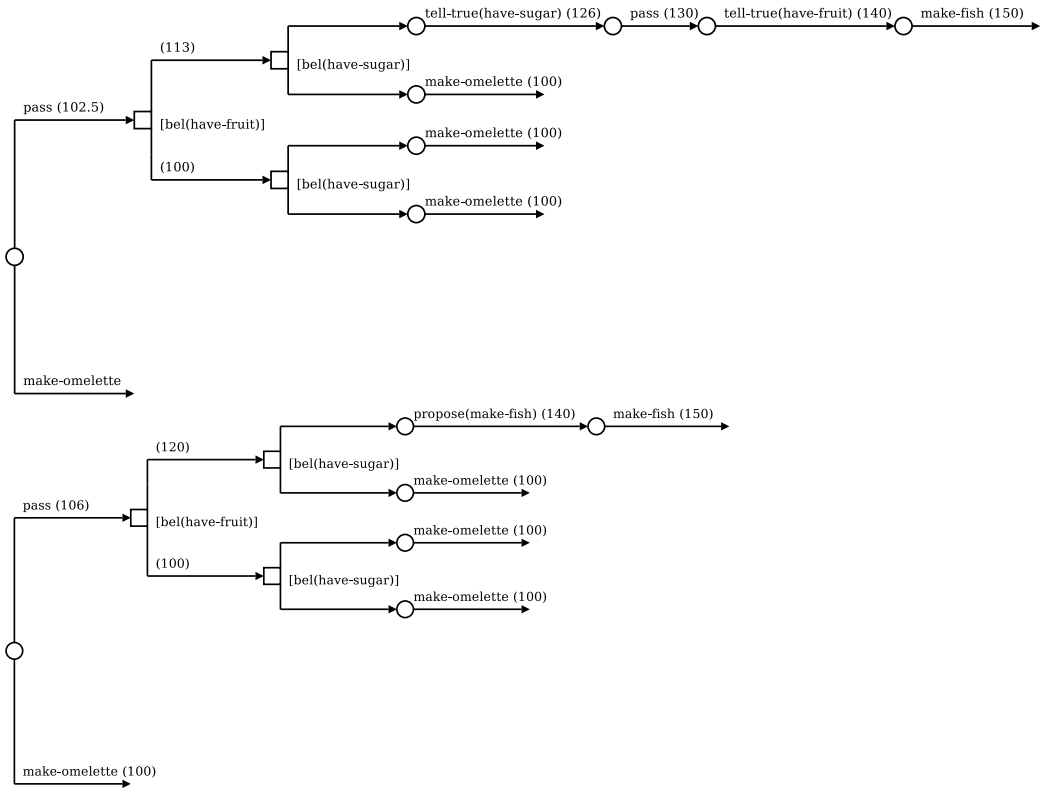

Figure 5.6: Game tree for the negotiation plan without proposing (top), and with proposing (bottom)

value for the tree is 102.5, which is a marginal gain over the 100 obtained from the plain domain-level plan. Next, the ask acts were added, but these were dominated by the pass and tell combination, and so the same result of 102.5 was obtained. Next, propose was added. This produced the tree in the bottom part of figure 5.6, with propose dominating instead of tell. Since the negotiation ends with the proposal, there is a smaller cost than in the top game tree in figure 5.6. The overall utility of the dialogue turns out to be 106, compared with 102.5 obtained without using propose. It can be concluded then that propose is a necessary member of the repertoire, since it is dominant in this example.

5.5.2 Demonstration 2: Holding the floor

In this demonstration, it is shown that the planner can effectively "hold the floor" in a negotiation. Holding the floor is when an agent decides whether to make a contribution to the dialogue, or whether to pass the turn to another agent, expecting that his contribution is more important. Such a decision cannot be made without considering the value of information. It is known that speakers use various cues to determine the passing of a turn [61], and that humans are very good at seamlessly coordinating their turns. Value of information might turn out to be useful in dialogue systems for deciding when to take the floor, and equally well for predicting when a conversational partner might want to take the floor, thus enabling coordination. Holding the floor is quite similar to the taking of initiative that was illustrated in both of the examples in chapter 4, where an agent must decide whether to contribute to the task, or to decline and allow the other agent to decide whether to contribute.

For the demonstration, an example problem was constructed in which agent 1 needs to know whether agent 2 has fruit, so that it can safely use up the eggs without spoiling agent 2's dessert plans. However, if agent 1 doesn't have any eggs, the question is not relevant. The expectation is that when agent 2 does not believe that agent 1 has eggs, then agent 1 needs to be forceful in asking, since agent 2, thinking the information is worthless, will not volunteer it without being asked. On the other hand, if agent 2 does believe that agent 1 has eggs, it will give the information without being asked. Therefore, agent 1 would not need to use an ask - it can just pass and wait for a tell, with less cost. Therefore agent 1 must make a decision between contributing to the negotiation with an ask, and passing to agent 2. The plan library for the problem is given in figure 5.7, while the corresponding

game tree is given in figure 5.8.

The utility function for the problem was defined as:

```
utility(make-omelette,150).
utility(make-fish,100).
utility(make-cake,100).
utility(make-fruit-salad,100).
```

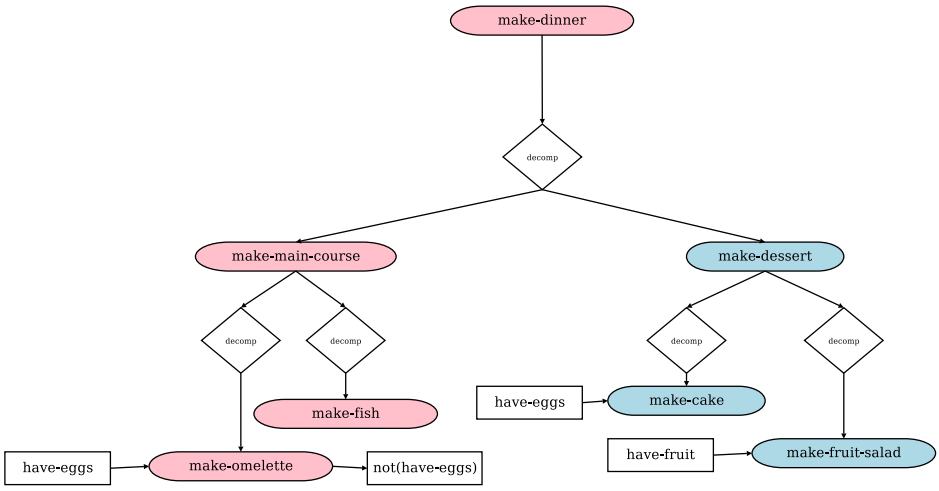

Figure 5.7: Plan library for "holding the floor" example

Results

An analysis was done for the problem where agent 2 decides whether to tell agent 1. The best utility gain that can be obtained by agent 2 is 50, obtained where agent 1 initially chooses make-omelette but is convinced to choose make-fish by agent 2. This gain is then multiplied by the value of the chance node for have-eggs, so that it ranges from 0 up to 50. The cost of telling is 10, so it would be expected that a decision surface between telling and ending the negotiation would occur where:

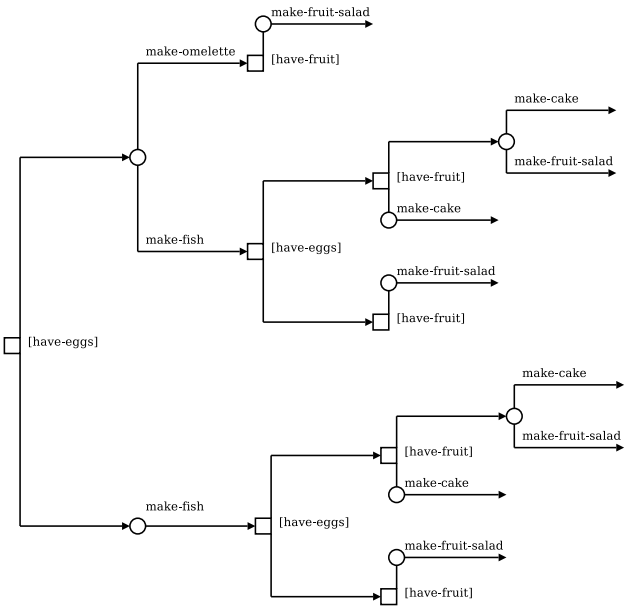

Figure 5.8: Game tree for the domain-level plan

$$\begin{aligned} 50p &= 10 \\ \Rightarrow p &= 0.2 \end{aligned} \tag{5.5}$$

To check this prediction, the efficiency of asking and passing to allow a tell were plotted for values of $p = 0.8$, and $p = 0.1$. At 0.1, agent 2 is expected not to use the tell act, and so agent 1 must seize the floor by asking. On the other hand, at 0.8, agent 2 is expected to use a tell act, and so agent 1 should decline the floor. These results are plotted in figure 5.10.

This demonstration provides a good illustration of the dry-land algorithm (see Section 3.4.7). The speaker uses the "propose" form of the ask act (for a definition, see Section 5.3). In response to the ask act, agent 2 must revise his beliefs so that the subject of the ask, tell, is efficient from agent 1's perspective. Therefore it searches the belief space from level 3 upwards, revising bel(have-eggs) at level 3 from 0.1 up to 0.94, and revising bel(have-

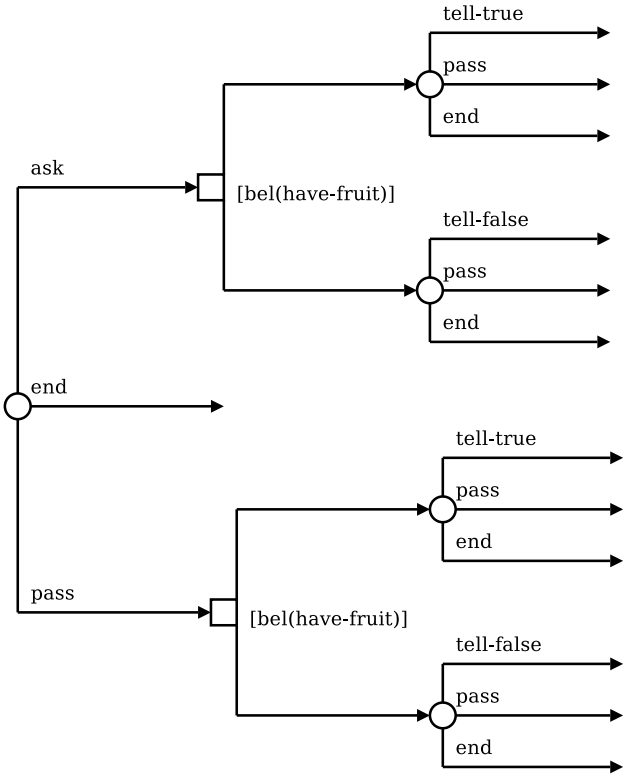

Figure 5.9: Game tree for the negotiation plan

fruit) a small amount from 0.6 to 0.56 at level 4. Since bel(have-eggs) is now so high at level 3, tell now becomes efficient for agent 2 as well.

Verification of results

In this section, the plots obtained in the last section are explained. First, the pass and tell alternative (see figure 5.9) is examined. Pass is followed by a chance node, with each branch of the chance node giving agent 2 a choice between ending the dialogue and telling. There is a third alternative, pass, but it is always dominated by end. In the **true** branch, agent 2 is expected to use tell-true only when the value for have-fruit is likely to be false at level

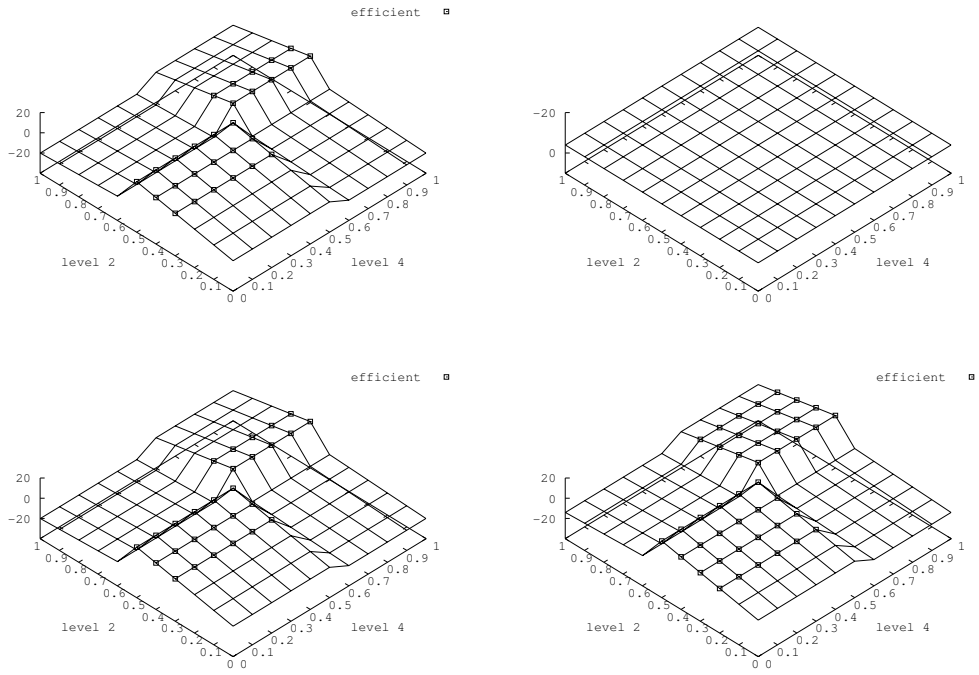

Figure 5.10: Utility of (left) **ask**, and (right) **pass**, at (top) $p = 0.1$, and (bottom) $p = 0.8$

4. Similarly, tell-false is used when this value is likely to be true.

Figure 5.11 plots the utility of pass and end, with respect to the have-fruit belief at levels 2 and 4, while figure 5.12 plots the difference in utility between them. A quadrant-by-quadrant analysis of this difference plot is now used to explain the results.

- **level 2 low, level 4 low** This is where agent 2 most likely prefers omelette, because the level 2 belief is low and agent 2 believes that agent 1 chooses omelette, because the level 4 belief is low. Since both beliefs are low, there is generally agreement between the agents, but as the value of the second level belief increases, there is a greater chance that agent 2 actually prefers fish, therefore it tells agent 1 and obtains a

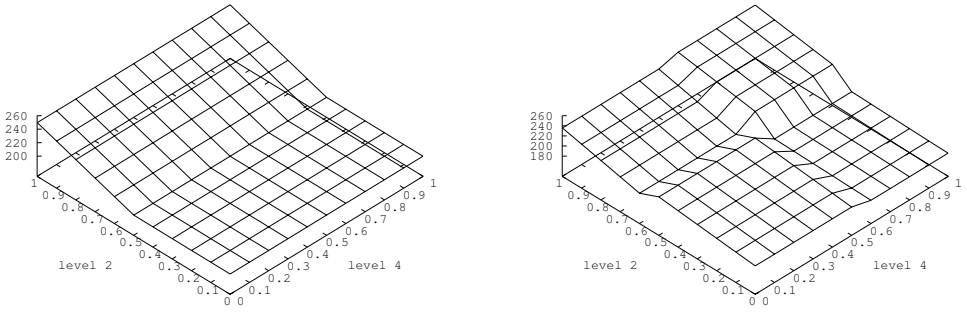

Figure 5.11: Utility of (a) end (b) pass

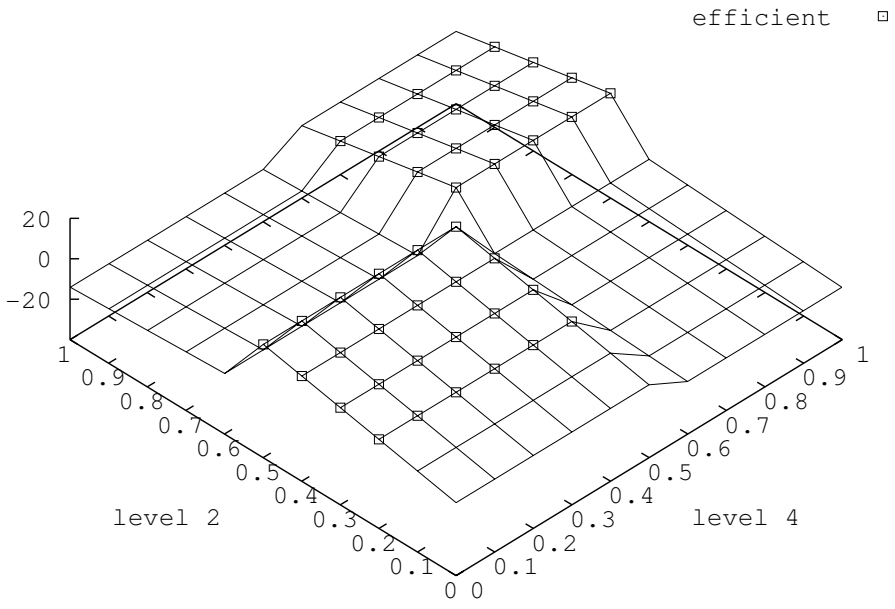

Figure 5.12: Difference image between pass and end

small gain in utility in figure 5.11 (b) over the corresponding quadrant in figure a. If this value increases over the 0.5 mark, there is a sudden change where agent 1 begins to choose fish without being told, and so the tell act is wasted.

- **level 2 high level 4 high** This is where agent 2 most likely prefers fish, and agent 2 believes that agent 1 chooses fish. Since both beliefs are high, there is generally agreement between the agents, but as the value of the second level belief decreases, there is a greater chance that agent 2 prefers omelette, therefore it tells agent 1 and obtains a gain in utility. Once the 0.5 line is crossed, agent 1 chooses omelette without being told and so the tell is wasted.

- **level 2 low level 4 high** This is where agent 2 most likely prefers omelette, but agent 2 believes that agent 1 chooses fish. agent 2 tells him so that it will change to omelette. However, since it generally chooses omelette whether it is told or not, there is a slight loss in this quadrant. The loss eases as the level 2 belief increases since agent 2 tends towards choosing fish.

- **level 2 high level 4 low**

  This is where agent 2 most likely prefers fish, and agent 2 believes that agent 1 chooses omelette. agent 2 tells him so that it will change to fish. Since it would have chosen fish anyway, there is a slight loss in this quadrant. This eases off slightly as the second level belief decreases.

An easier but incomplete way of verifying these results is to analyse the corner points. At (0,0) agent 2 should reason that since agent 1 already has the same beliefs about have-fruit, the dialogue should end, and make-omelette be chosen at 200. By the same token, at (1,1) the dialogue ends

with make-fish and make-fruit-salad chosen at 250. At (0,1), agent 2 needs to always correct agent 1's mistaken belief, resulting in make-omelette chosen at 200, with 10 taken away for the cost of always telling. Otherwise, make-fish would have been chosen, giving 150 and no fruit salad. Similarly at (1,0), the belief is corrected, resulting in make-fish and make-fruit-salad at 250 with 10 taken away, instead of a mistaken choice of make-omelette at 200.

Apart from the [pass, tell] strategy, agent 1 can use an [ask, tell] strategy (see figure 5.9, and 5.10). The difference between the two strategies is the cost incurred in using an ask instead of a pass, and a small belief revision change at level 4 to the have-fruit belief that is caused by the dry-land algorithm. This belief revision change makes do difference to the second agent's decision to tell. Therefore, the ask plot has a similar form to the pass plot.

This demonstration has shown how using initiative in holding the floor can depend on a probabilistic model of belief. Figure 5.10 shows how the decision depends on the probability value in the belief model. It is interesting as well to note from these figures the sensitivity of the decision of whether to continue with an ask or pass or whether to end the negotiation. Another note of interest is the deep nesting required to decide whether an ask is efficient. Notice in figure 5.10 that for the agent to decide between asking and ending the negotiation, the belief model must be examined to a depth of level 4.

5.5.3 Demonstration 3: Request and propose

Request and propose are quite similar acts, and it is not immediately clear that each is a necessary member of the repertoire. It seems odd also that request should be used at all with autonomous agents since it violates their free choice. However, it is still an expressive and useful act, and so if the agents can agree to always submit to requests, then it is of value. The

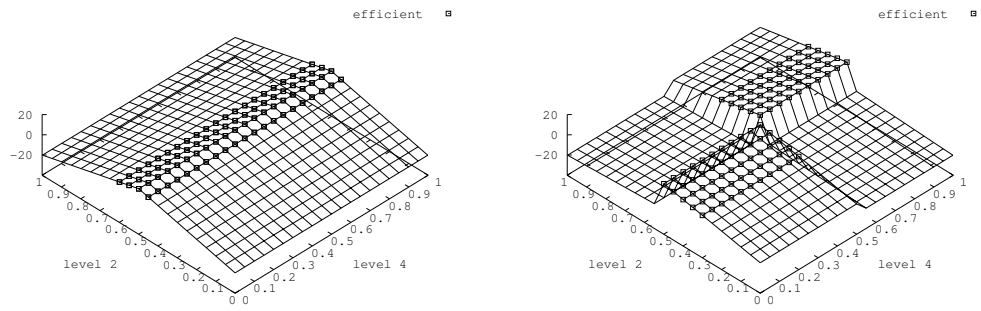

Figure 5.13: Difference image between (a) request(tell(P)) and end, and (b) propose(tell(P)) and end

question of handling requests in in a non-cooperative setting, where such agreements might be made insincerely, is an item for future work.

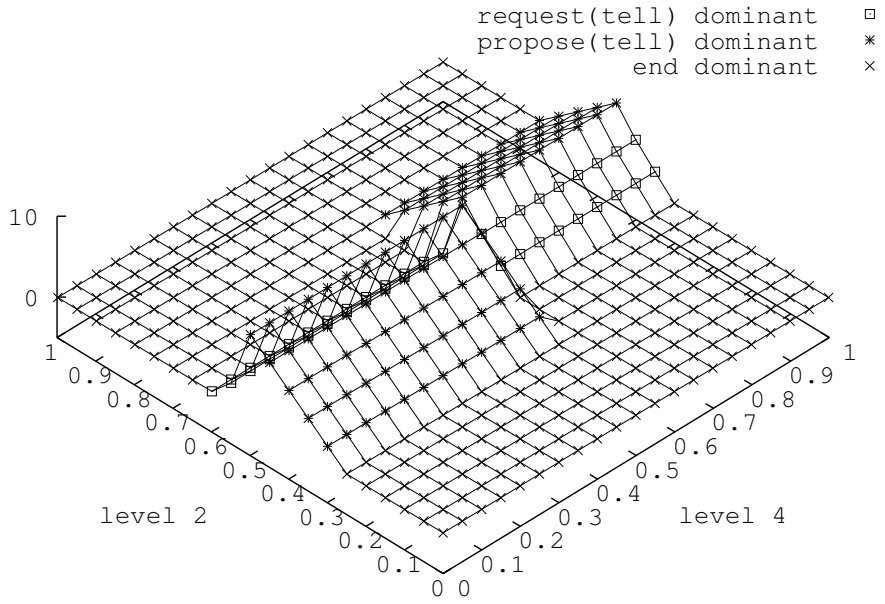

Figure 5.14: Dominance of propose(tell(P)) and request(tell(P))

The demonstration looks at asking, which is a composition of the request act with the tell act, or the propose act with the tell act. The fruit problem of Section 5.2.1 is used for the demonstration. A plot of the utility of each is given in figure 5.13. On the left, the request act is plotted. Since agent 2 unconditionally answers, the plot is derived from the basic value of information plot given in figure 5.3. The plot for propose(tell) is of the same form as that given in Section 5.5.2. Notice that neither request nor propose entirely covers the other over the belief space, and so each is dominant in some region of the belief space. Figure 5.14 plots the utility gain of the dominant strategy over ending the negotiation. Therefore each is a necessary member of the repertoire. Once again, it is apparent here that negotiation decisions are sensitive to the probability values of the probabilistic belief model.

5.6 Acts for non-cooperative dialogue

It was shown in Section 3.3.2 that in a cooperative setting, a speaker will rarely violate the mental state preconditions of an act, since it would lead the hearer to a state of false belief, and an irrelevant choice. In a non-cooperative dialogue, agents will certainly want to use acts like propose and tell insincerely. To allow this, some more rules are now added, to complement those of Section 5.3. These differ from the sincere forms in that they have no preconditions. However, in utterance planning, these acts would be decomposed to the same surface form, and so in plan recognition, a hearer would produce each act as an alternative hypothesis. The schemas are now:

```
name:            tell-true-sincere
parameter:       P
precondition:    bel(P)
```

```
effects:          {}
decomposition:    {surface-tell-true(P)}

name:             tell-false-sincere
parameter:        P
precondition:     bel(not(P))
effects:          {}
decomposition:    {surface-tell-false(P)}

name:             tell-true-insincere
parameter:        P
precondition:     {}
effects:          {}
decomposition:    {surface-tell-true(P)}

name:             tell-false-insincere
parameter:        P
precondition:     {}
effects:          {}
decomposition:    {surface-tell-false(P)}

name:             tell-sincere
parameter:        P
precondition:     {}
effects:          {}
decomposition:    { [tell-true(P)],
                    [tell-false(P)] }

name:             tell-insincere
parameter:        P
```

```
precondition:     {}
effects:          {}
decomposition:    { [tell-true-insincere(P)],
                    [tell-false-insincere(P)] }

name:             tell
parameter:        P
precondition:     {}
effects:          {}
decomposition:    { [tell-sincere(P)],
                    [tell-insincere(P)] }

name:             request-pair-sincere(P)
parameter:        P
precondition:     {intend(P)}
effects:          {}
decomposition:    { [request(P), P ]}

name:             request-pair-insincere(P)
parameter:        P
precondition:     {}
effects:          {}
decomposition:    { [request(P), P ]}

name:             request-pair(P)
parameter:        P
precondition:     {}
effects:          {}
decomposition:    { [request-pair-sincere(P)],
                    [request-pair-insincere(P)] }
```

```
name:             request(P)
parameter:        P
precondition:     {}
effects:          {}
decomposition:    {}

name:             propose-sincere
parameter:        P
precondition:     {intend(P)}
effects:          {}
decomposition:    {surface-propose(P)}

name:             propose-insincere
parameter:        P
precondition:     {}
effects:          {}
decomposition:    {surface-propose(P)}

name:             propose(P)
parameter:        P
precondition:     {}
effects:          {}
decomposition:    { [propose-sincere(P)],
                    [propose-insincere(P)] }

name:             ask-forced-sincere
parameter:        P
precondition:     {}
effects:          {}
decomposition:    { [request-pair-sincere(tell(P))] }
```

```
name:             ask-forced-insincere
parameter:        P
precondition:     {}
effects:          {}
decomposition:    { [request-pair-insincere(tell(P))] }

name:             ask-forced(P)
parameter:        P
precondition:     {}
effects:          {}
decomposition:    { [ask-forced-sincere(P)],
                    [ask-forced-insincere(P)] }

name:             ask-autonomous-sincere
parameter:        P
precondition:     {}
effects:          {}
decomposition:    { [propose-sincere(tell(P))] }

name:             ask-autonomous-insincere
parameter:        P
precondition:     {}
effects:          {}
decomposition:    { [propose-insincere(tell(P))] }

name:             ask-autonomous(P)
parameter:        P
precondition:     {}
effects:          {}
```

```
decomposition:    { [ask-autonomous-sincere(P)],
                    [ask-autonomous-insincere(P)] }
```

5.7 Design and implementation of the negotiation planner

Two paths could have been taken in the design and implementation of the negotiation planner. One was to maximise reuse and code the negotiation acts as STRIPS rules within the input to the domain-level planner. The other path was to maximise efficiency and hard-code the planner. The first approach seemed quite appealing, especially as pass, tell, request and propose have all been defined using STRIPS operators. At first glance, it would seem that merely providing these plan rules would be enough to equip the domain level planner described in chapter 3 for planning negotiation dialogues. However, the parameters of the STRIPS operators need to be instantiated from the belief set and from the game tree, and so something more than plain plan rules is required. As well as this, direct coding of the negotiation planner rather than use of interpreted rules would result in faster construction of the game tree since the planning and plan recognition processes are bypassed. This path was taken for the current implementation.

The negotiation planner is quite simple, since it does little more than prepend a game tree to the domain level plan. There is a set of act generators, each of which supplies a set of alternatives at each choice node. They are:

- **pass** this returns a pass act
- **ask** this returns ask acts for all available indefinite beliefs in the speakers belief set

- **tell-true** this returns tell acts for all of the speaker's positive beliefs that are indefinite to the hearer. The speaker's belief model is consulted to check whether they are indefinite

- **tell-false** this returns tell acts for all of the speaker's negative beliefs that are indefinite to the hearer. The speaker's belief model is consulted to check whether they are indefinite.

- **propose** this returns all of the best strategies from all of the choice nodes in the game tree. A call is made to the domain-level planner to obtain the game tree. To identify the choice node whose best strategy is being proposed, the content of the propose may need to identify the path that leads to the choice node. For example one rendering of a propose might be "If you decide to go shopping, then I will mind the children". To keep things simple for the moment, the current implementation only allows proposals to refer to the choice made at the root of the domain-level game tree.

The game tree is constructed recursively. There is a function that calls each of the generators. Each generator returns a set of edges corresponding with acts, and the function gathers these to form a choice node. Each generator recursively calls the function to obtain the remainder of the game tree. At each choice node, a special act is used to end the negotiation and begin the domain-level plan. This represents the possibility that either agent could choose to end the negotiation at any time and begin execution of the domain-level plan.

The belief revision module is called as the game tree is constructed. This prevents the planner from doing things like asking the same question many times.

One important issue with the negotiation planner, more so than with the problems given in chapter 4 is the branching factor of the game tree. The tell generator produces as many alternatives as there are beliefs in the belief set, and the propose generator produces as many alternatives as there are choice nodes in the domain-level game tree. For even small problems, dozens of alternatives could be produced. The recombination planner would not be of much use here since, for example, a tell act on a proposition can occur as an alternative at each step of the negotiation (see the discussion of the conditions for recombination to be of use, Section 3.4.9). A form of heuristic search, (Section 3.4.9) would be much more useful. For the heuristic, the utility of the domain-level plan can be used by calling the evaluation function of the domain-level planner, in the context of the revised belief model, and subtracting the cost of the negotiation acts in the path. A beam search would be a good way to perform the heuristic search since the complexity of the search would then be guaranteed to be linear with the plan depth.

The negotiation planner provides much the same game tree that the use of the STRIPS rules would have, but without the flexibility of being able to add new rules. In particular, generators for non-cooperative dialogues have not so far been specified. The request act is also unavailable, but both pragmatic forms of ask are available.

A description of the implementation of the negotiation planner is given in the appendix.

5.8 Related work

Gmytrasiewicz and Durfee [25] have applied their Recursive Modelling Method to the computation of value of information of negotiation acts (see Section

2.10). The RMM uses a tree of game matrices, with a belief value at each node of the tree. Value of information computations are used to find the difference in expected utility between the total tree, which the agent must use if it does not have the benefit of information, and the weighted sum of utility over the pair of child subtrees that are obtained when the informed belief is placed at the root of the RMM tree. The communicative acts are as follows. Modelling acts are used to communicate beliefs, and so correspond with the **tell** act, with an equivalent pragmatic definition. Intentional acts are used to communicate preferences and so correspond with the **propose** act. Their pragmatic definition is that the hearer prunes all but the preferred alternative from the matrix, which is a less effective but simpler approach to the dry-land algorithm described here. Instead, the dry-land algorithm seeks an explanation for the preferred alternative. Once an explanation is found, the choice node is effectively pruned, but as well as that, the hearer has gained some information about the beliefs of the proposer. Question acts are used to declare an agent's ignorance of alternatives, allowing an autonomous response by the hearer using a modelling act. Again this is a simpler approach to the dry-land interpretation of questions that is used here. Requests are not included as communicative acts, even though they have been shown here to be necessary. Additional acts that have no correlates here are imperative acts, which are used to declare knowledge of which of a number of uncertain alternatives is the true one, and acknowledgement acts, which are used because messages are sometimes not heard over noisy communication channels.

5.9 Conclusion

This chapter has investigated the planning of negotiation dialogues, which are meta-level dialogues in which agents exchange information about their beliefs before choosing a domain-level plan. It was argued that such negotiation must take the probability of belief states into account, since there are may be many candidate plans for negotiation that are possible, but few that have a reasonable chance of being chosen, and therefore few that are worth the effort of negotiation. On the other hand, meta-level planning using a logical belief model considers all plan candidates equally, and may fail to be useful because of this. A set of negotiation acts was chosen with a number of desirable properties in mind - that they correspond with the acts seen in human dialogue, that they are simple, efficient, and fully expressive without any redundancy of expression. The chosen repertoire of acts - "pass", "tell", "propose" and "request" was demonstrated to have these properties. It was possible to formally specify these acts by writing STRIPS rules for them, but in implementing the negotiation planner, a recursive function was written to generate a negotiation game tree rather than directly using the STRIPS rules. This game tree is then evaluated using the evaluation module of the domain-level planner.

The planner has been shown to be able to decide whether a negotiation act is efficient, but so far the examples given have been small. There may be some issues of coherence of long negotiations where the negotiators should move the focus point [28] in a regular manner over the domain-level plan. For long negotiations, the game trees grow quickly with the number of alternative negotiation acts and with the length of the negotiation. It has been suggested that a heuristic search be used to cope with the rapid expansion of the game tree, although no results about the efficacy of this approach are available yet.

An example of a larger problem might be one in which a kitchen assistant robot must schedule a large number of meals, in cooperation with a human chef. Such a problem would have a suitably large belief set that an extended negotiation would occur. There might be many differing beliefs about the set of tasks to be accomplished, availability of ingredients, and availability and state of the cooking implements. This would be especially interesting in an environment that is not mutually and fully observable, so that information becomes available more through dialogue than through observation of actions taking place in the environment.

Chapter 6

Future Work

6.1 Introduction

In this chapter, some open questions arising from the work presented in this thesis are discussed. These are questions that are worth exploring in the future, but are not thus far adequately addressed. Instead, sketches will be given of ways in which they may be addressed will be provided that may form the basis of future work. Two directions of importance are related to the evaluation of the planner. A comparison should be made with reinforcement learning techniques in order that measurements are made of the efficiency gain of the planner over a reinforcement learning system, given the same amount of training material. The planner must also be shown to efficient in a human setting, where bounded rationality threatens the expectations that the planner has about the outcome of the dialogue, in assuming an ideal counterpart.

Section 6.2 describes experiments in comparing the planner with a planner based on reinforcement learning. It also discusses the possible limitations of the planner with respect to dialogues with human beings, the shortcomings of

the simulation approach to evaluation, and how the planner might be further developed once experimentation with human subjects is used. Section 6.3 discusses ways in which the planner's design can be developed to encompass a more general planning model, or to improve its efficiency. Section 6.4 looks at some example dialogues problems that were not covered in the thesis, but which could be tackled by the planner. Section 6.5 looks at issues in negotiation dialogues. Section 6.6 shows how to integrate the planner with a statistical speech recognition system.

6.2 Evaluation

In this section, a comparison is made between planned dialogue and reinforcement learning, in Section 6.2.1. Subsequently, evaluation in a human setting, rather than in simulation, is discussed in Section 6.2.2.

6.2.1 Comparing reinforcement learning with planning

In Section 2.9.1, reinforcement learning was discussed as an alternative to planning of dialogues. Both approaches choose a dialogue strategy by training on dialogue data, and as a consequence it is important to compare them. It might be argued that reinforcement learning eliminates much of the complexity involved in using plan rules, by defining the dialogue form using states and state transitions rather than a mental state and plan rules. Learning a policy is then straightforward, and seemingly is the best that can be done with the available dialogue evidence. However, it has been shown in Chapter 4 that example dialogue problems can be specified quite easily with the planner. To compete with reinforcement learning, it is only required that the planner can produce better dialogues, given the same amount of training

material.

Reinforcement learning ignores much of the information that should be shared between states. To take an example, consider the flight booking problem given in Chapter 4, and the game tree for the problem given in figure 4.15. Notice that the precondition intend(book-flight-window) appears at many different choice nodes in the game tree. This means that in many different states, the agent would have the opportunity to gather evidence for that precondition. On the other hand an MDP system would not allow such sharing of information between states. It would only train on dialogues whose path includes that state. Effectively, the planner is taking the training data from three different states where the MDP system is only using one. It is not hard to think of other examples. For instance two robots assembling a car would quickly learn the applicable and inapplicable plans of each other by inferring preconditions. For example, if robot 1 sees robot 2 use tool a for task x, and having tool a is a precondition to task y, robot 1 has already got some evidence about the applicability of task y, without having experienced an instance of task y. Reinforcement learning on the other hand would only learn about task y from direct evidence of the execution of task y. For problems like this in which agents must try plans they have never seen tried before, planning rather than reinforcement learning is appropriate.

The question of which approach is the better one will often depend on the amount of training data available. If training data is inexpensive, or if there is a small number of states, an MDP may be the better choice due to its simplicity. On the other hand, if training data is expensive, the planner may well be the better candidate. Often, training material is obtained by running the system with real users, and so the training expense does come into play.

It is proposed that an experiment be carried out to compare the two approaches, using a suitable practical example, perhaps the flight booking problem seen in example 1 of Chapter 4. This could be carried out as a simulation experiment, using randomly generated belief states input to the planner to produce artificial dialogues. These dialogues would provide training material for a planner based on reinforcement learning and the ordinary planner. Each would be evaluated in a simulated dialogue against the ordinary planner. The experiment might show that the performance of the reinforcement learning approach is worse than that of the planner. Better still than this, training and testing could be done with real users, especially as the planner might perform particularly well if the dialogue partner is another instance of the planner. It would be important to choose a good distribution of problems, since each style of planning is suited to a particular kind of problem.

6.2.2 Dialogues with human beings

The planner has been developed and evaluated in a simulated environment, where the strategies of the dialogue partner are assumed to be those that the planner would choose. More likely than not, human factors will have to be considered that would require some change of the planning model. It is likely that human dialogue partners respond to their limitations in working memory and inferential capacity by using dialogue policies of the type described in Section 2.9.1 more often than by generating a game tree. If a game tree is used at all, it will likely be a severely pruned one, with much more rough calculations of the probability and the utility of outcomes, and with little use of a deeply nested belief model. Whichever planning model they use, it is likely that they will form plans that are outside of the domain

of specialisation of the planner's set of plan rules. For example the planner does not consider hypotheses that are not strictly focussed, yet humans can work around unexpected focus shifts if they have to. At the moment, there are no mechanisms to respond to such plan parsing failures.

Although the planner has been developed using established theories of dialogue planning, there is as yet no direct evidence that the planner is efficient outside the simulation. There are therefore two objectives for future work. First, there is a need to find out how well the current planner performs compared with its performance in the simulation. Second, and especially if the planner performs poorly, human rationality in decision making [20] and in dialogue planning needs to be explored (for example [75], [7]). It might be possible to use parameters like "depth of game tree" or "use of nested belief model", and learn the parameter values using training dialogues with users.

6.3 Improvements to design features of the planner

6.3.1 Evaluation of game trees using a logical model of belief

While the planner has been developed using the maximum expected utility rule for evaluation of game trees, an alternative evaluation rule is to check whether particular dialogue outcomes are possible, and to choose the current act based on those. The probability of the outcomes is of no concern. This was discussed in Section 3.4.5, but was not implemented. Such evaluation would be an implementation of Pollack's [53] claim that agents plan dialogues

using different sets of beliefs about plan rules and the domain state, but would retain the traditionally preferred logical model of nested beliefs. Such planning would be interesting where it is very important that there is no possibility of plan failure. Implementation of this idea is straightforward - the continuous utility function should be replaced with a two valued one of success and failure, and the maximum expected utility rule used at chance nodes should be replaced by one that selects the worst outcome of the two alternatives, given the evaluating agent's beliefs. A less elegant but effective way to achieve the same effect with the current planner is to use extremely large negative utility values to represent failures. These outcomes would be rejected even if the probability of the outcome is small, if the negative utility is large enough.

6.3.2 Beliefs about plan rules

In the current implementation of the planner, there is only one game tree, which is constructed using beliefs about plan rules from only level 1 and level 2 of the belief set. This single tree is then evaluated using a minimax-like algorithm. In future, there should be a different tree from the perspective of each agent, because each has different beliefs about the applicable plan rules. The agent at level 1 would construct a tree using the level 1 and level 2 beliefs, whereas it would expect the agent at level 2 to be using levels 2 and 3, and so on. By a series of recursive calls the evaluator can call the planner to construct each game tree in context, evaluate it, and use the resulting choice of the agent to form the best play.

For many dialogue problems, agents do not change their beliefs about plan rules, and so for the purposes of this thesis, the current implementation is adequate. Negotiation rules would never change. However, in a negotiation

over a domain plan, it may happen that one of the agents, a novice, has little expertise, whereas the other, an expert, has lots, and the point of the dialogue would be for the expert to convey the rules to the novice as efficiently as possible. The planner could be used for example, to plan out a **tell** act where the expert tells the novice how to make a pavlova, with the domain plan appended at the end of the instructional dialogue, in much the same way as the negotiation dialogues were planned in Chapter 5. There would be a chance node in the domain plan to represent uncertainty about whether the novice knows the plan rule. The value of information of the instructional dialogue could then be calculated. The expert might also find out what the novice does and does not know by observing his domain plans and using belief revision on the plan rule beliefs, thereby triggering hints as it executes the plan. In another scenario, a negotiation may involve experts from different domains, who in their negotiation must explain why their proposals are reasonable. In both of these scenarios, the ability to generate different game trees from different perspectives is essential.

6.3.3 Improved belief revision

The current design of the belief revision system is based on lazy revision, that is, revising only the least controversial beliefs. For example, if agent 1 and agent 2 were to disagree about the proposition "blue(sky)", and agent 2 executed the act tell(blue(sky)), agent 1 would come to believe that agent 2 believes that blue(sky), revising at level 2, but would not address his conflicting belief at level 1. Level 1 beliefs are always absolute (true or false), in contrast to beliefs at the other levels, which are continuous estimates of level 1 beliefs. However, these estimates cannot currently be communicated by the planner - it only deals with level 1, absolute beliefs. Supposing though

that they could be communicated, it would be straightforward to conflate each agent's estimate by summing the frequency counts of each. For example agent 1 might estimate that agent 3 believes P at 0.6 since on 3 of 5 occasions, an act was observed as evidence of P. On the other hand, agent 2 might estimate 3 of 15. The summed count would then be $3 + 3 = 6$ of $5 + 15 = 20$ giving a probability of 0.3. There is no similar way of conflating level 1 beliefs in the current system.

One good example of where conflicting level 1 beliefs need to be resolved is that of misconception correction. An example given by Pollack is one of a caller to a hospital who asks for Kathy's number. A precondition to asking is that the caller believes that Kathy is still in hospital. The receptionist can choose whether to give Kathy's home number, give Kathy's home number and correct the misconception, or to give the hospital number as appropriate (see figure 6.1). Correcting the misconception has some dialogue cost, but this is made up for since the caller can then visit Kathy at home. This problem was input to the planner. It is hard to say whether the receptionist should revise his beliefs about Kathy being discharged when the caller asks, or whether the caller should revise his beliefs when the receptionist tells him that Kathy has been discharged. The former case leads to the planner choosing the play shown in figure 6.2. Unfortunately the receptionist chooses not to correct the misconception and the plan eventually fails. By using the strategy of the caller revising his beliefs, the best play chosen by the planner is given in figure 6.3. This strategy results in the desired behaviour, but it is hard to make a domain-independent decision about which of the agents is right. Notice that the planner as it is currently implemented produces the play in figure 6.4, with the receptionist giving the correct number but incorrectly omitting the misconception correction since the caller will not as a result

revise his level 1 beliefs.

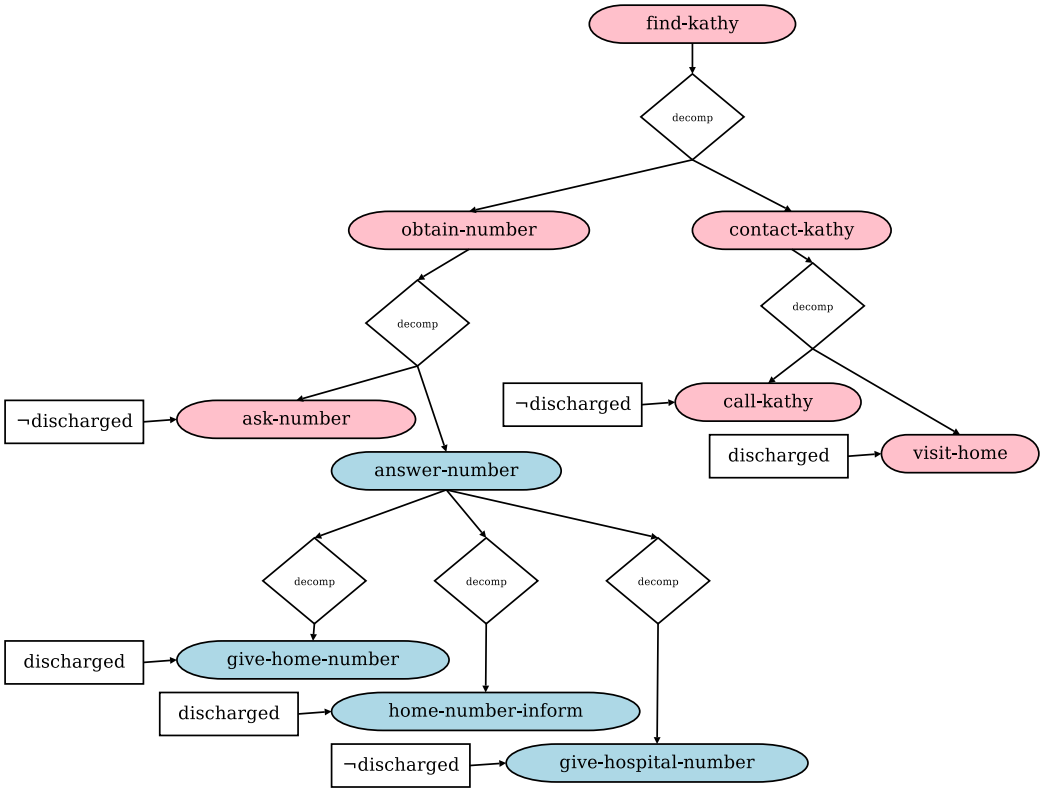

Figure 6.1: Plan Library for misconception problem

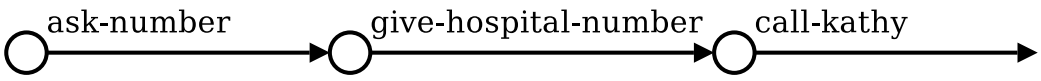

Figure 6.2: Best play for misconception problem with receptionist revision strategy

To properly deal with problems such as the misconception problem above, agents must reason about the amount of evidence upon which their beliefs are founded, and the inferences that can be drawn from conditional dependencies between beliefs. For instance, the fact that the receptionist is at the hospital

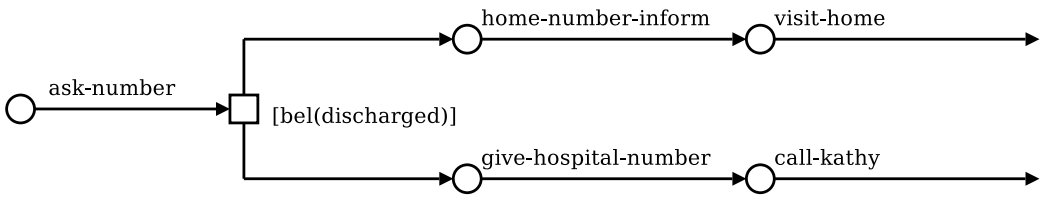

Figure 6.3: Best play for misconception problem with caller revision strategy

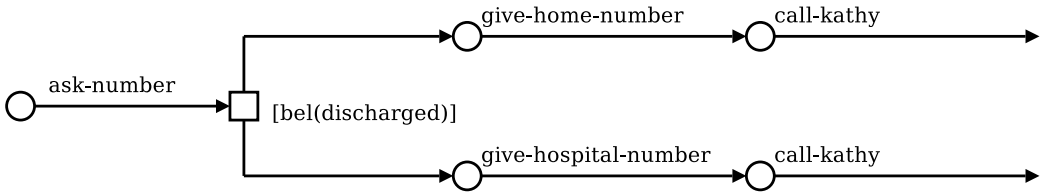

Figure 6.4: Best play for misconception problem with lazy revision strategy

should be supporting evidence for his belief, that can be used to conflate it with the caller's previous belief. They might also conduct a dialogue of providing supporting beliefs in a structured argument, which would update the evidence beliefs in the hearer's belief model, thereby reinforcing the conclusion of the argument. For instance, the receptionist might mention that he is in the hospital to support the conclusion. Value of information judgements could be used in the selection of the argument. This would require a belief model that allows for non-independent beliefs, such as a Bayesian network. There is already a system by Horvitz and Paek [35] that makes similar value of information judgements about collecting evidence to support a hypotheses. Coincidentally, it also involves a receptionist problem, where clarifications are planned to disambiguate a user's intention.

It was argued in Section 3.4.7 that for the examples in this thesis, it is convenient that beliefs should be independent. However, there is a style of dialogue that is not focussed on the immediate domain plan, but rather

provides value of information over the long-term by improving the agent's general knowledge. In such dialogues, the ability to draw inferences from acquired beliefs is a lot more important. For example, an agent may learn that there is sugar in the cupboard, which supports his immediate plan of making a pavlova, but the inference that the week's shopping has been done and that there is probably flour as well provides value of information on many occasions in the distant future. Because of this, and because of the need to look at evidence and at argumentation, future work might focus more on the conditional dependencies between beliefs, and use a Bayesian belief network [50] to represent these.

The planner is not yet capable of revising the parent intention rules (see Section 3.4.7), since such revision is quite complicated. It is hoped that in the future that the some form of revision can be implemented. The approach that is used at the moment is to take the first plan tree that fits an act sequence, and revise the rules based on this. This only works for examples that have only one parse. For example, the window-seat problem given in Section 4.6 has a unique parse.

6.3.4 Multilogue planning

In Section 3.4.10 an outline was given to extend the planner to deal with dialogues between three or more speakers. The design changes were minimal, requiring little more than the use of a tree-shaped, rather than a linear nested belief model, and the assumption of round-robin turn-taking. Multilogue planning would be particularly suitable for negotiations in cooperative planning problems. For example, a task allocation problem might require bids from a number of agents, who as part of their bid, inform the task owner of beliefs about their capabilities and resource states. For example,

the head chef might ask all the other cooks, "Does anyone know how to peel potatoes?", before deciding who will make the main course and who will make the dessert.

6.3.5 Dealing with complexity

Through the course of the experiments conducted for this thesis, the complexity of the problems grew, and it became obvious both from a computational complexity analysis of the planning problem and from practical experience that it was easy to construct a reasonable planning problem that would overwhelm the hardware used for the experiment. In particular, it grew difficult to collect plentiful data for some experiments reported in Chapter 5. It is also necessary to control the complexity of the tree so that value of information decisions can be made for long-range plans that are deep rather than wide. In the design chapter (see Section 3.4.9), a number of approaches to the complexity problem were discussed. One of these, "probability mass search" was implemented for controlling the complexity of chance nodes, but was not used. A heuristic beam search was proposed for dealing with choice node complexity. However, it may not be useful when the agents have opposing utility functions, since one agent may unwittingly prune an alternative that the other agent subsequently chooses. On the other hand, there are many applications in which the system and the user share a utility function. In these, a beam search would be most appropriate. The relationship between dialogue efficiency and beam width has yet to be explored. Another promising technique is that of recombination, whose approach was to recombine alternative subdialogues in the game tree, one the subdialogue had closed. This technique is important since it reduces the complexity of the problem to linear from exponential. However, it is not always applicable. While

the game tree can take a compact recombined form, the evaluation algorithm complexity is not isomorphic to the game tree shape. This is because where beliefs are revised differently in parallel branches, recombination of belief models in evaluation can only happen if the value of the belief is never needed again in the evaluation. More often than not this will not be the case, and especially in the kind of long-range planning where complexity measures are particularly suitable. The best approach to limiting complexity remains to be seen.

6.4 Further example problems

In this Section, some types of dialogue problems that could be solved in the future by the planner are described.

6.4.1 Long-range planning and user-model acquisition

Long-range planning is the planning of immediate dialogue which provides value of information that is realised not immediately but in the longer term. For example, in the risk problem of Section 4.5, it was shown that over successive runs of the same dialogue, the performance of the system improves with each dialogue adding evidence that provides a gain in all of the following dialogues. This long-range benefit can make the difference between choosing a dialogue in which such evidence is given and choosing some alternative dialogue. These kinds of dialogues seem to happen a lot in everyday life - people with little to do immediately often fill the time by playing games in their environment, or chatting about generally interesting things, but these lead to accumulated wisdom which will have a long-term benefit. Schoolchildren participate in a lot of this sort of dialogue. It would be interesting to

exercise the planner to generate dialogues that would be of such long-term benefit, and particularly apply it to the planning of tutorial style dialogue.

To perform long-range planning, the planner would need to generate the most probable of the agents' future courses of action, many steps into the future, so that the value of the current dialogue act could be estimated. Probability mass pruning for chance nodes and heuristic search for choice nodes (see Section 3.4.9) allow the most probable and viable outcomes to be explored, so that the planner can easily generate a game tree that has a very low branching factor but is instead very deep so that distant outcomes are properly covered. As an example, in a cooking domain, a deep game tree of 1000 nodes but with a breadth of only 10 would cover most of a cook's important activities over the next few days, so that a cookery lesson can be planned, and evaluated in the context of that domain plan. Suitable examples are yet to be constructed to demonstrate how efficient long-range dialogues can be constructed.

The long-range planner could also be applied to a problem in user modelling - that of explicit user model acquisition [38]. This is where explicit questions that are not part of the user's immediate plan are asked of the user so that the system can perform an initial classification of the user, such as whether they are experts or novices in the domain. This contrasts with implicit user modelling where the system passively observes the user. The planner could generate a long-range plan for the user at the start of a session. By attaching this game tree to the leaves of the different acquisition question subtrees, the value of information of the acquisition questions can be found. In an analogous fashion, physical acts for learning about an environment could be planned. For example, one might envisage a walking robot who must explore an environment by trying different routes, gathering

information about obstacles, so that future route planning problems can be more readily solved.

6.4.2 Dry-land algorithm

In Section 4.7, it was shown that the dry-land algorithm is important for belief revision when acts do not have preconditions. In the example, the travel agent chooses between offering a window seat and chatting, and the dry-land algorithm infers from its choice of chat that it either does not have a window seat or that it believes the user does not want one. The system's model of the user is thus revised. Here is another interesting example that would be a worthwhile demonstration of the algorithm. It is the problem of a job interviewee who chooses not to admit that he does not have a driver's license, until he is explicitly questioned. Instead he chooses to talk about something more impressive. In this context, the interviewer can use the dry-land algorithm to revise downwards the belief that he has a driver's license. He must do this since if he thought the interviewee did have one, then his rational choice would on the contrary have been to boast about it. The belief is revised to the decision surface between admitting and keeping quiet.

The planner is capable of handling this problem, but it has not been tested. Perhaps in the future the planner could be used to analyse job interview transcripts. A diagram of the plan library is given in figure 6.5.

6.4.3 Knowledge engineering and large problems

The problems discussed so far have had a relatively small number of plan rules. When a human partner is introduced, the planner will have to use large sets of plan rules that in turn generate larger game trees than have been seen so far. The problems of acquiring these rules and working with

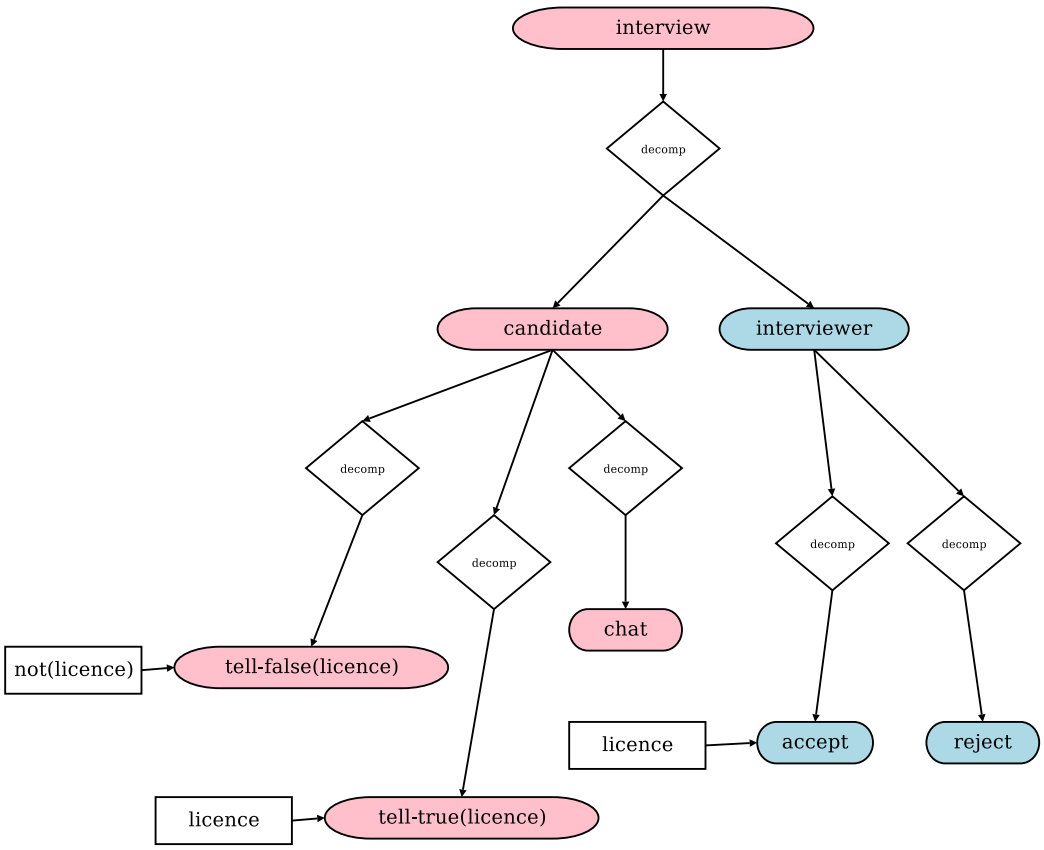

Figure 6.5: Game tree for the interview problem

complex planning problems have yet to be addressed. There is no procedure for inducing the rules from example dialogues. One approach is by expert hand-crafting of the rules, another is by automatic inference from dialogue examples. It is conventional that a "tell" act has a precondition that the speaker believes what he is telling, but it may be necessary to examine many examples to make this generalisation from a lot of "tell"s. As for learning the plan structures, there needs to be some method of choosing rule sets that can generate a set of observed example dialogues. There has been work in grammar induction, and particularly under the assumption that the language is generated by a context free grammar, that might be applicable to the task

of automatically finding the dialogue rules [4].

6.5 Negotiation dialogue

This section looks at future work in the negotiation dialogue area.

6.5.1 Request and propose

In Section 5.5.3, it was shown that in a cooperative setting, once an agent agrees to always, unconditionally execute the object of a request act, there are times when it can be more efficient to use than a propose act. However, suppose that agent 2 were to make such an agreement, and then, due to differences in belief, agent 1 requests an alternative that agent 2 does not prefer. Agent 2's rational choice is then to break the agreement. It would seem then that the rigid pragmatic form of the request act is impossible. The experiment has shown that the strong form of request can be superior to the weaker propose. However, there is probably a trade off between the utility offered by this strength and the utility offered by allowing the second agent to break the agreement. There might be some continuum of acts ranging in strength between propose and request, and the agents should plan an expectation that the agreement will be broken, especially in a self-interested setting. Further experiments might show that each act on the continuum is dominant in some region of the belief space. The contrast between the two acts also needs to be explored in the self-interesting setting, since self-interested agents are more likely to prefer autonomous forms of proposing.

6.5.2 Planning in self-interested settings

All of the examples in this thesis have used a shared utility function, and so have been examples of fully cooperative dialogues. However, when agents have individual utility functions, veracity is lost, and so agents can compute the value of misinformation as well as that of information. In performing plan recognition, hypotheses would be formed for each of the sincere and insincere forms of the speakers acts, so that the hearer can estimate whether a lie was the rational strategy for the agent. One example of this would be a "friend or foe" type of game where if both agents cooperate in a venture they can expect to do better than if both defect. However, if one defects and the other cooperates, the defector will be better off than had they both cooperated. For example a fraudster in a dialogue with a banking dialogue system might insincerely give his account number. It is then up to the dialogue system to decide between the risk of fraud, and the expense of a security measure such as the asking of the speaker's date of birth. This decision would rest on the system's expectation that the speaker is insincere. As another example, a negotiation of strategies between an alliance of two parties in a war, or in a competitive marketplace, might result in an insincere proposal to attack a third party, who is stronger than each, at a specific time. If the proposer defects, he can allow the other two to weaken one another to the point where he may enjoy a victory over both rather than a shared victory. The planning of communications in such a setting must accommodate both the planning and the recognition of insincere acts.

6.5.3 Focussing and plan recognition in negotiation dialogue

It is well known that focussed plan construction helps to reduce the number of plan recognition hypotheses to be considered by the hearer (see Section 2.5). However, the negotiation planner at present must entertain many different negotiation act hypotheses since they are not chosen by their relation to the subject of the previous negotiation act. It is clear from corpora of human dialogues for collaborative planning, for example the TRAINS corpus [33], that on the contrary, speakers move in a rigorously focussed way as they traverse the domain plan structure. For the sake of game trees with a manageable branching factor, and to improve the planner's performance in recognising noisy or elliptical input, a selection mechanism that penalises unfocused negotiation acts is proposed for future implementations of the planner.

6.6 Combining statistical methods in speech recognition with probabilistic planning

Up until now, it has been assumed that the planner deals with noiseless, unambiguous input, in the form of a dialogue act specification. As a result, belief revision has been straightforward. If the planner were to be used in conjunction with a statistical speech recogniser or semantic interpretation component, belief revision would need to deal with a probability distribution over act hypotheses, rather than just one.

Stolke et al [70] showed that information about dialogue context can make

a slight improvement in speech recognition performance. Their approach was to use dialogue act classification based on n-grams to predict the dialogue act class of the next utterance. Once a distribution over classes for the utterance is predicted, a language model is determined by mixing, in the proportions of the distribution, language models for each act type. It may be possible to apply a similar method by using expectations generated by the planner to obtain a model of the user's expected utterance.

Here is an outline of the mechanism that could be used. A typical speech recognition system consists of a hidden markov model (HMM) acoustic model combined with a language model for word sequences that determines the probability of the given signal given the dialogue act $P(U|A)$. The planner estimates the belief state, $P(B)$, and through the planning mechanism, the conditional dependence of the dialogue act on the belief state $P(A|B)$. Therefore, the conditional dependence of the utterance signal can be related to the belief state $P(U|B)$, if an model can be provided for producing the signal from the act. To perform belief revision, the conditional dependence of the belief state upon the utterance signal is required $P(B|U)$. This can be computed using Bayes rule as follows:

$$P(B|U) = P(U|B).P(B)/P(U) \tag{6.1}$$

The only unknown in this formula is P(U), but since it is invariant with the belief state, it cancels out when the relative probabilities of belief states need to be found.

As an example, consider a belief state in which the hearer believes that the speaker intends to have the red ball at $p(red) = 0.75$, and the blue ball at $p(blue) = 0.25$, and that there are only two signals "red" and "blue". Suppose the system picks up the signal "red". $P(U|B)$ might be $p("red"|red) = 0.8$,

$p(\text{"}blue\text{"}|red) = 0.2$, $p(\text{"}blue\text{"}|blue) = 0.8$, $p(\text{"}red\text{"}|blue) = 0.2$. The revised belief would then be $p(red) = 0.96$ and $p(blue) = .04$. Notice that unlike before where beliefs were revised to 0 or 1, this revision reflects the error and ambiguity in the meaning of the signal.

In a noisy environment, the planner would be able to respond to such weak revisions by using clarification subdialogues based on value of information, with cut-off points for clarification appearing near 0 and 1. Choices could be made between more and less risky types of signal, for instance, "pass the red ball" would have a higher probability given intend(red), than just saying "red". The planner might go so far as to test different strategies by generating a speech signal using text-to-speech, and then feeding it straight into the speech recogniser as belief revision is performed at the next level in the game tree. This would be useful for predicting the amount of information the subdialogue is likely to provide, so that the value of information can be calculated.

In the spirit of the dry-land algorithm where a search is made for the belief state that maximises the probability of an observed act, the planner could be used with an acoustic model to search for the belief state that maximises the probability of the given signal. This approach throws away the remainder of the "n-best" list that is typically used in speech recognition, which could be useful on subsequent turns, but on the other hand it is a useful simplification that conforms with the current design of the planner.

6.7 Conclusion

This chapter has described how some open issues can be resolved in future work on the planner. These include comparison with other approaches

to planning, accounting for human rationality, improvement of the current design to match the planning theory and to enhance efficiency, further interesting dialogue problems, and integration with a speech recogniser in a spoken dialogue system.

Chapter 7

Conclusions

The main objective of the work presented in this thesis was to develop an agent-based dialogue manager that could use a nested belief model to plan efficient dialogues. To this end, a planning system has been implemented, and demonstrated using a number of examples. It is now in a suitable state to be reused for further research on dialogue planning, or to serve as a design model for agent-based dialogue managers. It is easy to program the system with an input file specifying plan rules and initial beliefs.

For the planner to generate correct dialogue and for it to generate a correct expectation of the user's dialogue contribution, it was required to adhere to current theories of dialogue planning. Using Carberry's model of plan recognition in dialogue [10], and Pollack's observation that agents apply different plan rules and state beliefs to the same plan [53], a model of plan formation was developed. It was argued that while this model of dialogue planning is useful in selecting correct plans, it does not say which of the valid plans is the most efficient. Therefore the planning theory was used to generate the alternatives in a Bayesian game [31]. This allowed different dialogue strategies to be evaluated, and for a user model, in the

form of a probabilistic nested belief model, to be used to find out which of the strategies had enabled preconditions.

One condition of acceptance of the planner is that the dialogues it produces are more efficient than those produced by current dialogue management systems. It was shown using examples and with simulated dialogues that the planner is more efficient by virtue of using a user model in the form of a probabilistic nested belief model. However, the evidence given here is only part of what is required. The simulation experiments assumed an ideal dialogue partner, and there is no direct empirical data to support the claim that the planner would be as effective against a human partner. However it is clear enough that the introduction of the user model could not result in a system that is any worse than current systems. Human trials are therefore the most important item for future work. As well as measuring the system, these trials may lead to a further cycle of development, where the planning model is changed to reflect the behaviour of the user. A secondary remaining objective in evaluating the system is that it is empirically compared with the reinforcement learning approach that is often used in learning dialogue strategies. However, it has been argued that in some situations, the planner would be superior to reinforcement learning.

A second condition of acceptance is that the planner can be used by a dialogue system designer without his full understanding of the complicated mechanisms used to decide strategies and to revise the belief model. It is apparent from the examples given in Chapters 4 and 5 that this has been achieved. The designer need only input a file describing the dialogue plan rules. An example of such a file appears in Section A.2.

The planner has not yet been integrated with the components that make up a complete dialogue system, namely a text planning and speech synthesis

system, and a speech recognition system. The speech recognition part is the most interesting since it must use a statistical model to translate from speech to a parameterised dialogue act suitable for use by the planner. This was discussed as an item for future work. At the moment, the planner is ready to be used only with descriptions of dialogue acts for input and output.

A set of domain-independent dialogue acts were developed which can be used to generate negotiation dialogues over a domain-level plan. These were shown to be useful in making value of information judgements, with application in collaborative planning problems. The negotiation acts were built into the planner's repertoire allowing automatic generation of negotiations.

To contribute to the research community, the implemented planner is intended to be released using the name "PED" (Planner for Efficient Dialogues). It is intended that it and its derivatives should remain free and open-source under the terms of the Lesser GNU public licence. This licence at the same time allows the system to be freely used as a "library", linked to non-GPL software, for example, to a non-GPL speech recogniser. A guide to this implementation can be found in the appendix.

Appendix A

Description and usage guide for the implemented planner

This chapter is intended as a guide to the implemented planner, that can help with repetition or examination of experiments and their recorded data, and with maintenance and reuse of the provided code in future work. Section A.1 describes the various parts of the system, and highlights the most important procedures. Section A.2 defines the syntax of the planner's input files. Section A.3 gives an example of an input file used with the planner. Finally, section A.4 gives an index so that the reader can find the files related to the experiments in the thesis.

A.1 Software architecture

The system was developed in Prolog using Swi Prolog, a free Prolog interpreter, and uses built-in predicates that may or may not be available to other interpreters. Therefore this interpreter is recommended for further work. The corresponding source code and experiment scripts are available

under the Lesser GNU Public Licence (LGPL) on the world wide web at http://planeffdia.sourceforge.net/. The various parts of the program are now described.

A.1.1 Domain-level planner

The **domain_planner** directory contains the source files that are used for domain-level planning. These are:

- **build_plan_tree.pl** contains the code that constructs the game tree. There is a procedure **build_plan_tree** that is called recursively to obtain the tree

- **incr_plan_rec.pl** The **incremental** procedure is used to apply plan rules to obtain a plan structure. It is called by **build_plan_tree** procedure to obtain a set of alternatives for a choice node in the game tree. It is a recursive procedure since it needs to incrementally add an act to the plan structure. Each act is added by the **find_choice** procedure. **incremental** checks for preconditions to acts, and passes any that are found back to **build_plan_tree**, so that a chance node can be inserted before the choice node.

- **utility.pl** The **make_choice** procedure is used to evaluate the game tree and return the tree's best play. This returned structure is a tree with chance nodes and choice nodes, where all but the best alternative have been pruned at the choice nodes. **make_choice** calls the procedure **e_u** which is used to evaluate a game tree. **e_u** in turn calls **make_choice** to find the best play to evaluate.

- **belief_revision.pl** The **belief_revision** procedure performs ordinary precondition and effect belief revision by updating the belief model.

Due to its relatively slow execution time, dry-land belief revision has not yet been integrated so that the ordinary domain level problems run quickly.

- **fuzzyise.pl** This file contains utilities for the sampling form of the planner, which was used in the experiments of section 4.5. It randomly chooses actual belief states according to the distribution estimated by the belief model.

- **dialogue_manager.pl** This is a rudimentary dialogue manager, that has a control loop to choose the system's act, perform dry-land and ordinary belief revision, update the history list, then read the user's act, perform dry-land and ordinary belief revision, and update the history list. It was only used once, in the experiment on belief model acquisition, where a user had a dialogue by reading canned English text for the system's act, and then performing his response by selecting from a multiple-choice list. This experiment was reported in section 4.7.

- **utils.pl** This file contains a set of general-purpose utility procedures.

A.1.2 Negotiation planner

The negotiation planner is contained within one file, **critical_points.pl**, contained in the **negotiation_planner** directory. The main procedure is called **neg_tree** and it recursively generates the negotiation game tree. At each choice node, a set of procedures is called to provide the set of alternative negotiation acts for that point in the dialogue. Once the game tree has been constructed, the domain-level game tree is attached to the leaves. Evaluation is performed using the code from the domain-level planner, but the file provides appropriate dry-land procedures for the evaluator to call.

A.2 Input file format

The syntax of the planner's input file is now specified using Backus Naur Form notation.

```
<input-file> ::= <belief-model> <costs> <reward> <seed>

<belief-model> ::= belief-model(<agent-name>,[<levels>]).

<levels> ::= <level>

<levels> ::= <level> , <levels>

<level> ::= nesting_level(<level-number>,[<beliefs>])

<beliefs> ::= <belief>

<beliefs> ::= <belief>, <beliefs>

<belief> ::= p(<proposition>,<probability>)

<proposition> ::= decomp(<act>,<childlist>)

<proposition> ::= intend(<proposition>)

<proposition> ::= bel(<proposition>)
```

```
<proposition> ::= <identifier>


<costs> ::= <cost>

<costs> ::= <cost> <costs>

<cost> ::= utility(<act>,<real>).


<reward> ::=  any prolog function mapping a <plan> to a <real>.
              often a constant


<seed> ::= seed_root(<act>).
```

A.3 Example input file

This section gives the input file used in example 1 of chapter 4.

```
/*****************************************/
/* experiment configuration */
/*****************************************/

/* use sampling? */
```

```
/* sampling(on,50,50). */
/* sampling(off,10,1000). */



/***********************************/
/* belief model for the blue agent */
/***********************************/


belief_model(blue_agent,[ A,
                          nesting_level(2,B),
                          nesting_level(3,C),
                          nesting_level(4,B),
                          nesting_level(5,C),
                          nesting_level(6,B),
                          nesting_level(7,C),
                          nesting_level(8,B),
                          nesting_level(9,C),
                          nesting_level(10,B),
                          nesting_level(11,C)
                        ]) :-

belief_model(blue_agent_basis,
             [A,
      nesting_level(2,B),
      nesting_level(3,C)
```

```
     ]
).


belief_model(
blue_agent_basis,
[
nesting_level(
1,
[
p(decomp(fix-car,
  [ask-car-spanner,lend-car-spanner,
    use-car-spanner]),1),
p(decomp(ask-car-spanner,
  [ask-ambiguous]),1),
p(decomp(ask-car-spanner,
  [ask-car-unambiguous]),1),

p(decomp(fix-bike,
  [ask-bike-spanner,lend-bike-spanner,
            use-bike-spanner]),1),
p(decomp(ask-bike-spanner,
  [ask-ambiguous]),1),
p(decomp(ask-bike-spanner,[ask-bike-unambiguous]),1),

p(intend(fix-car,
  [ask-car-spanner]),1),
```

```
p(intend(ask-car-spanner,
  [ask-ambiguous]),1),
p(intend(ask-car-spanner,
  [ask-car-unambiguous]),1),

p(intend(fix-bike,
  [ask-bike-spanner]),1),
p(intend(ask-bike-spanner,
  [ask-ambiguous]),0),
p(intend(ask-bike-spanner,
  [ask-bike-unambiguous]),1)
]
),
nesting_level(
2,
[
p(decomp(fix-car,
  [ask-car-spanner,lend-car-spanner,
    use-car-spanner]),1),
p(decomp(ask-car-spanner,
  [ask-ambiguous]),1),
p(decomp(ask-car-spanner,
  [ask-car-unambiguous]),1),

p(decomp(fix-bike,
  [ask-bike-spanner,lend-bike-spanner,
    use-bike-spanner]),1),
```

```
p(decomp(ask-bike-spanner,
  [ask-ambiguous]),1),
p(decomp(ask-bike-spanner,
  [ask-bike-unambiguous]),1),

p(intend(fix-car,
  [ask-car-spanner]),1),
p(intend(ask-car-spanner,
  [ask-ambiguous]),0.5),
p(intend(ask-car-spanner,
  [ask-car-unambiguous]),1),

p(intend(fix-bike,
  [ask-bike-spanner]),1),
p(intend(ask-bike-spanner,
  [ask-ambiguous]),0.5),
p(intend(ask-bike-spanner,
  [ask-bike-unambiguous]),1)
]
),
nesting_level(
3,
[
p(decomp(fix-car,
  [ask-car-spanner,lend-car-spanner,
    use-car-spanner]),1),
p(decomp(ask-car-spanner,
```

```
  [ask-ambiguous]),1),
p(decomp(ask-car-spanner,
  [ask-car-unambiguous]),1),

p(decomp(fix-bike,
  [ask-bike-spanner,lend-bike-spanner,
    use-bike-spanner]),1),
p(decomp(ask-bike-spanner,
  [ask-ambiguous]),1),
p(decomp(ask-bike-spanner,
  [ask-bike-unambiguous]),1),

p(intend(fix-car,
  [ask-car-spanner]),1),
p(intend(ask-car-spanner,
  [ask-ambiguous]),0.5),
p(intend(ask-car-spanner,
  [ask-car-unambiguous]),1),

p(intend(fix-bike,
  [ask-bike-spanner]),1),
p(intend(ask-bike-spanner,
  [ask-ambiguous]),0.5),
p(intend(ask-bike-spanner,
  [ask-bike-unambiguous]),1)
]
)
```

```
]
).



/************************************************/
/* cost model */
/************************************************/


/* nb all atoms have to have a utility */
utility(ask-ambiguous,-5).
utility(ask-car-unambiguous,-10).
utility(ask-bike-unambiguous,-10).
utility(lend-car-spanner,-10).
utility(lend-bike-spanner,-10).
utility(ask-clar,-3).
utility(answer-bike,-1).
utility(answer-car,-1).

utility(use-car-spanner,0).
utility(use-bike-spanner,0).


reward(Plan,100) :-
  plan_contains(Plan,fix-car),
  plan_contains(Plan,use-car-spanner), !.
```

```
reward(Plan,100) :-
  plan_contains(Plan,fix-bike),
  plan_contains(Plan,use-bike-spanner), !.

reward(Plan,80).



seed([alternative(plan(ask-car-spanner,
                       [plan(ask-ambiguous,[])]),
                  ask-ambiguous,
                  null
      ),

      alternative(plan(ask-car-spanner,
                       [plan(ask-car-unambiguous,[])]),
                  ask-car-unambiguous,
                  null
      )
]).
```

A.4 Index of experiment materials

Here is a list of the experiments, in their order of appearance in the thesis, and the location of the results, input and control files that corresponds with them. A brief description of the experiment procedure is given for each.

- **Chapter 4: Risking misinterpretation** This experiment corresponds with the **spanner_demos** directory. Two input files were used in this experiment. **config.pl** was used where no clarification dialogue was desired in the game tree. **config_clarify.pl** differed in that plan rules for the clarification subdialogue were added. The file **test_harness.pl** has a procedure that runs the planner for each of the demonstrations. The data generated in the experiment was written to the **data** subdirectory. There is a set of gnuplot scripts in this directory which generate plots from the data.

- **Chapter 4: Goal introduction problem** This experiment corresponds with the **window_seat_demos** directory. Two input files were used in this experiment, **config.pl** and **config2.pl**. The difference between these is that the passenger who wanted a window seat but didn't get one has a lower reward in the second configuration. The file **test_harness.pl** has a procedure that runs the planner for each of the demonstrations. The data generated in the experiment was written to the **data** subdirectory. There is a set of gnuplot scripts in this directory which generate plots from the data.

- **Chapter 4: Belief model acquisition** This experiment corresponds with the **window_seat_acquisition** directory. The input file **config.pl** is similar to that used in the goal introduction experiment. The experiment was controlled using the dialogue manager program in the file **domain_planner/dialogue_manager.pl**. Data was collected by running the Prolog interpreter in debugging mode and examining the belief state of the agent.

- **Chapter 5: Telling and proposing** This experiment corresponds

with the **negotiation_examples/propose** directory. The input file was **config.pl**. The **test** procedure in the file **test_harness.pl** was used to generate and evaluate the game tree. Different sets of acts were enabled and disabled to show the difference in utility they make. This was done by commenting out lines in the file **negotiation_planner/critical_points.pl**.

- **Chapter 5: Holding the floor** This experiment corresponds with the **negotiation_examples/ask_beat_tell** directory. The input file was **config.pl**. The file **test_harness.pl** was used to create two sets of results. First the value of bel(have-eggs) was set to 0.1 at level 3 for one set of results. For the second set, it was set to 0.8. These resultsets have indicative filenames in the data directory. There is a gnuplot script to generate the plots.

- **Chapter 5: Request and propose** This experiment corresponds with the **negotiation_examples/auto_ask_got_fruit** directory. The input file was **config.pl**. The file **test_harness.pl** was used to create the set of results. There is a gnuplot script to generate the plots.

- **Chapter 6: Misconception** This experiment corresponds with the **misconception** directory. The input file was **config.pl**. The file **test_harness.pl** was used to consult the input file and call the planner. Different belief revision strategies were tried by modifying the belief revision code. These modifications were thereafter removed.

Bibliography

[1] J Allen and C R Perrault. Analyzing intention in utterances. In *Readings in natural language processing*, pages 441–458. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1986.

[2] James F. Allen. The TRAINS project: A case study in building a conversational planning agent. *Journal of Experimental and Theoretical AI (JETAI)*, 7:7–48, 1995.

[3] James F. Allen, Bradford W. Miller, Eric K. Ringger, and Teresa Sikorski. A robust system for natural spoken dialogue. In Arivind Joshi and Martha Palmer, editors, *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 62–70, San Francisco, 1996. Morgan Kaufmann Publishers.

[4] Dana Angluin and Carl H. Smith. Inductive inference: Theory and methods. *ACM Comput. Surv.*, 15(3):237–269, 1983.

[5] Douglas E. Appelt. *Planning English Sentences*. Cambridge University Press, New York, NY, USA, 1985.

[6] J. L. Austin. *How to do Things with Words*. Oxford University Press, New York, 1962.

[7] Michael E. Bratman, David Israel, and Martha Pollack. Plans and resource-bounded practical reasoning. In Robert Cummins and John L. Pollock, editors, *Philosophy and AI: Essays at the Interface*, pages 1–22. The MIT Press, Cambridge, Massachusetts, 1991.

[8] B. C. Bruce. Belief systems and language understanding. Technical Report 2973, Bolt Beranek and Newman Inc., 1975.

[9] Sandra Carberry. A pragmatics-based approach to ellipsis resolution. *Comput. Linguist.*, 15(2):75–96, 1989.

[10] Sandra Carberry. *Plan Recognition in Natural Language Dialogue*. MIT Press, Cambridge, MA, USA, 1990.

[11] Sandra Carberry. Techniques for plan recognition. *User Modeling and User-Adapted Interaction*, 11(1-2):31–48, 2001.

[12] J. Carletta. Planning to fail, not failing to plan: Risk-taking and recovery in task-oriented dialogue. In *Proc. of the 14th COLING*, pages 896–900, Nantes, France, 1992.

[13] Eugene Charniak. Statistical techniques for natural language parsing. *AI Magazine*, 18(4):33–44, 1997.

[14] Eugene Charniak and Robert P. Goldman. A bayesian model of plan recognition. *Artif. Intell.*, 64(1):53–79, 1993.

[15] Herbert H. Clark and Edward F. Schaefer. Contributing to discourse. *Cognitive Science*, 13(2):259–294, 1989.

[16] Daniel Corkill. Hierarchical Planning in a Distributed Problem-Solving Environment. In *Proc. 7th Intl. Joint Conf. on AI*, Tokyo, January 1979.

[17] Daniel Dennett. *The Intetional Stance*. MIT Press, 1989.

[18] Marie desJardins, Edmund H. Durfee, Charles L. Ortiz Jr., and Michael Wolverton. A survey of research in distributed, continual planning. *AI Magazine*, 20(4):13–22, 1999.

[19] E.H. Durfee and V.R. Lesser. Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5):1167–1183, September 1991.

[20] J. St. B. T. Evans and D. E. Over. *Rationality and Reasoning*. Psychology Press, 1996.

[21] R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[22] Foundation for Intelligent Physical Agents. Fipa communicative act library specification sc00037j 2000. http://www.fipa.org, 2000.

[23] Peter Gardenfors, editor. *Belief Revision*. Cambridge University Press, New York, NY, USA, 1992.

[24] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In J. Allen, J. Hendler, and A. Tate, editors, *Readings in Planning*, pages 729–734. Kaufmann, San Mateo, CA, 1990.

[25] Piotr J. Gmytrasiewicz and Edmund H. Durfee. Rational communication in multi-agent systems. *Autonomous Agents and Multi-Agent Systems Journal*, 4(3):233–272, 2001.

[26] Piotr J. Gmytrasiewicz, Sanguk Noh, and Tad Kellogg. Bayesian update of recursive agent models. *User Modeling and User-Adapted Interaction*, 8(1-2):49–69, 1998.

[27] Paul Grice. *Studies in the Way of Words*. Harvard University Press, Cambridge, Mass., 1989.

[28] Barbara J. Grosz and Candace L. Sidner. Attention, intentions, and the structure of discourse. *Comput. Linguist.*, 12(3):175–204, 1986.

[29] Barbara J. Grosz and Candace L. Sidner. Plans for discourse. In Philip R. Cohen, Jerry L. Morgan, and Martha E Pollack, editors, *Intentions in Communication*, pages 417–444. MIT Press, Cambridge, MA, 1990.

[30] Curry I. Guinn. An analysis of initiative selection in collaborative task-oriented discourse. *User Modeling and User-Adapted Interaction*, 8(3-4):255–314, 1998.

[31] J. Harsanyi. Games with incomplete information played by bayesian players. *Management Science*, 14:159–82,320–34, 486–502, 1967-8.

[32] David Heckerman, Dan Geiger, and David M. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *Mach. Learn.*, 20(3):197–243, 1995.

[33] Peter A. Heeman and James F. Allen. The trains 93 dialogues. Technical report, University of Rochester, Rochester, NY, USA, 1995.

[34] Jaako Hintikka. *Knowledge and Belief: An Introduction into the logic of the two notions*. Cornell University Press, Ithaca, 1962.

[35] E. Horvitz and T. Paek. A computational architecture for conversation, 1999.

[36] Masato Ishizaki, Matthew Crocker, and Chris Mellish. Exploring mixed-initiative dialogue using computer dialogue simulation. *User Modeling and User-Adapted Interaction*, 9(1-2):79–91, 1999.

[37] R. Kass and T. Finin. A general user modelling facility. In *CHI '88: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 145–150, New York, NY, USA, 1988. ACM Press.

[38] Robert Kass and Tim Finin. Modeling the user in natural language systems. *Comput. Linguist.*, 14(3):5–22, 1988.

[39] H. A. Kautz and J. F. Allen. Generalized plan recognition. In *Proc. of AAAI-86*, pages 32–37, Philadelphia, PA, 1986.

[40] Alfred Kobsa and Wolfgang Pohl. The user modeling shell system bgp-ms. *User Model. User-Adapt. Interact.*, 4(2):59–106, 1995.

[41] Alfred Kobsa and Wolfgang Wahlster, editors. *User models in dialog systems*. Springer-Verlag New York, Inc., New York, NY, USA, 1989.

[42] Saul A. Kripke. Semantical analysis of modal logic I: Normal modal propositional calculi. *Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik*, 9:67–96, 1963.

[43] Harry R. Lewis, Christos H. Papadimitriou, and Christos Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997.

[44] D.V. Lindley. *Making Decisions*. Wiley, 1985.

[45] Diane J. Litman and James F. Allen. A plan recognition model for clarification subdialogues. In *Proceedings of the 22nd annual meeting on Association for Computational Linguistics*, pages 302–311, Morristown, NJ, USA, 1984. Association for Computational Linguistics.

[46] Bruce Lucas. Voicexml for web-based distributed conversational applications. *Commun. ACM*, 43(9):53–57, 2000.

[47] R.D. Luce and H. Raiffa. *Games and Decisions*. J. Wiley, New York, 1957.

[48] Kathleen F. McCoy. Reasoning on a highlighted user model to respond to misconceptions. *Comput. Linguist.*, 14(3):52–63, 1988.

[49] Michael F. McTear. Spoken dialogue technology: enabling the conversational user interface. *ACM Comput. Surv.*, 34(1):90–169, 2002.

[50] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[51] C. Raymond Perrault and James F. Allen. A plan-based analysis of indirect speech acts. *Comput. Linguist.*, 6(3-4):167–182, 1980.

[52] W. Pohl, A. Kobsa, and O. Kutter. User model acquisition heuristics based on dialogue acts. In *Proc. of the Fourth International Conference on User Modeling*, page 225, Hyannis, MA, 1994.

[53] Martha E. Pollack. A model of plan inference that distinguishes between the beliefs of actors and observers. In *Proceedings of the 24th conference on Association for Computational Linguistics*, pages 207–214, Morristown, NJ, USA, 1986. Association for Computational Linguistics.

[54] Lance A. Ramshaw. A metaplan model for problem-solving discourse. In *EACL*, pages 35–42, 1989.

[55] Anand S. Rao and Michael P. Georgeff. Modeling rational agents within a BDI-architecture. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1991.

[56] Charles Rich and Candace L. Sidner. COLLAGEN: A collaboration manager for software interface agents. *User Modeling and User-Adapted Interaction*, 8(3-4):315–350, 1998.

[57] E Rich. Stereotypes and user modeling. In *User Models in Dialogue Systems*, pages 75–85. Springer Verlag, Berlin, 1989.

[58] N. Roy, J. Pineau, and S. Thrun. Spoken dialogue management using probabilistic reasoning, 2000.

[59] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[60] E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[61] Harvey Sacks, Emanuel A. Schegloff, and Gail Jefferson. A simplest systematics for the organization of turn-taking for conversation. *Language*, 50(4):696–735, 1974.

[62] David Sadek and Renato de Mori. Dialogue systems. In Renato de Mori, editor, *Spoken Dialogues with Computers*, pages 69–121. Academic Press, New York, 1998.

[63] Ken Samuel, Sandra Carberry, and K. Vijay-Shanker. Dialogue act tagging with transformation-based learning. In *Proceedings of the 17th international conference on Computational linguistics*, pages 1150–1156, Morristown, NJ, USA, 1998. Association for Computational Linguistics.

[64] K. Scheffler and S. Young. Corpus-based dialogue simulation for automatic strategy learning and evaluation, 2001.

[65] J. R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, England, 1969.

[66] C. E. Shannon. A mathematical theory of communication. *Bell Sys. Tech. J.*, 27:379–423, 623–656, 1948.

[67] C. L. Sidner. Focusing in the comprehension of definite anaphora. In M. Brady and R. C. Berwick, editors, *Computational Models of Discourse*, pages 267–330. MIT Press, Cambridge, MA, 1983.

[68] H. A. Simon. *Models of Man*. Wiley, 1957.

[69] Ronnie W. Smith, Alan W. Biermann, and D. Richard Hipp. An architecture for voice dialog systems based on prolog-style theorem proving. *Comput. Linguist.*, 21(3):281–320, 1995.

[70] Andreas Stolcke, Noah Coccaro, Rebecca Bates, Paul Taylor, Carol Van Ess-Dykema, Klaus Ries, Elizabeth Shriberg, Daniel Jurafsky, Rachel Martin, and Marie Meteer. Dialogue act modeling for automatic tagging and recognition of conversational speech. *Comput. Linguist.*, 26(3):339–373, 2000.

[71] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998. A Bradford Book.

[72] Jasper A. Taylor, Jean Carletta, and Chris Mellish. Requirements for belief models in cooperative dialogue. *User Modeling and User-Adapted Interaction*, 6(1):23–68, 1996.

[73] M. Vilain. Getting serious about parsing plans: A grammatical analysis of plan recognition. In *Proc. of AAAI-90*, pages 190–197, Boston, MA, 1990.

[74] M. Walker, D. Litman, C. Kamm, and A. Abella. Evaluating spoken dialogue agents with paradise: Two case studies, 1998.

[75] Marilyn A. Walker. The effect of resource limits and task complexity on collaborative planning in dialogue. *Artificial Intelligence*, 85(1-2):181–243, 1996.

[76] Marilyn A. Walker, Jeanne C. Fromer, and Shrikanth Narayanan. Learning optimal dialogue strategies: a case study of a spoken dialogue agent for email. In *Proceedings of the 17th international conference on Computational linguistics*, pages 1345–1351, Morristown, NJ, USA, 1998. Association for Computational Linguistics.

[77] Michael Wolverton and Marie desJardins. Controlling communication in distributed planning using irrelevance reasoning. In *AAAI/IAAI*, pages 868–874, 1998.

[78] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice, 1994.

[79] Bo Zhang, Qingsheng Cai, Jianfeng Mao, Eric Chang, and Baining Guo. Spoken Dialogue Management as Planning and Acting under Uncertainty. In *Proc. 7th European Conference on Speech Communication and Technology*, Aalborg, Denmark, September 2001.