

# Structured Probabilistic Modelling for Dialogue Management

Doctoral Dissertation by

Pierre Lison



Department of Informatics  
Faculty of Mathematics and Natural Sciences  
University of Oslo

Submitted for the degree of Philosophiae Doctor

October 7, 2013



“Probability theory is nothing but common  
sense reduced to calculation.”

Pierre Simon de Laplace.  
*Théorie Analytique des Probabilités* (1812).



# **Abstract**

TODO



# Acknowledgements

TODO

**Typesetting:** The thesis is written with L<sup>A</sup>T<sub>E</sub>X 2<sub><</sub> based on a document layout prepared by the University of Oslo. The bibliographical references are generated with BIBL<sup>E</sup>X. The diagrams and data plots are drawn using Apple's *Pages* and *Numbers* applications, while the mathematical figures are rendered with the *MATLAB* numerical computing environment.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Contributions . . . . .	3
1.3	Outline of the thesis . . . . .	6
<b>2</b>	<b>Background</b>	<b>11</b>
2.1	What is spoken dialogue? . . . . .	11
2.1.1	Turn-taking . . . . .	12
2.1.2	Dialogue acts . . . . .	13
2.1.3	Interpretation of dialogue acts . . . . .	14
2.1.4	Grounding . . . . .	15
2.2	Spoken dialogue systems . . . . .	17
2.2.1	Architectures . . . . .	17
2.2.2	Components . . . . .	20
2.2.3	Applications . . . . .	23
2.3	Dialogue management . . . . .	23
2.3.1	Hand-crafted approaches . . . . .	23
2.3.2	Statistical approaches . . . . .	26
2.4	Summary . . . . .	29
<b>3</b>	<b>Probabilistic models of dialogue</b>	<b>33</b>
3.1	Graphical models . . . . .	33
3.1.1	Representations . . . . .	34
3.1.2	Inference . . . . .	38
3.1.3	Learning . . . . .	39
3.2	Reinforcement learning . . . . .	43
3.2.1	Markov Decision Processes . . . . .	44
3.2.2	Partially Observable Markov Decision Processes . . . . .	46
3.2.3	Factored representations . . . . .	48
3.3	Application to dialogue management . . . . .	49
3.3.1	Supervised learning from Wizard-of-Oz data . . . . .	49
3.3.2	MDP-based optimisation of dialogue policies . . . . .	50
3.3.3	POMDP-based optimisation of dialogue policies . . . . .	52
3.4	Summary . . . . .	55

<b>4 Probabilistic rules</b>	<b>57</b>
4.1 Structural leverage . . . . .	57
4.2 Formalisation . . . . .	60
4.2.1 Probability rules . . . . .	61
4.2.2 Utility rules . . . . .	63
4.2.3 Quantification . . . . .	64
4.3 Rule instantiation . . . . .	65
4.3.1 Probability rules . . . . .	66
4.3.2 Utility rules . . . . .	69
4.3.3 Quantification . . . . .	71
4.4 Processing workflow . . . . .	74
4.4.1 Domain representation . . . . .	74
4.4.2 Update algorithm . . . . .	75
4.4.3 Detailed example . . . . .	79
4.5 Advanced modelling . . . . .	82
4.5.1 Operations on lists . . . . .	82
4.5.2 Operations on strings . . . . .	83
4.6 Relation to previous work . . . . .	84
4.7 Conclusion . . . . .	86
<b>5 Learning from Wizard-of-Oz data</b>	<b>87</b>
5.1 Parameters of probabilistic rules . . . . .	87
5.1.1 Generalities . . . . .	87
5.1.2 Parameter priors . . . . .	89
5.1.3 Instantiation . . . . .	92
5.2 Supervised learning of rule parameters . . . . .	94
5.2.1 Wizard-of-Oz training data . . . . .	94
5.2.2 Learning cycle . . . . .	95
5.3 Experiments . . . . .	98
5.3.1 Dialogue domain . . . . .	99
5.3.2 Wizard-of-Oz data collection . . . . .	100
5.3.3 Experimental setup . . . . .	102
5.3.4 Empirical results and analysis . . . . .	106
5.4 Conclusion . . . . .	107
<b>6 Learning from interactions</b>	<b>111</b>
6.1 Bayesian reinforcement learning . . . . .	111
6.2 Optimisation of rule parameters . . . . .	114
6.2.1 Model-based approach . . . . .	114
6.2.2 Model-free approach . . . . .	119
6.3 Experiments . . . . .	121
6.3.1 Dialogue domain . . . . .	122
6.3.2 Simulator . . . . .	126
6.3.3 First experiment . . . . .	132

6.3.4	Second experiment . . . . .	133
6.4	Conclusion . . . . .	137
<b>7</b>	<b>Implementation</b>	<b>139</b>
7.1	Toolkit design . . . . .	139
7.1.1	Generalities . . . . .	139
7.1.2	Specification of dialogue domains . . . . .	141
7.1.3	Core algorithms . . . . .	143
7.1.4	Comparison with other architectures . . . . .	146
7.2	System integration . . . . .	149
7.2.1	Architecture . . . . .	149
7.2.2	Individual modules . . . . .	149
7.2.3	Graphical interface . . . . .	152
7.3	Conclusion . . . . .	153
<b>8</b>	<b>User evaluation</b>	<b>157</b>
8.1	Interaction scenario . . . . .	157
8.1.1	Dialogue domain . . . . .	158
8.1.2	System design . . . . .	161
8.1.3	Wizard-of-Oz data collection . . . . .	164
8.2	Dialogue management models . . . . .	164
8.2.1	Approach 1: handcrafted model . . . . .	165
8.2.2	Approach 2: factored statistical model . . . . .	167
8.2.3	Approach 3: rule-structured model . . . . .	170
8.2.4	Learning curves . . . . .	171
8.3	User trials . . . . .	173
8.3.1	Experimental setup . . . . .	173
8.3.2	Metrics . . . . .	173
8.3.3	Results . . . . .	173
8.3.4	Analysis . . . . .	173
8.4	Conclusion . . . . .	173
<b>9</b>	<b>Concluding remarks</b>	<b>177</b>
9.1	Summary of contributions . . . . .	177
9.2	Future work . . . . .	177
<b>A</b>	<b>Relevant probability distributions</b>	<b>179</b>
<b>B</b>	<b>Domain specifications</b>	<b>185</b>
B.1	Experiments in Section 5.3 . . . . .	185
B.2	Experiments in Section 6.3 . . . . .	187
B.3	Experiments in Chapter 8 . . . . .	191



# Mathematical notations

## Probability distributions:

$X$	Random variable
$Val(X)$	Range of values for the random variable $X$
$P(X)$	Probability distribution for the random variable $X$
$P(X_1, \dots X_n)$	Joint probability distribution for $X_1, \dots X_n$
$P(X_1, \dots X_n   Y_1, \dots Y_m)$	Conditional probability distribution for $X_1, \dots X_n$ given $Y_1, \dots Y_m$
$E(X)$	Expectation of the random variable $X$
$P(\mathbf{X}   \mathbf{e})$	Posterior distribution of variables $\mathbf{X}$ given evidence $\mathbf{e}$
$U(\mathbf{X}   \mathbf{e})$	Utility distribution of variables $\mathbf{X}$ given evidence $\mathbf{e}$
$\boldsymbol{\theta}_X$	Parameters associated with the random variable $X$
$P(X ; \boldsymbol{\theta}_X)$	Probability of $X$ given the parameters $\boldsymbol{\theta}_X$

## Graphical models:

$\mathcal{B}$	Bayesian network
$P_{\mathcal{B}}(X)$	Probability distribution for $X$ in the Bayesian network $\mathcal{B}$
$(\mathbf{X} \perp \mathbf{Y}   \mathbf{Z})$	Conditional independence of variables $\mathbf{X}$ and $\mathbf{Y}$ given $\mathbf{Z}$
$Y \rightarrow X$	Directed edge from variable $Y$ to variable $X$
$parents(X)$	Parents of variable $X$ such that $Y \rightarrow X$ for all $Y \in parents(X)$

## Reinforcement learning:

$s$	Current state
$\mathcal{S}$	Set of possible states
$s_t$	State at time $t$
$a$	System action
$\mathcal{A}$	Set of possible actions
$R(s, a)$	Immediate reward of action $a$ in state $s$
$\gamma$	Discount factor
$h$	Planning horizon
$V(s)$	Value function for state $s$ (= expected return)
$Q(s, a)$	Action-value function for action $a$ in state $s$
$\pi(s)$	MDP dialogue policy, defined as a function $\pi : \mathcal{S} \rightarrow \mathcal{A}$
$o$	Observation
$\mathcal{O}$	Set of possible observations
$b$	Belief state $b(s) = P(s)$

$\mathcal{B}$	Belief state space $\subset \Re^{ S -1}$
$V(b)$	Value function for belief state $b$
$Q(b, a)$	Action–value function for action $a$ in belief state $b$
$\pi(b)$	POMDP dialogue policy, defined as a function $\pi : \mathcal{B} \rightarrow \mathcal{A}$

### Dialogue-specific variables:

$u_u$	User utterance
$\tilde{u}_u$	Speech recognition hypotheses for user utterance
$a_u$	User dialogue act
$\tilde{a}_u$	Speech understanding hypotheses for the user dialogue act
$i_u$	User intention
$c$	Interaction context
$a_m$	System dialogue act
$u_m$	System utterance

### Probabilistic rules:

$r$	Probability or utility rule
$c_i$	$i$ -th condition of a rule, expressed as a logical formula
$e_{(i,j)}$	$j$ -th effect for condition $c_i$ , expressed as a value assignment
$p_{(i,j)}$	Probability of effect $e_{(i,j)}$
$I_1, \dots, I_k$	Set of input variables of a rule
$O_1, \dots, O_l$	Set of output variables of a probability rule
$A_1, \dots, A_l$	Set of decision variables of an utility rule
$\mathbf{e}$	Conjunction of effects $e_1 \wedge \dots e_n$
$\mathbf{e}(X)$	(Possibly empty) set of values specified for variable $X$ in $\mathbf{e}$
$\mathbf{y}$	Universally quantified variables $y_1, \dots, y_p$ of a rule
$\mathbf{g}$	Possible grounding for the universally quantified variables $\mathbf{y}$

### Miscellaneous:

$\mathbf{1}(\phi)$	Indicator function, with $\mathbf{1}(\phi) = 1$ if $\phi$ is true and 0 otherwise
$\phi[a/b]$	First-order formula $\phi$ where all occurrences of $a$ are replaced by $b$
$ \mathcal{S} $	Cardinality of the set $\mathcal{S}$ (i.e. number of elements)

# Chapter 1

## Introduction

Spoken language is one of the most powerful system of communication at our disposal. A large part of our waking hours is spent in social interactions mediated through natural language. The central role of spoken language in our daily lives is largely due to its remarkable proficiency at conveying (sometimes highly elaborate) ideas in a robust and efficient manner.

Is it possible to exploit this basic fact to develop more user-friendly technologies? Most of our everyday activities are now relying on “smart” electronic devices of various kinds, from mobile phones to personal computers and navigation systems. As these technologies gain in autonomy and sophistication, user interaction design becomes increasingly important. Human-computer interfaces should provide the user with rich interaction channels that can unlock the full potential of their applications, yet remain easy to understand and control. One natural way to achieve this goal is to endow computers with a capacity to understand, even in a limited manner, the communication medium that is most intuitive to human beings, namely spoken language.

The ongoing research on *spoken dialogue systems* (SDS) is precisely trying to realise this objective. A spoken dialogue system is a computational agent that can converse with humans through everyday spoken language. Such systems are expected to play an ever-increasing role in our interactions with technology. They have a wide spectrum of applications, ranging from voice-enabled mobile applications to in-car navigation assistants, smart home environments, tutoring systems, and (in a not-too-distant future) service robots assisting us in our daily chores.

Figure 1.1 depicts an example of interaction between a human user and a spoken dialogue system. When the user starts talking, the system extracts the corresponding speech signal through a microphone. The speech signal is then processed to analyse its content. Once this analysis is completed, the system must then determine how to react. In this case, the system decides to greet back the user and selects the words to express it (“*good morning, sir*”). The final step is then to synthesise these words through an artificial voice, which closes the loop.<sup>1</sup>

### 1.1 Motivation

Although spoken dialogue systems can greatly enhance the user interaction experience in many of today’s technologies, their practical development can be a demanding enterprise. Speech is indeed much more complex than other types of interfaces such as keyboards or touch screens.

---

<sup>1</sup> Needless to say, the schema hides a great deal of internal complexity. The next chapter describes in more detail the software architectures used to design practical spoken dialogue systems.



Figure 1.1: Schematic view of a spoken dialogue system.

The present thesis concentrates on the problem of *dialogue management*. Dialogue management is a central component in spoken dialogue systems and lies at the intersection between speech understanding and production. It serves a double role. Its first function is to maintain a representation of the current dialogue state. This representation reflects the system knowledge of the current conversational situation, and often includes multiple features related to the dialogue history, the external context, and the tasks to perform. This dialogue state is regularly updated with new information in the form of new user utterances or perceived changes in the context.

The second function of dialogue management is to make decisions. Based on the current dialogue state, dialogue management must decide which actions to undertake. These actions are often communicative in nature (e.g. uttering a sentence), but can also pertain to physical actions (e.g. grasping an object). Dialogue management is therefore responsible for controlling the flow of the interaction, by (1) interpreting the user intentions in their context and (2) selecting which actions to perform. In the example from Figure 1.1, this step corresponds to the decision of responding to the user utterance “hello computer!” with another greeting action, “good morning, sir”.

Along with speech recognition, dialogue management is arguably one of the most difficult processing task in spoken dialogue systems. This difficulty stems from two defining characteristics of verbal interactions:

1. Verbal interactions are *complex*. Taking part in a dialogue requires tracking a multitude of factors, such as the interaction history, the hypothesised goals and preferences of the dialogue participants, and the external context. These factors depend on one another through multiple relations straddling the linguistic and extra-linguistic boundaries. Selecting the action that is most appropriate in a particular situation is thus a difficult decision problem.
2. Verbal interactions are also crippled with *uncertainties*. In order to make sense of a given dialogue, a conversational agent must face numerous sources of uncertainty, including error-prone speech recognition, lexical, syntactic and referential ambiguities, partially observable environments, and unpredictable interaction dynamics.

The combination of these two properties forms an explosive mix. In order to make sense of the interaction and act appropriately, the dialogue system must resort to sophisticated reasoning in order to interpret the user intentions in their context and plan the best course of action. And it must do so under high levels of noise and uncertainty, where many pieces of information can be erroneous, missing, ambiguous, or fragmentary. This task is defined in the artificial intelligence literature as *sequential decision-making under uncertainty*, and is known to be a difficult computational problem, especially for domains as complex as spoken dialogue. Decision-making and action execution must also occur in real-time, since dialogue is by nature a real-time process.

Research on dialogue management can be divided into two main lines of investigation that reflect their focus on either of the two challenges we just mentioned.

On the one hand, structural complexity is often dealt with conceptual tools borrowed from formal logic and classical planning. These approaches provide principled methods for the interpretation and generation of dialogue moves through logical reasoning on the basis of a formal representation of the mental states of the dialogue participants (including their shared knowledge). Based on such representation, dialogue is then framed as a collaborative activity in which the dialogue participants work together to coordinate their actions, maintain a shared conversational context, resolve open issues and satisfy social obligations (???). These approaches can yield detailed analyses of various conversational behaviours, but they generally assume complete observability of the dialogue state and provide only a limited account of errors and uncertainties. In addition, they require the knowledge base on which the inference is grounded to be completely specified in advance by domain experts. Their deployment in practical applications is therefore non trivial.

On the other hand, the problem of uncertainty is usually addressed by probabilistic modelling techniques (???). The state of the dialogue is here represented as a probability distribution over possible worlds. This distribution represents the system's current knowledge of the interaction and is regularly updated as new observations are collected. These probabilistic models provide an explicit account for the various uncertainties that can arise during the interaction. They also enable the dialogue behaviour to be automatically optimised in a data-driven manner instead of relying on hand-crafted mechanisms. Dialogue strategies can therefore be adapted to new environments or users without having to be reprogrammed. However, probabilistic models typically depend on large amounts of training data to estimate their parameters – a requirement that is hard to satisfy for many dialogue domains. Probabilistic models of dialogue are also usually limited to a handful of state variables and are difficult to scale to domains featuring rich conversational contexts.

The work described in this thesis aims at reconciling these two strands of research through a new, hybrid framework to dialogue modelling and control.

## 1.2 Contributions

The present thesis exposes an original approach to dialogue management based on *structured probabilistic modelling*. The overarching motivation for this work is to design probabilistic models of dialogue that are scalable to rich conversational domains yet only require modest amounts of training data for their optimisation.

An extensive body of work in the machine learning and decision-theoretic planning literature shows how to confront this issue by relying on more expressive representations, able to capture relevant aspects of the problem *structure* in a compact manner. By taking advantage of hierarchical or

relational abstractions, system designers can leverage their domain knowledge to yield probabilistic models which are both easier to learn (due to a reduced number of parameters) and more efficient to use (since the structure can be exploited by the inference algorithm).

This thesis demonstrates how to translate these insights in dialogue modelling.

*Probabilistic graphical models* (?) constitute the theoretical foundations for a large part of our work. Graphical models provide a generic, principled framework for representing and reasoning over complex probabilistic problems. They also come with well-defined data structures and efficient general-purpose algorithms for model estimation and inference. As shown in previous work (see for instance ?), one can elegantly represent the dialogue state as a Bayesian network (a well-known type of directed graphical model) factored in a set of state variables describing various aspects of the conversational situation. The dialogue state is graphically depicted as a directed acyclic graph where the nodes correspond to particular variables and the edges are conditional dependencies between variables. To exploit such representation for decision-making tasks, the dialogue state can be extended with action and utility nodes that describe the utility for the agent of performing particular actions in a given situation.

The statistical estimation of such complex probabilistic structures is however a non-trivial endeavour, owing to the large number of variables and dependencies involved. The main novelty of our approach is the idea of representing the model distributions in a structured manner through the use of *probabilistic rules*. Probabilistic rules encode conditional distributions between variables in terms of structured mappings associating particular conditions expressed on a set of input variables to probabilistic effects on a set of output variables. Utility distributions are also encoded in a similar manner. The conditions and effects of probabilistic rules are defined as first-order logical formulae. As new information becomes available to the dialogue manager, the Bayesian network representing the dialogue state is updated by instantiating the rules in the form of new nodes mediating between the input and output variables. Probabilistic rules are therefore employed as *high-level templates* for the generation of a classical probabilistic model.

The resulting modelling framework offers two major benefits. Most importantly, the reliance on more expressive representations can drastically reduce the number of parameters associated with the models. Instead of being encoded through traditional probability tables, the conditional distributions between states variables are expressed through high-level rules that capture conditional dependences with a compact set of parameters (one for each possible effect). As a consequence, these models are much easier to learn and generalise to unseen data. In addition, the framework enables expert knowledge to be directly integrated in the probabilistic dialogue models. System developers can therefore exploit powerful abstractions to encode their prior knowledge of the dialogue domain in the form of pragmatic rules or task-specific constraints. While the usefulness of domain-specific know-how has long been recognised, their use has most often be reduced to a mere external filter for classical statistical models (??). By contrast, our approach incorporates such knowledge source in the very structure of the statistical model.

We conducted several experiments to assess the validity of our approach in three distinct learning scenarios:

1. The first experiment focused on the problem of estimating the utilities of various system actions given a small data set collected from Wizard-of-Oz interactions.<sup>2</sup> Based on dialogue

---

<sup>2</sup>A Wizard-of-Oz interaction is an experimental procedure borrowed from the field of human-computer interaction

models encoded with probabilistic rules, the utilities of the different actions were learned through imitation learning. We were able to show that the rule structure enabled the learning algorithm to converge faster and with better generalisation performance than unstructured models. This work was originally presented in (?).

2. The second experiment extended the above approach to reinforcement learning. The goal of this study was to estimate the transition model of the domain from interactions with a user simulator. We compared the relative learning performance of two modelling approaches: one relying on unstructured distributions, and one based on probabilistic rules. The empirical results demonstrated the benefits of capturing the domain structure with probabilistic rules. The results were first published in (?).
3. Finally, the third experiment was designed to evaluate the approach through live interactions with real users. **to be completed**

An additional contribution of this thesis is the development of a software toolkit that implements all the data structures and algorithms presented in this work. The toolkit is called *openDial* and is freely available under an open source licence.<sup>3</sup> The purpose of the toolkit is to enable system developers to design and evaluate dialogue systems based on probabilistic rules. All domain-specific knowledge is declaratively specified in the rules for the domain. The system architecture is therefore reduced to a small set of core algorithms for accessing and updating the dialogue state (?). This architectural design makes the toolkit fully generic and domain-independent. The *openDial* toolkit comes with a user interface allowing developers to interactively test their system and visualise how the internal dialogue state is evolving over time.

We carried out all the experiments described in this thesis in a particular application domain, namely *human–robot interaction* (HRI). The choice of this application domain as a test bed for our framework was motivated by two factors. The first factor is the presence of a complex situated environment in which the agent must complete its tasks. This dialogue context is highly dynamic – the physical position of objects and persons may for instance change in the course of the interaction, and the system tasks are regularly altered as a result of the coordinated actions of the robot and human user. The second factor relates to the occurrence of multiple sources of uncertainty caused

---

(?). In a Wizard-of-Oz experiment, the human subjects are asked to interact with a computer system that has all the appearances of reality, but is actually remotely controlled by an (unseen) human agent operating behind the curtains. Wizard-of-Oz studies are often conducted to provide the system designers with interaction data from real users before the system is fully implemented. The term is a cultural reference from the 1939 film “The Wizard of Oz” (based on an earlier book by Frank Baum), where an illusionist impersonates a powerful wizard by controlling an intimidating display from behind a curtain.

<sup>3</sup>The toolkit can be downloaded at <http://opendial.googlecode.com>. (not ready yet)

by e.g. imperfect sensory devices, unreliable motors, and failure-prone speech recognition.<sup>4</sup> This combination of a rich conversational context and high levels of uncertainty is precisely the focus of our thesis work and justifies the selection of human-robot interaction as a test bed to evaluate the performance our modelling approach in real settings.

The Nao robot from Aldebaran Robotics was used as a development and testing platform in all our experiments.<sup>5</sup> An example of interaction with the robot is shown in Figure 5.7. Most of our experiments involved the Nao robot interacting with a human user in a shared visual environment featuring a few basic objects that can be both perceived and grasped by the robot. The interactions typically revolved around the completion of a few simple tasks such as moving an object from one position to another under the supervision of the human user. Chapters 5–8 provide a detailed description of the interaction scenarios, data collection and evaluation set-ups followed for the experiments conducted as part of our thesis work.

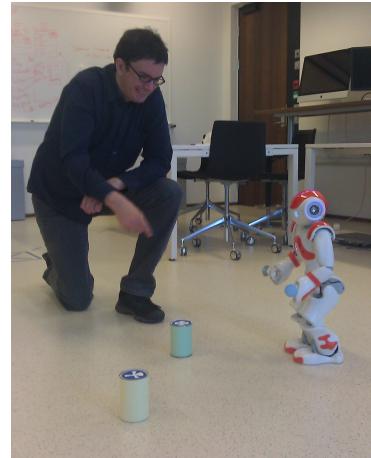


Figure 1.2: Human user interacting with the Nao robot.

## 1.3 Outline of the thesis

We provide here a brief outline of the thesis structure, chapter by chapter.

### Chapter 2: Background

This chapter introduces the fundamental concepts and methods used throughout this thesis. We start with an overview of some of the core linguistic properties of dialogue and describe key notions such as turn-taking, dialogue acts and grounding. We then describe the software architectures used to design spoken dialogue systems and the role of each component within them. We also mention a range of important applications for spoken dialogue systems. Finally, we survey the various approaches that have been put forward in the research literature to address the dialogue management problem, including both hand-crafted and statistical approaches.

### Chapter 3: Probabilistic modelling of dialogue

The chapter starts by reviewing the core notions of directed graphical models, which constitute the formal basis for our framework. We define how Bayesian networks are constructed, and show how they can be augmented to capture temporal sequences and decision-theoretic problems. We also briefly describe the most important methods for learning and inference

---

<sup>4</sup>For practical reasons, the microphones capturing the speech signals are often placed on the robot itself, and are therefore located at a significant distance from the speaker. This distance between source and receiver is a major degradation factor in speech recognition (?). Moreover, the microphones are also adjacent to a number of mechanical motors that may disturb the sound signal and lead to spurious detections.

<sup>5</sup>cf. <http://www.aldebaran-robotics.com>.

based on such models. We then move to the field of reinforcement learning and spell out its most central elements, such a Markov Decision Processes, value functions and policies. We also examine how reinforcement learning methods can be extended to partially observable settings, since dialogue is a prototypical example of decision-making under partial observability. Finally, the last section translate these concepts to the field of dialogue management, and discusses both supervised and reinforcement learning approaches to the optimisation of dialogue policies.

## **Chapter 4: Probabilistic rules**

This chapter lays down the central concepts and algorithms of our own modelling approach to dialogue management. We define what probabilistic rules are and how they are internally structured through conditions and effects. We describe two main types of rules, used to respectively encode probability and utility distribution. We then explain how the rules are practically instantiated in the Bayesian Network representing the dialogue state, as well as the processing workflow that is followed to update the dialogue state and perform action selection. The chapter also addresses some advanced modelling questions, and concludes by comparing our framework to previous work.

## **Chapter 5: Learning from Wizard-of-Oz data**

This chapter shows how the parameters attached to probabilistic rules can be automatically learned from training data, in a supervised learning fashion. The algorithm used for estimating the rule parameters is grounded in Bayesian learning techniques. To validate our approach, we detail an experiment on a statistical estimation task based on Wizard-of-Oz data collected in a human–robot interaction domain. The experiment illustrates the benefits of probabilistic rules compared to unstructured distributions.

## **Chapter 6: Learning from interactions**

Chapter 6 extends parameter estimation to a reinforcement learning context. We show how the parameters of rule-structured dialogue models can be efficiently learned from observations collected during the interaction itself, without having access to any gold standard annotations. The learning procedure follows a model-based Bayesian reinforcement learning approach. Finally, we report the results of an experiment carried out with a user simulator. The experiment concentrated on the estimation of the transition model for a human–robot interaction domain, and evaluated the relative performance of a model structured with probabilistic rules compared to a plain probabilistic model.

## **Chapter 7: Implementation**

Chapter 7 uncovers how the various algorithms and data structures presented in this thesis are technically integrated in the system architecture. We explain how the openDial toolkit is structured, describe how dialogue domains are practically specified in a generic XML format and discuss the implementation and performance tuning of the algorithms used for probabilistic inference and online planning. We also briefly compare the openDial architecture to

related software frameworks. Finally, the chapter presents the integrated dialogue system employed to carry out the experiments in this thesis and the graphical user interface developed to visualise the evolution of the dialogue state over the course of the interaction.

### **Chapter 8: User evaluation**

This chapter presents a user evaluation of our approach in a HRI domain. XXX

### **Chapter 9: Concluding remarks**

The final chapter concludes this dissertation with a summary of the presented research contributions, followed by an outline of future work.

ba



# Chapter 2

## Background

We introduce in this chapter the most important concepts and methods employed in the field of spoken dialogue systems, with special emphasis on dialogue management. We start by reviewing some key linguistic concepts that are particularly relevant for our work: turn-taking, dialogue acts and grounding. A proper understanding of these aspects is indeed a prerequisite for the design of conversationally competent dialogue systems. After this linguistic overview, we move to a more technical discussion of the software architectures used to implement practical dialogue systems. These architectures typically comprise multiple processing components, from speech recognition to understanding, dialogue management, output generation and speech synthesis. We briefly describe the role of each component and their positions in the global processing pipeline.

Last but not least, the final section of this background chapter delves into the diverse set of approaches that have been put forward to tackle the dialogue management problem. We first present hand-crafted approaches, starting with finite-state policies and pursuing with more sophisticated methods based on logic- or plan-based reasoning. Finally, we survey the more recently developed statistical approaches to dialogue management that seek to automatically extract dialogue strategies from data, based on supervised and reinforcement learning methods.

### 2.1 What is spoken dialogue?

We communicate in order to fulfil a wide array of social functions, such as exchanging ideas, collecting experiences, sustaining relationships, or collaborating with others to accomplish shared goals. These communication skills are developed in early childhood, and our cognitive abilities are in many ways shaped and amplified by this disposition for verbal interaction.

One of the most important property of dialogue is that it is fundamentally a *collaborative activity*, with emphasis on both words. It is, first of all, an *activity* motivated by the desire to fulfil specific (practical or social) goals. This activity is subject to particular costs to minimise (the communication effort) and is composed of a temporal sequence of basic actions (the dialogue turns). Furthermore, if we abstract from so-called “internal dialogues” with oneself, dialogue involves per definition at least two participants that must act together to keep the dialogue on track. As shown by a wealth of studies in psychology and linguistics (?????), human conversations are characterised by a high degree of *collaboration* between interlocutors. Individuals participating in a dialogue routinely collaborate in order to coordinate their contributions and ensure mutual understanding, thereby making the interaction more efficient. This collaboration is done mostly unconsciously and

is part and parcel of the conversational skills we develop as speakers of a given language.

We describe in the next sections four major aspects of this collaborative activity:

1. The dialogue participants take *turns* in a conversation.
2. These turns are structured into basic communicative units called *dialogue acts*.
3. The interpretation of these dialogue acts is subordinated to the *conversational context* in which they are uttered.
4. The participants continuously provide *grounding signals* to each other in order to indicate how they understand (or fail to understand) each other's contributions.

### 2.1.1 Turn-taking

Turn-taking is one of the most basic (yet often neglected) aspect of spoken dialogue. The physical constraints of the communication channel impose that participants take turns in order to speak. Turn-taking is essentially a resource allocation problem. In this case, the resource to allocate is called the *conversational floor*, and social conventions dictate how the dialogue participants are to take and release their turns.

The field of *conversation analysis* studies what these conventions are and how they interact to shape conversational behaviours. The control of the conversational floor in human conversations is indeed remarkably efficient. Empirical cross-linguistic studies have shown that the average transition time between turns revolves around 250 ms. (?).<sup>1</sup> In addition, most of the utterances do not overlap: ?) indicates that less than 5 % of the speech stream contains some form of overlap in spontaneous conversations.

A wide variety of cues are used to detect turn boundaries, such as silence, hesitation markers, syntax (complete grammatical unit), intonation (rising or falling pitch) and body language, as described by ?). These cues can occur jointly or in isolation. Upon reaching a turn boundary, a set of social conventions governs who is allowed to take the turn. The current speaker can explicitly select the next person to take the turn, for instance when greeting someone or asking a directed question (?). This selection can also occur via other mechanisms such as gaze. When no such selection is indicated, other participants are allowed to take the turn. Alternatively, the current speaker can continue to hold the floor until the next boundary.

Turn-taking is closely related to the notion of *initiative* in human–computer interaction. The vast majority of dialogue systems currently deployed are either system-initiated or user-initiated. In a system-initiated dialogue, the dialogue system has full control on how the interaction is unfolding – i.e. the system is asking all the questions and waiting for the user responses. A user-initiated dialogue is the exact opposite: in such settings, the user is assumed to lead the interaction and request information from the system. The most complex – but also most natural – interaction style is the mixed-initiative, where both the user and the dialogue system are allowed to take the initiative at any time and decide to either provide or solicit information whenever they see fit (?).

---

<sup>1</sup>Interestingly, this duration is shorter than the time required for a human speaker to plan the motor routines associated with the physical act of speaking. This means that the next speaker must start planning his utterance before the current turn is complete, and predict when a potential turn boundary is likely to appear.

The turn-taking behaviour of most current-day dialogue systems remains quite rudimentary. The most common method to detect the end of a user turn is to wait for a silence longer than a manually fixed threshold, typically ranging between  $\frac{1}{2}$  and 1.0 second. Some system architectures also include routines for handling barge-ins – that is, user interruptions – (?), while others simply ignore them altogether. Turn-taking has recently become a focus of research in its own right in the dialogue system literature (??), in an effort to break away from the ping-pong interaction style that characterises most current dialogue interfaces.

### 2.1.2 Dialogue acts

Each turn is constituted of one or more utterances. As argued by ?) and ?), utterances are nearly always purposeful: they have specific goals and are intended to provoke a specific psychological effect on the listener(s). They are therefore best described as actions rather than abstract statements about the world. The notion of dialogue act embodies precisely this idea.<sup>2</sup> ?) defines a dialogue act as a “functional unit of a dialogue used by the speaker to change the context”.

In his seminal work on the philosophy of language, ?) established a taxonomy of speech acts divided in five central categories:

**Assertives:** Committing the speaker to the truth of a proposition.

Examples: “*I swear I saw him on the crime scene.*”, “*I bought more coffee.*”

**Directives:** Attempts by the speaker to get the addressee to do something.

Examples: “*Clean your room!*”, “*Could you post this for me?*”

**Commissives:** Committing the speaker to some future course of action.

Examples: “*I will deliver this review before Monday.*”, “*I promise to work on this.*”

**Expressives:** Expressing the psychological state of the speaker about a state of affairs.

Examples: “*I am so happy for you!*”, “*Apologies for being late.*”

**Declaratives:** Bringing about a different state of the world by the utterance.

Examples: “*You’re fired.*”, “*We decided to let you pass this exam.*”

Modern taxonomies of dialogue acts are significantly more detailed than the one introduced by Searle. They also provide detailed accounts of various dialogue-level phenomena such as grounding (cf. next section) that were absent from Searle’s analysis. The most well-known annotation scheme is DAMSL (Dialogue Act Markup in Several Layers) and was initially formalised by ?). DAMSL defines a rich, multi-layered annotation scheme for dialogue acts that is both domain- and task-independent. A modified version of this scheme was applied to annotate the Switchboard corpus<sup>3</sup> based on a set of 42 distinct dialogue acts (?), including greeting and closing actions, acknowledgements, clarification requests, self-talk, responses, and many more. An interesting aspect of DAMSL is the use of two complementary dimensions in the markup: the *forward-looking functions*, which

<sup>2</sup>Dialogue acts have gone through multiple names over time, owing to the diverse range of research fields that have studied them, from philosophy to descriptive and computational linguistics. As listed in ?), alternative denominations include speech acts (?), communicative acts (?), conversation acts (?), conversational moves (?), and dialogue moves (?).

<sup>3</sup>The Switchboard corpus is a corpus of spontaneous telephone conversations collected in the early 1990’s. It includes about 2430 conversations averaging 6 minutes in length; totalling over 240 hours of recorded speech with native speakers of American English (?).

are the traditional speech acts in Searle's sense (assertions, directives, information requests, etc.) and the *backward-looking functions* that respond back to a previous dialogue act and can signal agreement, understanding, or provide answers. Both backward- and forward-looking functions can be present in the same utterance.

Determining the dialogue act corresponding to a given utterance is a non-trivial operation. The type of utterance only gives a partial indication of the underlying dialogue act – a question can for instance express a directive (“*Could you post this for me?*”). In order to accurately classify a dialogue act, a variety of linguistic factors must be taken into account, such as prosody, lexical, syntactic and semantic features, and the preceding dialogue history (????).

### 2.1.3 Interpretation of dialogue acts

Dialogue acts are strongly contextual in nature: their precise meaning can often only be comprehended within the particular conversational context in which they appear. The successful interpretation of dialogue acts must therefore venture beyond the boundaries of the isolated utterance. We briefly review here three striking aspects of this dependence on context.

#### Non-sentential utterances

Non-sentential (also called elliptical) utterances are linguistic constructions that lack an overt predicate. They include expressions such as “*where?*”, “*around 8 PM*”, or “*brilliant!*”. Their interpretation generally requires access to the recent dialogue history to recover their intended meaning. This can lead to ambiguities in the resolution, as illustrated in these examples modified from ?):

- A: “When do they open the new station?” → B: “Tomorrow” (*short answer*)
- A: “They open the station today” → B: “Tomorrow” (*correction*)
- A: “They open the station tomorrow” → B: “Tomorrow” (*acknowledgement*)

Various accounts of non-sentential utterances have been proposed, based on e.g. discourse coherence (?) or interaction-oriented semantics (??). Machine learning approaches have also been developed (??).

#### Implicatures

As shown by ?), an important part of the semantics of dialogue acts is not explicitly stated but rather implied from the context. Consider the following constructed example:

- A: Is William working today?  
B: He has a cold.

In order to retrieve the “suggested” meaning behind B’s utterance – namely, that William is probably not working –, one needs to assume that B is cooperative and that his response is therefore relevant to A’s question. If an utterance initially seems to deliberately violate this principle, the listener must search for additional hypotheses required to make sense of the dialogue act. ?) formalised these ideas in terms of a cooperative principle composed of four conversational maxims that are assumed to hold in a natural conversation: the maxim of quality (“be truthful”), the maxim of quantity

(“be exactly as informative as required”), the maxim of relation (“be relevant”) , and the maxim of manner (“be relevant”). These notions have been further developed by various theorists such as ?) and ?). A computational account of these implicatures (and application to dialogue systems) is provided by ?).

### Referring expressions

Finally, dialogue acts are replete with linguistic expressions that refer to some aspect of the conversational context. These references can be either deictic or anaphoric.

A deictic marker is a reference to an entity that is determined by the context of enunciation. Examples of such markers are “here” (spatial reference), “yesterday” (temporal reference), “this mug” (demonstrative), “you” (reference to a person), or even pointing gestures. By their very definitions, deictic markers refer to different realities depending on the situation in which they are used: a “here” uttered in a classroom differs from a “here” uttered in the countryside.

In addition, dialogue can also include anaphoric expressions – that is, expressions that refer to an element that has been previously mentioned through the history of the dialogue. An simple example of such anaphoric expression can be seen in the question-answer pair “*Is William working today?*” → “*He has a cold*”, where the pronoun “he” must be resolved to “William”.

The appropriate processing of deictic and anaphoric expressions is an important question in dialogue systems, and pertains both to the interpretation and production process. Multiple approaches have been pursued, relying on symbolic (?) or statistical techniques (??). Researchers have also investigated the integration of salience measures (?), multimodal cues (??), the processing of spatial referring expressions (?) and the incrementality of the resolution process (??).

#### 2.1.4 Grounding

Dialogue acts are executed as part of a larger collaborative activity that requires the active coordination of all conversational partners, i.e. speaker(s) as well as hearer(s). This coordination takes place at various levels. The first and most visible level is the content of the conversational activity. The partners must ensure mutual understanding of each other’s contribution, to control that they remain “on the same page”. In addition, they also coordinate the process by which the conversational activity moves forward – by signalling that they are attending to the person who currently holds the conversational floor and acknowledging her/his contributions to the dialogue.

As an illustration, consider this short excerpt from a conversation transcribed in the British National Corpus (?) :

KATHLEEN : How come they can take time off yet you can’t?  
STEVE : He’s been there longer than me.  
KATHLEEN : Oh.  
STEVE : I can, I might have two holidays now, two days’ holiday. ...  
KATHLEEN : Well ... I don’t get that, me.  
STEVE : What?  
KATHLEEN : All these two days’ holiday and this, you’ve had Christmas.  
STEVE : You get two point summat<sup>4</sup> days per month worked

---

<sup>4</sup>“Summat” is slang for “something” in the Yorkshire region.

KATHLEEN : Oh so you should've got them for January? ...  
 STEVE : right?  
 KATHLEEN : Yeah.  
 STEVE : And I worked three month before Christmas so I got six point summat days  
 KATHLEEN : For Christmas.  
 STEVE : so then I had all Christmas off.  
 KATHLEEN : Oh!  
     Yeah I get it now.  
     ... I thought you got Christmas off like we got Christmas off.  
 STEVE : No.  
     You gotta earn them. ...

(<http://www.phon.ox.ac.uk/SpokenBNCdata/KCX.html>)

We can observe in this short dialogue that the interlocutors constantly rely on the *common ground* of the interaction to move the dialogue forward. They regularly check what pieces of information are mutually known and understood (e.g. “right?”). They also make use of a variety of signals to indicate when things are properly grounded (“oh”, “yeah”, “I get it”) and when they are not (“I don’t get that”, “what?”). The common ground progressively expands as the dialogue unfolds – for instance, the system of holiday entitlement is not initially part of the shared knowledge for both speakers at the onset of the conversation, but becomes so towards the end.

The common ground is defined as the collection of shared knowledge, beliefs and assumptions that is established during an interaction.<sup>5</sup> Each dialogue act is built upon the current common ground and participates in its gradual expansion and refinement. This process is called *grounding*. A variety of feedback mechanisms can be used to this effect. As described by ?), positive evidence of understanding can be expressed via cues such as:

**Continued attention:** The hearer shows that he/she continues to attend to the speaker.

**Relevant next contribution:** The hearer produces a relevant follow-up, as in the answer “He’s been there longer than me” following the question that precedes it.

**Acknowledgement:** The hearer nods or utters a backchannel such as “mm”, “uh-uh”, “yeah”, or an assessment such as “I see”, “great”, “I get it now”.

**Demonstration:** The hearer demonstrates evidence of understanding by reformulating or completing the speaker utterance.

**Display:** The hearer reuses part of the previous utterance.

---

<sup>5</sup>An information that is part of the common ground for a given group is more than simply known by every member of the group. All group members must also be aware that the information is shared and known by the other members. Formally speaking, a proposition  $p$  is part of the common knowledge for a group of agents  $G$  when all the agents in  $G$  know  $p$ , and they also all know that they all know  $p$ , and they all know that they all know that they all know  $p$ , and so on *ad infinitum*. This definition can be formalised using the mathematical apparatus of set theory or epistemic logic (?).

Communication problems can also occur, owing to e.g. misheard or misunderstood utterances. The hearer should in this case provide negative feedback to signal trouble in understanding. A large panel of clarification and repair strategies are available to recover from these communicative failures. These strategies include backchannels (“*mm?*”), confirmations (“*Do you mean that...?*”), requests for disambiguations, invitations to repeat, and tentative corrections.

All in all, these positive and negative signals enable the dialogue participants to dynamically synchronise what the speaker intends to express and what the hearers actually understand. This grounding process operates mostly automatically, without deliberate effort. It is closely related to the concept of interactive alignment that has recently been articulated by [??](#)). Humans show a clear tendency to (unconsciously) imitate their conversational partners. In particular, they automatically align their choice of words, a phenomenon called lexical entrainment [?](#)). But alignment also occurs on several other levels such as grammatical constructions [?](#), pronunciation [?](#), accents and speech rate [?](#), and even gestures and facial expressions [?](#).

A proper treatment of grounding is critical for the development of conversational interfaces. As already mentioned in the introductory chapter, comprehension errors are indeed ubiquitous in spoken dialogue systems. The potential sources of misunderstandings are abundant, from error-prone speech recognition to out-of-domain utterances, unresolved ambiguities, and unexpected user behaviour. Appropriate grounding strategies are crucial to address these pitfalls. Grounding for dialogue systems is an active area of research and important advances have been made regarding the formalisation of rich computational models of grounding [??](#)), the generation of clarification requests [??](#)), the design of human-inspired error handling strategies [?](#)), the integration of non-verbal cues such as gaze, head nods and attentional focus [?](#) and the development of incremental grounding mechanisms [?](#)).

## 2.2 Spoken dialogue systems

After reviewing some of the core properties of human dialogues, we now discuss how to develop practical computer systems that aim to emulate such type of conversational behaviour. In the introduction chapter, Figure 1.1 represented a dialogue system as a black box taking speech inputs from the user and generating spoken responses. Real-world dialogue systems have however a complex internal structure, as we detail in the next pages.

### 2.2.1 Architectures

Spoken dialogue systems (SDS) often take the form of complex software architectures that encompass a wide range of interconnected components. These components are dedicated to various tasks related to speech processing, understanding, reasoning and decision-making. These tasks can be grouped into five major components:

1. *Speech recognition*, in charge of mapping the raw speech signal to a set of recognition hypotheses for the user utterance(s).
2. *Natural language understanding*, in charge of mapping the recognition hypotheses to high-level semantic representations of the dialogue act performed by the user.

3. *Dialogue management*, in charge of interpreting the purpose of the dialogue act in the larger conversational context and deciding what communicative action to perform (if any).
4. *Natural language generation*, in charge of finding the best linguistic (and extra-linguistic) realization for the selected communicative action.
5. And finally, *text-to-speech synthesis*, in charge of synthesizing an audio signal out of the generated utterance.

Figure 2.1 shows the flow of information for a prototypical spoken dialogue system. In practice, dialogue system architectures often employ additional middleware to provide the “software glue” between the components and handle the information exchange and scheduling of modules (????).

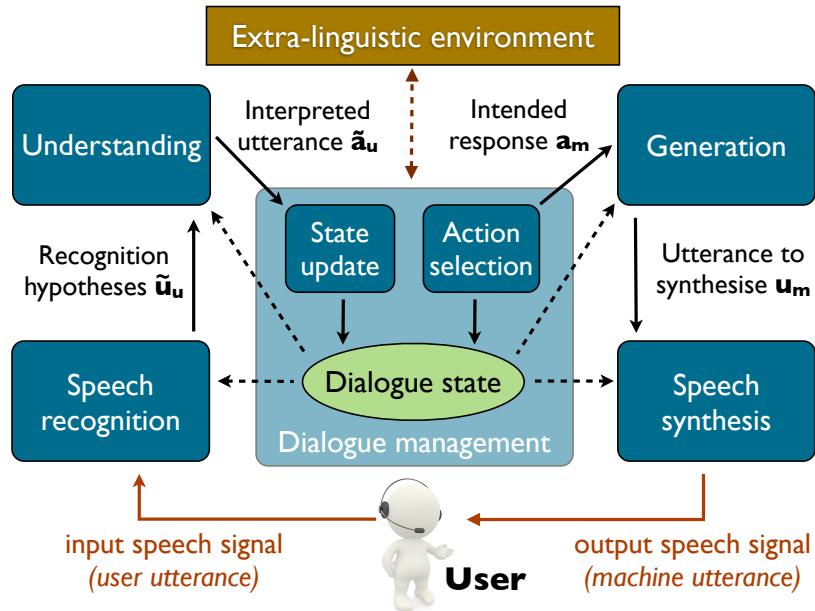


Figure 2.1: Information flow for a typical spoken dialogue system. The solid lines denote necessary input and outputs while the dotted lines represent optional contextual information.

Spoken dialogue systems can use other modalities than speech. In particular, additional communication channels such as touch, gestures, gaze, and other body movements can be fruitfully exploited. As shown by e.g. (?), multiple modalities can be employed to enrich communication in both directions (understanding and generation). In particular, the system can refine its understanding of the actual user intentions by fusing information perceived through multiple information channels such as gestures (?) or gaze (?). Non-verbal modalities can also be put to use to enhance how information is presented back to the user and convey additional grounding signals, through e.g. facial expressions and gestures. The use of multiple modalities can notably reduce understanding errors and cognitive load (?) as well as improve the overall user experience (?). For all their advantages, multimodal architectures pose however a number of additional challenges related to timing, synchronisation (?) and increased system complexity.

In addition to these non-verbal modalities, many dialogue domains are also grounded in an external context that must be accounted for. This external context might be a physical environ-

ment for human-robot interaction, a virtual world for embodied virtual agents, a spatial location for in-car navigation systems, or simply a database of factual knowledge for information systems. Contextual factors of relevance for the application must be continuously monitored by the dialogue system (and updated whenever necessary), as many components depend on the availability of such context model for their internal processing. Furthermore, the agent can often actively influence this context through external actions – for instance, a grasping action will modify the location of the gripped object. This contextual awareness necessitates the integration of additional functionalities for perception and actuation. In human–robot interaction domains, these extra-linguistic modules can notably include subsystems for object and scene recognition, spatial navigation, and various motor routines for locomotion and manipulation (??).

Several types of architectures have been proposed to assemble these components in a unified framework. The simplest approach is to arrange the components sequentially in a pipeline starting from speech recognition and ending with speech synthesis. This approach, although relatively straightforward to develop, suffers from a number of shortcomings, amongst which the rigidity of the information flow and the difficulty of inserting feedback loops between components. Pipelines also offer poor turn-taking capabilities, since the system is unable to react before the pipeline has been fully traversed (?). More advanced architectures – including the one put forward in this thesis – are based on the notion of *information state* (??). These approaches are essentially blackboard architectures revolving around a central dialogue state that is read and written by various modules connected to it. These modules monitor the state for relevant changes, in which case they trigger their processing routines and update the state with the result. The main advantages of such architectures are (1) a more flexible information flow, since the modules are allowed to process and update information in any order, and (2) the possibility to define modules that take full advantage of the contextual information encoded in the dialogue state. Figure 2.2 provides a graphical illustration of the difference between pipeline and information-state-based architectures.

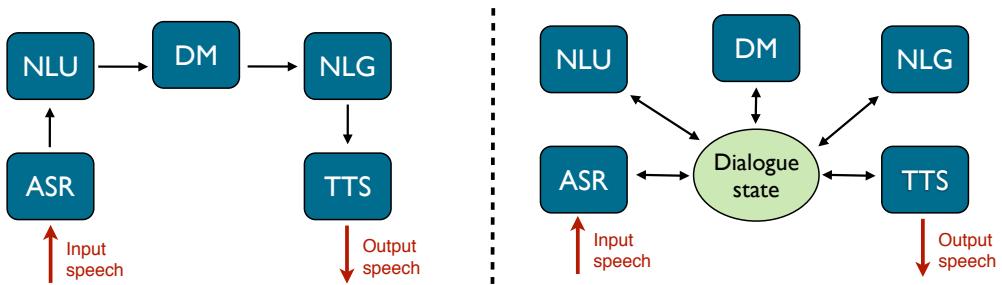


Figure 2.2: Comparison between pipeline (left) and information state (right) system architectures. ASR = *Automatic Speech Recognition*, NLU = *Natural Language Understanding*, DM = *Dialogue Management*, NLG = *Natural Language Generation*, and TTS = *Text-to-Speech Synthesis*.

Finally, a last aspect of dialogue system architectures that has been subject to recent research pertains to *incremental processing*. Many dialogue architectures must wait for an utterance to be fully pronounced to start its interpretation and decide on subsequent actions. This workflow usually leads to poor reactivity and unnatural conversational behaviours. To address this shortcoming, new architectures have been proposed to integrate incremental processing at various stages of interpretation and decision-making (?).

## 2.2.2 Components

As explained in the previous section, the components of a dialogue systems can typically be grouped in five major steps. We briefly describe here the role of these components and define their respective inputs and outputs.

### Speech recognition

Upon detection of a new speech signal emanating from the user, the first task is to recognise the corresponding utterance. Speech recognition is responsible for converting the raw speech signal from the microphone(s) into a set of hypotheses  $\tilde{u}_u$  representing the words uttered by the user. To this end, the speech signal is first converted into a digital format and split into short frames (usually 10 ms). A set of acoustic features is then extracted for each frame using signal processing techniques. Once these acoustic features are extracted, two statistical models are combined to estimate the most likely recognition hypotheses: the *acoustic model* and the *language model*.

The acoustic model defines the observation likelihood of particular acoustic features for a given phone<sup>6</sup>, while the language model defines the probability of a given sequence of words. This distinction rests on the formalisation of the speech recognition task as a *Hidden Markov Model* (HMM), where the states represent the sequence of phones, and the observations are the acoustic features.

For the practical development of spoken dialogue systems, the most important element of a speech recogniser is the language model. The language model effectively represents the set of utterances that can be accepted as inputs to the system (and their relative probabilities). The model can be encoded either in the form of a hand-crafted recognition grammar, or via statistical modelling based on a particular corpus of reference. In the latter case, the language model typically takes the form of an N-gram model, often a bi- or tri-gram corrected with appropriate smoothing and back-off techniques (??). It is also often beneficial to dynamically modify the language model at runtime to reflect the changing context and dialogue state. This real-time model adaptation can notably be realised by priming the words or expressions that are most contextually relevant (??).

The output of the speech recogniser is typically a N-best list (or recognition lattice) representing a set of possible hypotheses for the utterance, together with their relative confidence score or probabilities. Thus, the output of the speech recogniser is a list expressed as:

$$\tilde{u}_u = \langle (\tilde{u}_{u(1)}, p_1), (\tilde{u}_{u(2)}, p_2), \dots (\tilde{u}_{u(n)}, p_n) \rangle$$

where  $\tilde{u}_{u(i)}$  represents a specific recognition hypothesis and  $p_i$  its corresponding probability.<sup>7</sup>

### Natural language understanding

Once the recognition hypotheses for the utterance have been generated by the speech recogniser, the next task is to extract its semantic content. The goal of natural language understanding (NLU) is to

<sup>6</sup>A phone is an individual sound unit of speech. Technically speaking, acoustic models are not defined over entire phones but over sub-segments, typically decomposed into three parts: beginning, middle and end.

<sup>7</sup>In order to be proper probabilities, the usual axioms of probability theory  $0 \leq p_i \leq 1$  for all  $p_i$  and  $\sum_{i=1}^n p_i = 1$  must be satisfied. It should also be noted that in practice, many speech recognisers only provide raw confidence scores for their hypotheses. Estimating the exact correspondence between these scores and meaningful probabilities is a non-trivial task that has been investigated by e.g. ?).

build a representation of the meaning(s) expressed by the form of a given utterance. This task is a notoriously difficult endeavour, due to the combination of various factors. One major difficulty lies in speech recognition errors, with WER (Word Error Rates) often revolving around 20 % for many dialogue applications. The syntactic and semantic analysis of utterances is likewise complicated by the occurrence of sentential fragments, disfluencies of various sorts (e.g. filled pauses, repetitions, corrections) and ambiguities that must be resolved at multiple linguistic levels.

Natural language understanding can be decomposed in a number of steps. Parsing corresponds to the task of extracting the syntactic structure of the utterance and mapping it to a semantic representation. Spoken language parsing can be realised through various techniques, from keyword or concept spotting (??) to shallow semantic parsing (?), grammar-based parsing (?) and statistical parsing (?). It has been shown useful to apply upstream preprocessing techniques to correct speech recognition errors (?) and filter out disfluencies (?). In addition, referring expressions might also need to be resolved (?). Finally, the dialogue act associated with the utterance must be determined (??). ?) provides a survey of the various models and techniques used in the field of spoken language understanding.

Given speech recognition hypotheses  $\tilde{a}_u$  given as inputs, and possibly a representation of the dialogue history and external context, the task of natural language understanding is to extract a corresponding N-best list of dialogue act hypotheses  $\tilde{a}_u$  defined as:

$$\tilde{a}_u = \langle (\tilde{a}_{u(1)}, p_1), (\tilde{a}_{u(2)}, p_2), \dots (\tilde{a}_{u(n)}, p_n) \rangle$$

where  $\tilde{a}_{u(i)}$  represents a dialogue act hypothesis, usually represented in a logical form with various predicates and arguments, and  $p_i$  its corresponding probability.

## Dialogue management

Dialogue management occupies a central stage in spoken dialogue systems. As already mentioned in the introductory chapter, dialogue management serves a double role. The first task of the dialogue manager is to maintain a representation of the current dialogue state and update it as new information becomes available.<sup>8</sup> This dialogue state should encode every information that is of general relevance for the system, such as the recent dialogue history (encoded as a temporally ordered sequence of dialogue acts performed by the dialogue participants), the current conversational floor, the status of the task(s) to fulfil, and various features describing the context of the interaction. Furthermore, the dialogue state can also include information that is indirectly inferred from the individual observations. In particular, many dialogue domains include a variable that explicitly encode the hypothesised user intention. This user intention, although never directly observed, can often be derived from the user inputs through a sequence of reasoning steps. Similarly, the dialogue state can also integrate features that characterise the user profile and her/his preferences. Depending on the theoretical premises chosen for the system, the dialogue state can be either encoded as a fully observable data structure or represent partial observability through the definition of probability distributions on the values of the state variables.

---

<sup>8</sup>Some approaches explicitly distinguish between two types of management tasks: task management, responsible for monitoring and advancing the execution of the application objectives, and dialogue management *stricto sensu*, responsible for the more conversational aspects of the interaction. Establishing the boundary between the two types of tasks is however not always trivial.

The second task of dialogue management is to take decisions based on this dialogue state. This task is often called *action selection*. The dialogue manager is responsible for selecting the next action to perform by the system, which can be a communicative action (e.g. a piece of information to communicate, a question to task, a grounding signal to convey), an external action (e.g. a physical movement for a robot or a database manipulation for a booking system), a combination of the two, or no action at all. The action selection mechanism can take many forms, ranging from a direct mapping between states and actions to the application of logical rules or the use of offline or online planning techniques.

Dialogue management leads to two distinct outcomes: (1) an updated dialogue state that reflects the observations received as inputs (user dialogue acts, contextual changes etc.), and (2) a selected system action denoted as  $a_m$  (the  $m$  subscript standing for “machine” to distinguish it from the user act  $a_u$ ). As for the user dialogue act  $a_u$ , the system action  $a_m$  is often represented in a logical form composed of predicates and arguments.

Section 2.3 describes in more detail the various approaches and techniques that have been proposed in the literature to formalise the dialogue management process.

## **Generation**

Assuming the selected system action  $a_m$  relates to a verbal action, the following step is to find the best linguistic realisation for the abstract communicative goal defined in  $a_m$ . As for natural language understanding, a variety of generation techniques are available, from shallow generation strategies based on canned sentences or templates to more sophisticated approaches based on sentence planning and surface realisation (??). More recently, statistical methods have also been pursued in enhance the robustness and user-adaptivity of the generation algorithms (??).

The inputs of the generation module are the selected system action  $a_m$  and optionally the features defined in the dialogue state  $s$  (e.g. the user model and the external context). Given this information, the generation module will produce a corresponding user utterance denoted  $u_m$ . In the case of multimodal systems, the module may also deliver realisations for other modalities than the speech channel, such as gestures or facial expressions.

## **Speech synthesis**

The final step of the processing cycle is to synthesise the utterance in a speech waveform – a process called *text-to-speech synthesis*. This synthesis is performed in two consecutive stages. The utterance is first converted into a phonemic representation. This conversion involves multiple processing steps related to text normalisation, phonetic and prosodic analysis. Once the conversion is completed, the resulting phonemic representation is fed to a synthesiser in charge of producing the actual waveform. This synthesis can either be performed by gluing together pre-recorded units of speech from a speech database (concatenative synthesis) or by generating sounds using explicit acoustic models of the vocal tract (formant and articulatory synthesis). Most current dialogue systems rely on concatenative synthesis, and in particular unit selection (?).

### 2.2.3 Applications

Spoken dialogue systems have a wide variety of applications, ranging from academic research prototypes to mature commercial products. The first applications can be found in telephone-based systems for information access and service delivery. A large variety of systems have been developed in this area, for applications as diverse as automated call-routing (?), travel planning (?), weather updates (?), bus schedule retrieval (?) or tourist information (?). The recent emergence of smartphones also led to the development of new voice interfaces for multimodal local search (?), cross-lingual communication (?) and even pedestrian exploration (?). Many of these ideas have found their way into commercial products, as evidenced by the success of applications such as Apple's Siri, Nuance's Dragon Go! and Google Now.

Spoken dialogue systems can also be applied in domains where the use of touch interfaces and screens should be avoided because it is impractical or dangerous. This is notably the case for in-car navigation systems (??) where voice interfaces are to be preferred for safety reasons. The recent trends towards ubiquitous computing and “ambient” intelligence in smart home environments also offer promising applications of dialogue system technology (??).

Spoken dialogue systems are deployed in increasingly complex and open-ended interaction domains, where the artificial agent is no longer a mere executor of user commands, but increasingly plays the role of a collaborator or intelligent assistant. Conversational interfaces have notably developed in the healthcare sector to monitor – and hopefully improve – the health condition and fitness of patients through interactive dialogues (???). Substantial research has also been devoted into the development of interactive tutoring assistants in various learning contexts (????).

Finally, dialogue systems form an integral part of many robotic systems. Robots are deployed in increasingly social environments, such as homes, offices, schools and hospitals. There is therefore a growing need for robots endowed with communicative abilities. Human-robot interaction is an active area of research and has focused on aspects such as situated dialogue processing (??), adaptivity (?), symbol grounding (??) and multimodal interaction (???).

## 2.3 Dialogue management

Various approaches have been proposed to formalise the dialogue management problem. Common to virtually all approaches to dialogue management is (1) the representation of the agent's knowledge of the current situation in a data structure called the *dialogue state* and (2) the use of a decision mechanism to select the action to perform in each dialogue state. A wide range of strategies have been proposed to represent, update and act upon this dialogue state. We first describe hand-crafted approaches and then move on to the more recently developed statistical methods.

### 2.3.1 Hand-crafted approaches

#### Finite-state automata

The simplest approach to dialogue management is based on finite-state automata (FSA). A finite state automaton is defined by a collection of states and directed edges between them. Decision-making is made possible by associating each state with a specific action to execute at that state. Each edge in the automata is labelled with a condition on the user input that, if satisfied, will

move the current state from the source of the edge to its target. Figure 2.3 illustrates an example of finite-state automata for a simple, system-initiated interaction that takes user directions. If the user response is different from the five expected inputs, the system will ask the user to repeat until a admissible input is provided. The system will continue to request directions until the “stop” command is uttered, in which case a final state is reached.

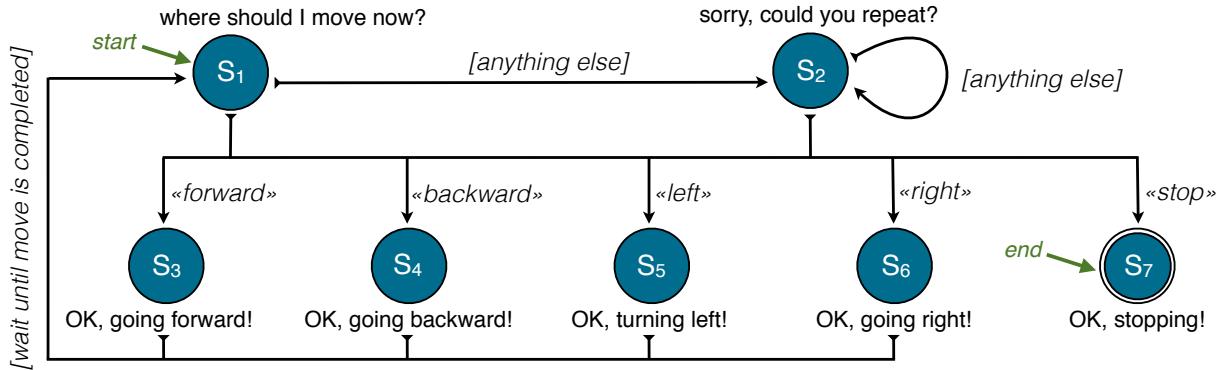


Figure 2.3: Example of finite-state automaton (FSA) for dialogue management, with seven possible states. The starting state for this FSA is  $s_1$  and the (in this case unique) ending state is  $s_7$ . The edges  $s_3, \dots, s_7 \rightarrow s_1$  are traversed once the movement is completed, and the two edges  $s_1, s_2 \rightarrow s_2$  are traversed for any other user input than the five specified directions.

A finite-state automaton is formally defined as a tuple  $\langle \mathcal{S}, \Sigma, \delta, s_0, \mathcal{F} \rangle$ , where  $\mathcal{S}$  is the set of possible states,  $\Sigma$  the set of inputs that the system can accept (in this case, the possible user inputs),  $\delta : \mathcal{S} \times \Sigma \rightarrow \mathcal{S}$  the transition function mapping every (state,input) pair to its successor state,  $s_0$  the start state, and  $\mathcal{F}$  the set of final states.

Finite-state automata provide a simple and versatile framework for the development of a dialogue manager. Their expressive power is however rather limited, as the dialogue state of a FSA is represented as a single, atomic symbol, and the possible user moves by a finite enumeration of possible transitions allowed for each state. Finite-state automata are therefore difficult to scale to more complex domains where the dialogue state might need to track multiple variables and allow for a large number of dialogue acts from the user.

### Logic- and plan-based approaches

Richer representations of the dialogue state are required to overcome the rigidity of finite-state automata. A popular alternative builds on representations that encode the dialogue state as a *frame* constituted of a set of slot-value pairs (?). Frame-based systems start with an empty frame that is gradually filled by the user inputs. After each user move, a set of production rules determine which actions to take – typically, a request to elicit a value for a particular slot – based on the current frame. The interaction continues until all slots in the frame are filled, which marks the completion of the dialogue.

Due to their greater expressivity, frame-based systems offer a number of advantages in terms of domain modelling and dialogue control. They remain however difficult to extend to other domains than classical slot-filling applications such as flight booking. The *information state* approach (?) is an attempt to provide a more solid theoretical foundation for dialogue management in rich

conversational domains. As already mentioned in Section 2.2.1, information state approaches rely on a blackboard architecture where various modules are attached to a central workspace called the information state. This information state is therefore continuously monitored by the modules integrated the dialogue system, and represent the full contextual knowledge available to the agent. In addition to the usual variables describing the dialogue history and the application task, the information state can also incorporate “mentalistic” entities such as the private and shared beliefs of the conversational agents. The information state can exhibit a rich internal structure encoded as attribute-value matrices (AVMs) or typed records (?).

Upon reception of a relevant input, the dialogue manager modifies this information state using a collection of update rules. In addition to state-internal operations that modify particular variables of the information state, the update rules are also employed to derive the actions to execute by the agent. Given a collection of rules and a generic strategy to apply them, the dialogue manager can both update its state and select the next action to perform by way of logical inference. This action selection can notably be grounded on the set of open questions raised and not yet answered during the interaction (??).

Plan-based approaches such as the ones developed by ?) and ?) take one step further. These approaches also rely on complex representations of the dialogue state that notably encompass the belief, desires and intentions (BDI) of each agent (??). In such settings, both the user and the system are assumed to act in pursuit of their long term goals. The interpretation of the user actions is cast as a *plan recognition* problem, where the system seeks to derive the belief, desires and intentions that best explain the observed conversational behaviour of the speaker. Similarly, the selection of system actions is derived from the (task-specific) long term objectives of the system. This search for the best action is an instance of a classical planning task, which can be solved using off-the-shelf planning algorithms. These algorithms require the declaration of a planning domain that specifies the preconditions and effects of every action. Agent-based frameworks such as the Constructive Dialogue Modelling approach developed by ?) follow similar lines, with a particular emphasis on conceptualising dialogue as a collaborative activity grounded in communicative principles of rational and coordinated interaction between agents.

### **Benefits and limitations of hand-crafted approaches**

The primary benefits of hand-crafted approaches to dialogue management lie in their ability to capture rich conversational phenomena and endow the system designer with a fine-grained control over the application behaviour. They have also laid the foundations for substantial advances in the semantic and pragmatic interpretation of dialogue moves (??), the formalisation of social obligations (?), the rhetorical structure of dialogue (?), or the use of plan-based reasoning to infer the user intentions (??). They nevertheless suffer from two important shortcomings:

1. They generally assume complete observability of the dialogue context and provide only a restricted account of uncertainties. This assumption of full observability is unfortunately difficult to reconcile with the technological limitations of speech recognition and understanding, and with the sometimes unpredictable nature of verbal interactions.
2. They require the dialogue domain to be specified by hand, either through the definition of an finite-state automaton, a collection of update rules or a set of action schemas for planning.

This requirement is hard to satisfy for many domains, since the behaviour of real users is often challenging to anticipate (unsurprisingly, human behaviour can be difficult to predict) and can deviate significantly from the expectations of the system developers.

Statistical approaches, to which we now turn, have been specifically developed to address these two issues.

### 2.3.2 Statistical approaches

Common to all statistical approaches to dialogue management is the idea of automatically optimising a dialogue policy (that is, a function associating each possible dialogue state to a system action) from interaction data. Starting from this shared premise, statistical approaches vary along multiple dimensions such as the type of learning algorithm, the representation of the dialogue state and policy, and the nature of the data on which to estimate the models. We outline in this section the core concepts of statistical approaches, which will be exposed in a more formal setting in the next chapter.

#### Supervised learning

The first possible approach is to learn dialogue strategies by imitation based on examples of expert behaviour. This expert behaviour can be recorded through so-called “Wizard-of-Oz” experiments. As already mentioned in the introduction chapter, a Wizard-of-Oz experiment is an interaction in which a human user is asked to interact with a system that is remotely operated by a human agent (without the user being made aware of this control). A hidden wizard is often preferred to a visible human interlocutor, as people tend to behave differently when they talk to a machine or a human person (?). One can collect multiple interactions of this type and record the wizard decisions at each point, along with their context. The resulting data set can be fed to a supervised learning algorithm in order to construct a dialogue policy that attempts to imitate the conversational behaviour of the wizard. Learning the dialogue policy is thus seen as a classification problem with states as inputs and actions as outputs. The goal of the learning algorithm is then to construct a classifier that optimises the classification accuracy for the Wizard-of-Oz data set, considering the wizard actions as “gold standards”. This classifier can be estimated with any standard machine learning methods (decision trees, logistic regression, etc.).

In a supervised setting, action selection is essentially viewed as a sequence of isolated decision problems. As argued by ?), this formalisation ignores some important characteristics of conversational behaviour, as dialogue is a dynamic process where the state and action at time  $t$  have a direct influence on the resulting state at time  $t + 1$ . This temporal connection between states is typically lost with classical supervised learning approaches. Furthermore, the state space grows exponentially with the number of state variables, and can therefore reach very large sizes. The training data available from a fixed Wizard-of-Oz corpus will therefore only cover a fraction of the state space for the domain. As a consequence, many states encountered at runtime will have no appropriate training examples on which to ground the action selection. Generalisation and abstraction techniques can however be used to mitigate this problem of data sparsity.

## Reinforcement learning

Reinforcement learning (RL) presents an attractive solution to the problem of dialogue policy optimisation. A reinforcement learning problem typically revolves around an *agent* interacting with its environment, typically to perform some practical task. Through its actions, the agent is able to change the state of its environment. After each action, the agent can observe both the new environment state resulting from its actions, as well as a numerical reward encoding the immediate value (positive or negative) of the executed action in relation to the agent's goal. The goal of the learning agent is to find the best action to execute in any given state via a process of trial and error – the best action being characterised as the one that maximise the agent's expected long-term reward.

Reinforcement learning tasks are generally formalised using *Markov Decision Processes* (MDPs), which are defined as tuples  $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$  with a state space  $\mathcal{S}$ , an action space  $\mathcal{A}$ , a transition function  $T$  that encodes the probability  $P(s' | s, a)$  of reaching state  $s'$  after executing action  $a$  in state  $s$ , and a reward function  $R$  that specifies the reward value associated with the execution of action  $a$  in state  $s$ . Dialogue can be expressed as a Markov Decision Process where the state space corresponds to the possible dialogue states and the actions to the set of (verbal or extra-verbal) actions available to the dialogue agent. The transition function  $T$  captures the “dynamics” of the conversation, and indicates how the dialogue state is expected to change as a result of the system actions. Finally, the reward function  $R$  expresses the objectives and costs of the application. A common reward function is to assign a high positive value for the successful completion of the task, a high negative value for a failure, and a small negative value for soliciting the user to repeat or clarify her/his intention.

Given a particular MDP problem, the goal of the learning agent is to find a policy  $\pi : \mathcal{S} \rightarrow \mathcal{A}$  that maps each possible state to the best action to execute at that state. The best action is defined as the action that maximises the *expected return* for the agent. Simply put, the return is the long-term (discounted) accumulation of rewards from the current state up to a given horizon.

Various learning methods have been devised to automatically extract this optimal policy from interaction experience. Due to the large amounts of cycles that are necessary to converge onto an optimal policy, direct interactions with real users are often impossible or highly impractical for many domains. Instead, most recent approaches have relied on the construction of a user simulator able to generate unlimited numbers of interactions on the basis of which the dialogue system can optimise its policy. The user simulator can either be designed by experts or “bootstrapped” from existing datasets or Wizard-of-Oz studies (??). The reliance on a user simulator for policy optimisation has the advantage of allowing the learning agent to explore millions of dialogue trajectories on a scale that would be impossible to achieve with real users. Simulated interactions run however the risk of deviating from real user behaviours.

A limitation faced by MDP approaches is the assumption that the dialogue state is fully observable. As frequently noted in the course of this thesis, this assumption simply does not hold for most dialogue domains, owing to the presence of multiple sources of uncertainty, in particular speech recognition errors. An elegant solution to this problem is to extend the MDP framework by allowing the state to be a hidden variable that is indirectly inferred from observations. Such extension gives rise to a *Partially Observable Markov Decision Process* (POMDP). POMDPs are formally defined as tuples  $\langle \mathcal{S}, \mathcal{A}, T, R, \mathcal{O}, Z \rangle$ . As in a classical MDP,  $\mathcal{S}$  represents the state space,  $\mathcal{A}$  the action space,  $T$  the transition probability  $P(s' | s, a)$  between states, and  $R$  the reward function

$R(s, a)$ . However, the actual state is no longer directly observable. Instead, the process is associated with an observation space  $\mathcal{O}$  that expresses the set of possible observations that can be perceived by the system (for instance, the N-best lists of user dialogue acts generated by the speech recogniser and NLU modules) . The function  $Z$  finally defines the probability  $P(o | s)$  of observing  $o$  in the current state  $s$ .

In the POMDP setting, the agent knowledge at a given time is represented by the *belief state*  $b$ , which is a probability distribution  $P(s)$  over possible states. The belief state is continuously updated as additional information becomes available in the form of e.g. new observations. Based on this belief state, a POMDP policy is defined as a function mapping each possible belief state to its optimal action. As for MDP-based reinforcement learning, POMDP approaches usually derive the dialogue policy from interactions with a user simulator (??). The optimisation process is however considerably more complex than for MDPs, as the belief state is a continuous and high-dimensional structure. Approximation techniques are therefore necessary in order to extract dialogue policies of reasonable quality in such complex space. The next chapter fleshes out the theoretical foundations of these modelling strategies and their applications to spoken dialogue systems.

### Benefits and limitations of statistical approaches

As stated in the previous sections, one key benefit of statistical approaches is the improved robustness towards errors and unexpected events. This robustness stems primarily from the use of probabilistic reasoning techniques that explicitly account for the uncertainty inherent in spoken dialogue. The second benefit is the possibility to optimise dialogue policies in a principled, data-driven manner based on a generic specification of the system objectives expressed in the reward function. This specification allows the system designer to explicitly encode the various goals and costs of the system. This ability to represent trade-offs between multiple, sometimes conflicting objectives is one important advantage of reinforcement learning approaches. Empirical studies have shown that automatically optimised policies can outperform hand-crafted strategies in both simulated environments and real user trials, based on both objective and subjective metrics of interaction success (??).

Statistical modelling techniques come however with a number of challenges of their own. The most pressing issue is the paucity of suitable data sets. Statistical models often require large amounts of training data to estimate their parameters. Unfortunately, real interaction data is scarce, expensive to acquire, and difficult to transfer from one domain to another. User simulators can partly alleviate this problem, but must often themselves be bootstrapped from data, and offer no guarantee of producing conversational behaviours that reflect those of real users. The computational complexity of the learning algorithm can also be problematic. Statistical approaches – and especially POMDP-based systems – must often carefully engineer their state and action variables to limit the size of the search space and ensure the learning process remains tractable. Albeit several dimensionality reduction techniques have been proposed in the literature (????), most work has so far concentrated on slot-filling applications. Application domains such as tutoring systems, cognitive assistants and human-robot interaction must however often deal with state-action spaces that are considerably more elaborate, with multiple tasks to perform, sophisticated user models, and a complex, dynamic environment. In such settings, the dialogue system might need to track a large number of variables in the course of the interaction, which quickly leads to a combinatorial explosion of the state space.

How to define appropriate statistical models for these dialogue domains remains an open question, to which the present thesis aims to offer preliminary answers.

Finally, many practical dialogue applications need to enforce generic constraints on the dialogue flow. Such constraints may for instance correspond to business rules specific to the particular application.<sup>9</sup> The incorporation of such constraints in the optimisation process of dialogue policies is however far from trivial. As noted by ??), this lack of direct control on the final policy is one of the main reasons for the slow adoption of reinforcement learning approaches in industrial applications. Although some researchers have worked on the integration of expert knowledge into dialogue policy optimisation (??), much work remains to be done to bring about a unified approach to dialogue management that combines the robustness of data-driven approaches with the control and expressivity of hand-crafted strategies.

Table 2.1 presents a comparison of the most important hand-crafted and statistical methods to dialogue management in terms of state representation, account of state uncertainty (in the sense of having multiple hypotheses about the current state, each one assigned with a specific probability), type of state update and action selection mechanism. The last row also describes how the approach developed in this thesis stands in comparison to these methods.

## 2.4 Summary

We have presented in this chapter the most important concepts and methods in the area of dialogue processing and management. Starting with a linguistic analysis of the most important dialogue phenomena, we discussed several key aspects of verbal interactions, such as their articulation in sequences of turns and dialogue acts. We also stressed the importance of contextual knowledge in the interpretation and production of dialogue acts, and the role of grounding signals to maintain mutual understanding among the conversational partners.

Section 2.2 described how spoken dialogue systems are internally structured. As we have explained, dialogue systems are often instantiated in complex software architectures that comprise numerous interconnected components for tasks such as speech recognition, understanding, dialogue management, natural language generation and speech synthesis. Dialogue systems can also be extended to handle (i.e. both perceive and act upon) extra-linguistic modalities and environmental factors. The range of possible applications of dialogue system technology is broad and includes domains as varied as mobile applications for information access and service delivery, in-car navigation systems, smart home environments, cognitive assistants, tutoring systems, and service robots.

The last section presented an overview of the dialogue management task. A key concept shared by virtually all approaches to dialogue management is the *dialogue state*, a data structure used to encode the system knowledge of the current conversational situation. This dialogue state can take multiple forms, from the atomic symbols used in finite-state approaches to the rich nested feature structures employed in information state formalisms. Based on this dialogue state, an action selection mechanism is then responsible for the selection of the next action to execute. In hand-crafted approaches, this mechanism is manually specified by the application developer, either via

---

<sup>9</sup>For instance, a flight booking system might be required by law to request the user to explicitly confirm his reservation at least once before printing any ticket.

<b>Approach</b>	<b>State representation</b>	<b>State uncertainty</b>	<b>State update mechanism</b>	<b>Action selection mechanism</b>
Finite state automata	Atomic state	no	Traversal of matching edge	Action associated with node
Frame-based systems (e.g. ??)	Slot/value pairs	no	Slot-filling given user inputs	Production rules
Information state update (e.g. ??)	Rich typed feature structure	no	Update rules	Decision rules
Plan-based systems	BDI model	no	Plan recognition and update of BDI model	Classical planning
Supervised approaches (e.g. ??)	Atomic/factored state	no	Extraction of state variables from history and task status	Classifier estimated from Wizard-of-Oz data by supervised learning
MDP-based systems (e.g. ??)	Atomic/factored state	no	Extraction of state variables from history and task status	Policy optimised via reinforcement learning from real or simulated dialogues
POMDP-based systems (e.g. ??)	Atomic/factored state	yes	Update of belief state	Policy optimised via reinforcement learning from real or simulated dialogues
Probabilistic rules	Factored state	yes	Structured belief state update (with probabilistic rules)	Policy optimised via (Bayesian) supervised or reinforcement learning

Table 2.1: Comparison of dialogue management approaches.

direct mappings from state to actions, or indirectly through the use of planning techniques. Statistical approaches, on the other hand, seek to automatically optimise dialogue policies from (real or simulated) interaction data. A wide range of learning techniques have been developed to perform this optimisation, from supervised learning on a Wizard-of-Oz data set to reinforcement learning based on a user simulator and a generic reward function. Reinforcement learning techniques can themselves be divided into MDP-based approaches, where action effects are stochastic but the dialogue state itself is assumed to be known, and POMDP-based approaches, which incorporate both stochastic action effects and state uncertainty.

We concluded our review of dialogue management paradigms by noting that both hand-crafted and statistical methods have significant challenges to overcome. This is especially striking for open-ended application domains such as human-robot interaction, as these domains often have a rich conversational context (including e.g. the dialogue history, the physical situation of the agent and the status of the tasks to perform) but are also associated with high amounts of noise, errors and unexpected events. One of the central claims of this thesis is that these domains are best addressed with a hybrid approach to dialogue management that combines probabilistic modelling with expert knowledge about the domain structure. Chapter 4 presents how such modelling approach can be formalised. But before doing so, we first need to lay down the mathematical apparatus required to construct and estimate probabilistic models of dialogue, which is the subject of the next chapter.



# Chapter 3

## Probabilistic models of dialogue

The previous chapter provided an overview of the most influential approaches to dialogue management, and outlined in particular the benefits of statistical techniques to account for the uncertainty and unpredictability inherent to spoken dialogue. The present chapter exposes the theoretical and methodological foundations of these statistical approaches as well as their application to the dialogue management task.

The first section of this chapter concentrates on the use of graphical models to design structured representations of probability and utility distributions. Graphical models provide mathematically principled methods for representing, estimating and reasoning over complex probabilistic domains. The section starts with the most well-known type of directed graphical model, namely Bayesian networks. We then show how to extend Bayesian networks to capture temporal sequence and express decision-theoretic problems through actions and utilities. We also review the most important inference and learning algorithms developed for these graphical models.

Based on these graphical models, the second section presents the fundamental principles of reinforcement learning, which is the learning framework employed by most statistical approaches to dialogue management. The section starts with a definition of Markov Decision Processes and explains how policies can be automatically optimised for such processes. The discussion is then extended to partially observable environments in which the current state is hidden and must be indirectly inferred from observations.

Once the mathematical foundations of graphical models and reinforcement learning are in place, the third and last section of this chapter describe how these concepts and techniques can be practically applied to model dialogue management tasks. The section provides a survey of the multiple approaches that have been developed in the last two decades to automatically optimise dialogue policies based on various flavours of supervised and reinforcement learning.

### 3.1 Graphical models

We describe in this section the core properties of (directed) graphical models,<sup>1</sup> their formal representation, and their use in learning and inference tasks.

---

<sup>1</sup>There also exists a variety of undirected graphical models, amongst which Markov networks, as well as partially directed models (e.g. Conditional Random Fields), but they will not be discussed nor employed in this thesis.

### 3.1.1 Representations

#### Bayesian networks

Let  $\mathbf{X} = X_1 \dots X_n$  denote a set of random variables, where each variable  $X_i$  is associated with a range of mutually exclusive values. This range can be either discrete or continuous. For dialogue models, the range of a variable  $X_i$  is typically discrete and can be explicitly enumerated. The enumeration of values for the variable  $X_i$  can be written  $Val(X_i) = \{x_i^1, \dots, x_i^m\}$ .

In the general case, the variables  $\mathbf{X}$  can be interrelated by complex probabilistic dependencies. These dependencies can be expressed through the joint probability distribution  $P(X_1 \dots X_n)$ . The size of this joint distribution is however exponential in the number  $n$  of variables, and is therefore difficult to manipulate (let alone estimate and reason over) directly.

It is fortunately possible to exploit conditional independence properties to reduce the complexity of the joint probability distribution. For three disjoint sets of random variables  $\mathbf{X}$ ,  $\mathbf{Y}$  and  $\mathbf{Z}$ , we say that  $\mathbf{X}$  and  $\mathbf{Y}$  are conditionally independent given  $\mathbf{Z}$  iff  $P(\mathbf{X}, \mathbf{Y} | \mathbf{Z}) = P(\mathbf{X} | \mathbf{Z})P(\mathbf{Y} | \mathbf{Z})$  for all possible combination of values for  $\mathbf{X}$ ,  $\mathbf{Y}$  and  $\mathbf{Z}$ . This conditional independence is denoted  $(\mathbf{X} \perp \mathbf{Y} | \mathbf{Z})$ .

Conditional independence allows a joint probability distribution to be decomposed into smaller distributions that are much easier to work with. For a variable  $X_i$  in  $X_1 \dots X_n$ , we can define the set  $parents(X_i)$  as the minimal set of predecessors of  $X_i$  such that the other predecessors of  $X_i$  are conditionally independent of  $X_i$  given  $parents(X_i)$ . The set of parents can be empty if the variable  $X_i$  is independent of all other variables. This definition enables us to decompose the joint distribution based on the chain rule:

$$P(X_1, \dots, X_n) = \prod_{i=1}^n P(X_i | X_1, \dots, X_{i-1}) \quad (3.1)$$

$$= \prod_{i=1}^n P(X_i | parents(X_i)) \quad (3.2)$$

This decomposition can be graphically represented in a *Bayesian network*. A Bayesian network is a directed acyclic graph (DAG) where each random variable is represented by a distinct node. These nodes are connected via directed edges that reflect conditional dependencies. In other words, an edge  $X_m \rightarrow X_n$  indicates that  $X_m \in parents(X_n)$ . Each variable  $X_i$  in the Bayesian network must be associated with a specific conditional probability distribution  $P(X_i | parents(X_i))$ . Together with the directed graph, the conditional probability distributions (abbreviated as CPDs) fully determine the joint probability distribution of the Bayesian network.

Given such definitions, the Bayesian network can be directly used for inference by querying the distribution of a subset of variables, often given some additional evidence. Two operations are especially useful when manipulating probability distributions:

- Marginalisation (also called “summing out”), which derives the probability of the variable  $X$  given its conditional distribution  $P(X | Y)$  and the distribution  $P(Y)$ :

$$P(X) = \sum_{y \in Val(Y)} P(X, Y) = \sum_{y \in Val(Y)} P(X | Y)P(Y) \quad (3.3)$$

- Bayes' rule, which reverses the order of a conditional distribution between two variables  $X$  and  $Y$  (possibly with some background evidence  $e$ ):

$$P(X | Y, e) = \frac{P(Y | X, e)P(X | e)}{P(Y | e)} \quad (3.4)$$

As an illustration, Figure 3.1 provides an example of Bayesian network that models the probability of occurrence of a fire at a given time. The probability of this event is dependent on the current weather. In addition, two (imperfect) monitoring systems are used to detect possible fires; one on the ground, and one via satellite.

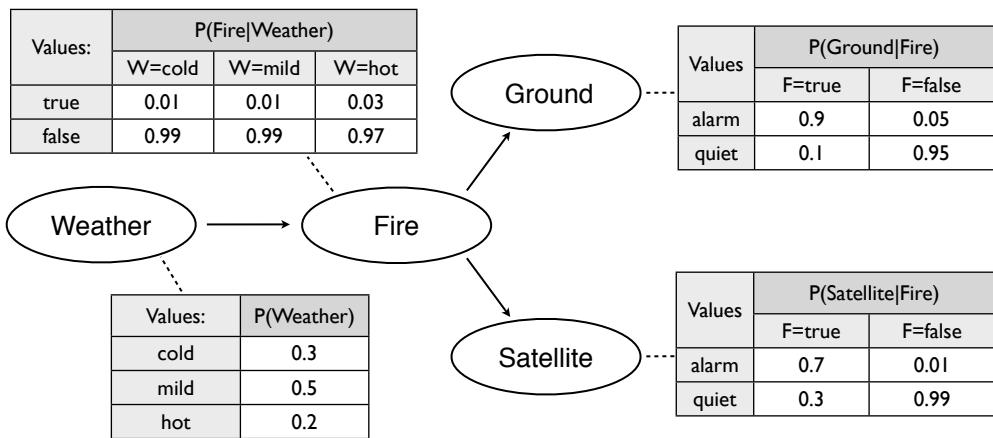


Figure 3.1: Example of Bayesian network with four random variables.

There is one distinct distribution for every combination of values in  $\text{parents}(X_i)$ . The probabilistic model defined in the figure includes therefore a total of eight distributions. The distributions in Figure 3.1 are *categorical* distributions.<sup>2</sup> Categorical distributions can be encoded with look-up tables that map every possible value in  $\text{Val}(X_i)$  to a particular probability. Many of the probability distributions used throughout this thesis will take the form of categorical distributions. Other representations for discrete CPDs are however conceivable, as will be shown later in this thesis.

A Bayesian network can also contain continuous probability distributions. These distributions are usually encoded with *density functions* represented in a parametric form. A well-known example of parametric distribution is the normal distribution  $\mathcal{N}(\mu, \sigma^2)$ , which is defined by its two parameters  $\mu$  (the mean) and  $\sigma^2$  (the variance). Continuous distributions can alternatively expressed with non-parametric methods based on e.g. kernel estimators. The reader is invited to consult ?) and ?) for more detail on these distributions.

Appendix A enumerates the most important discrete and continuous probability distributions employed in this thesis.

<sup>2</sup>Categorical distributions are often conflated with *multinomial* distributions, which specify the number of times an event will occur in a repeated independent trial with  $k$  exclusive categories, with each category having a fixed probability. A categorical distribution is equivalent to a multinomial distribution for a single observation.

## Reasoning over time

In order to apply Bayesian networks to tasks such as dialogue management, two additional elements are necessary. The first extension is to allow variables to evolve as a function of time. Such temporal dependencies are indeed necessary to account for the dynamic nature of dialogue (the dialogue state is not a static entity but is expected to change over time). Two assumptions are usually made to structure such temporal dependencies:

1. The first assumption, called the Markov assumption, is that the variable values at time  $t$  only depend on the previous time slice  $t - 1$ . Formally, let  $\mathbf{X}$  be an arbitrary collection of variables. We denote by  $\mathbf{X}_t$  the random variables that express their values at time  $t$ . The Markov assumption states that  $(\mathbf{X}_t \perp \mathbf{X}_{0:(t-2)} | \mathbf{X}_{t-1})$ .
2. The second assumption is that the process is stationary<sup>3</sup> – that is, that the probability  $P(\mathbf{X}_t | \mathbf{X}_{t-1})$  is the same for all values of  $t$ .

Given these two assumptions, we can define a stochastic process with a probability distribution  $P(\mathbf{X}_t | \mathbf{X}_{t-1})$  that specifies the distribution of the variables  $\mathbf{X}$  at time  $t$  given their values at time  $t - 1$ . Such model is called a *dynamic Bayesian network* (DBN). The distribution  $P(\mathbf{X}_t | \mathbf{X}_{t-1})$  can be internally factored and include dependencies both between the time slices  $t - 1$  and  $t$  and within the slice  $t$ . Figure 3.2 shows a concrete example of dynamic Bayesian network. The DBN provides a factored representation of the distribution  $P(R_t, F_t, G_t | R_{t-1}, F_{t-1})$ .

Given the specification of the distribution  $P(\mathbf{X}_t | \mathbf{X}_{t-1})$  and an initial distribution  $P(\mathbf{X}_0)$ , a dynamic Bayesian network can be “unrolled” onto multiple time slices. This unrolled model corresponds to a classical Bayesian network.

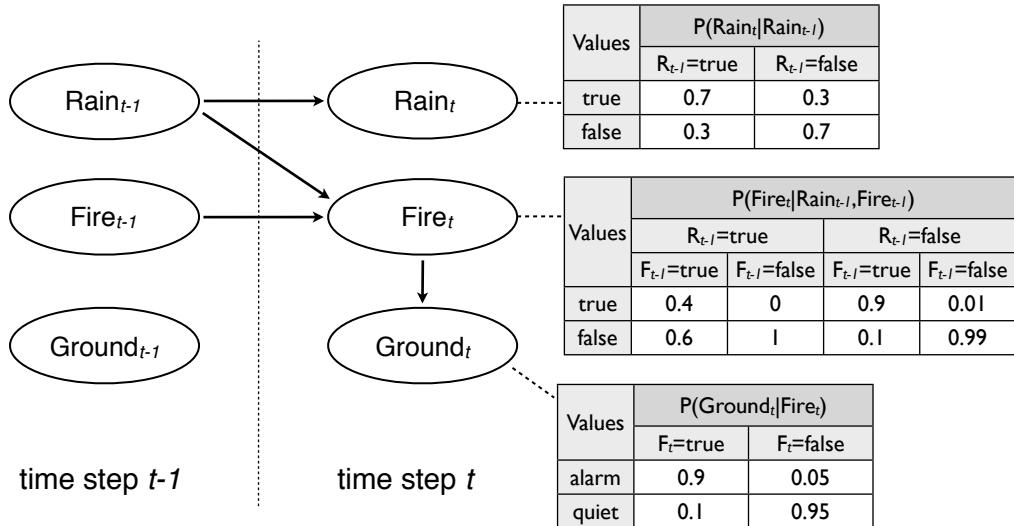


Figure 3.2: Example of dynamic Bayesian network.

<sup>3</sup>A *stationary* process must be distinguished from a *static* process: a static process is a stochastic process that remains constant for all time steps. In contrast, a stationary process can change over time, but the transition model that describes the dynamics of this process remains constant.

## Decision problems

Dynamic Bayesian networks are well-suited to represent temporal processes. However, in sequential decision tasks such as dialogue management, tracking the current state is only the first step of the reasoning process. The agent must also be able to calculate the relative utilities of the various actions that can be executed at that state. The second extension of Bayesian networks thus pertains to the inclusion of actions and utilities in addition to state variables.

*Decision networks*<sup>4</sup> are Bayesian networks augmented with a representation of action variables and their corresponding utilities. Decision networks may include three classes of nodes:

1. *Chance nodes* correspond to the classical random variables described so far. Chance nodes are associated with conditional probability distributions that define the relative probabilities of the node values given the values in the parent nodes.
2. *Decision nodes* (sometimes also called action nodes) correspond to variables that are under the control of the system. The values of these nodes reflect an active choice made by the system to execute particular actions.
3. *Utility nodes* express the utilities (from the system's point of view) associated with particular situations expressed in the node parents. Typically, these parents combine both chance and decision variables. Utility nodes are coupled with utility distributions that associate each combination of values in the node parents with a specific (negative or positive) utility.

Decision networks combined with temporal dependencies are called *dynamic decision networks*. Figure 3.3 illustrates a dynamic decision network with a decision variable containing two alternative actions assigned to distinct utility values depending on the occurrence of fire.

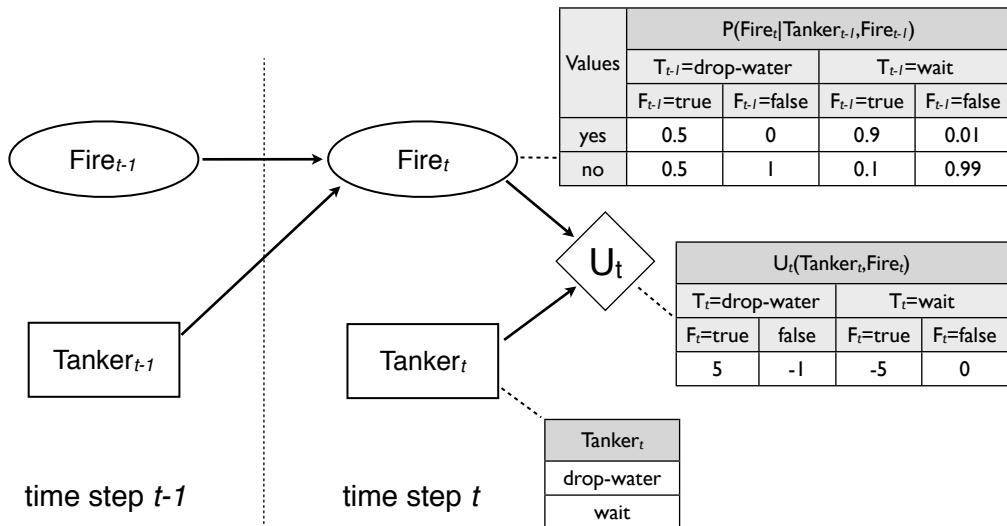


Figure 3.3: Example of dynamic decision network. By convention, chance nodes are represented with circles, decision nodes with squares, and utility nodes with diamonds.

<sup>4</sup>Decision networks are also called *influence diagrams*.

### 3.1.2 Inference

#### Generalities

The main purpose of probabilistic graphical models is to evaluate *queries* – that is, calculate a posterior distribution over a subset of variables, usually given some evidence. Assuming a graphical model defining the joint probability distribution of a set of variables  $\mathbf{X}$ , a probability query is a posterior distribution of the form  $P(\mathbf{Q} | \mathbf{E} = \mathbf{e})$ , where  $\mathbf{Q} \subset \mathbf{X}$  denotes the query variables,  $\mathbf{E} \subset \mathbf{X}$  the evidence variables, and  $\mathbf{e}$  a possible assignment of values for the evidence variables. If the set of evidence variables is empty, the query is reduced to the calculation of a marginal distribution. Graphical models augmented with decision and utility variables can also be used to answer utility queries of the form  $U(\mathbf{Q} | \mathbf{E} = \mathbf{e})$ . In this case, the query variables often correspond to decision nodes whose utility is to be estimated.

A wide range of inference algorithms have been developed to efficiently evaluate these probability and utility queries. These algorithms can be either exact or approximate.

Exact algorithms calculate the precise posterior distribution corresponding to the query through a sequence of manipulation operations on the CPDs contained in the graphical model. One popular algorithm for exact inference is variable elimination (?). Variable elimination relies on dynamic programming techniques to evaluate a query through a sequence of matrix operations (summation and pointwise product). These operations are defined on so-called “factors” that represent CPDs in a matrix format. Variable elimination can be generalised to handle utility queries using joint factors (?). Other algorithms such as message passing on clique trees can also be used (?).

Exact inference remains unfortunately difficult to scale to large, densely interconnected graphical models, and approximate techniques are often unavoidable in many practical domains. Algorithms for approximate inference in graphical models include approaches such as loopy belief propagation (?), variational methods (?), and a wide array of sampling techniques (?), sometimes also called Monte Carlo methods. Popular sampling techniques include various flavours of importance sampling (??) and Markov Chain Monte Carlo (MCMC) approaches such as Gibbs sampling (??).

Probabilistic inference is a difficult computational task. In fact, inference on unconstrained Bayesian Networks is known to be #P-hard, which is a complexity class that is strictly harder than NP-complete problems. This holds both for exact inference (?), and – perhaps more surprisingly – also for approximate inference (?).

The openDial toolkit we have developed for this thesis includes two inference algorithms: generalised variable elimination (?; p. 1103) and a specific type of importance sampling algorithm called likelihood weighting, which we outline below. These algorithms are used in the processing workflow of the dialogue manager to (1) update the dialogue state upon the reception of new observations and (2) select system actions on the basis of this updated dialogue state. The details of this workflow will be provided in Section 4.4.

#### Likelihood weighting

To make our discussion of inference algorithms for graphical models more concrete, we describe below a simple but efficient sampling method called *likelihood weighting* (?), which we have used in most of the experiments conducted in this thesis.

The key principle behind all types of sampling algorithms is to estimate the posterior distribution expressed in the query by collecting a large quantity of samples (i.e. assignments of values to the random variables) drawn from the graphical model. Likelihood weighting proceeds by sampling the random variables in the graphical model one by one, in topological order (i.e. from parents to children).<sup>5</sup> For instance, sampling the network in Figure 3.1 will start with the variable *Weather*, then *Fire* (based on the value drawn for the parent *Weather*), and finally *Ground* and *Satellite* (based on the value drawn for *Fire*). In order to account for the evidence  $\mathbf{e}$ , every sample is associated with a specific *weight* that expresses the likelihood of the evidence given the assignment for all the other variables. For graphical models that include utility variables, samples also record the total utility accumulated for the sampled values. The pseudocode in Algorithm 3.1.2 outlines the sampling procedure.

---

**Algorithm 1 : GET-SAMPLE ( $\mathcal{B}, \mathbf{e}$ )**


---

**Input:** Bayesian/decision network  $\mathcal{B}$  over  $\mathbf{X}$  with topological ordering  $X_1, \dots, X_n$

**Input:** Evidence  $\mathbf{E} = \mathbf{e}$

**Output:** Full sample drawn from  $\mathcal{B}$  together with a weight and utility

```

Initialise sample  $\mathbf{x} \leftarrow \langle \mathbf{e} \rangle$ 
Initialise weight  $w \leftarrow 1$  and utility  $u \leftarrow 0$ 
for all  $X_i \in X_1, \dots, X_n$  do
    if  $X_i$  is a chance or decision variable and  $X_i \notin \mathbf{E}$  then
         $x_i \leftarrow$  sample value drawn from  $P(X_i | \mathbf{x})$ 
         $\mathbf{x} \leftarrow \mathbf{x} \cup \langle x_i \rangle$ 
    else if  $X_i$  is a chance variable and  $X_i \in \mathbf{E}$  then
         $w \leftarrow w \times P(X_i = \mathbf{e}(X_i) | \mathbf{x})$ 
    else if  $X_i$  is a utility variable then
         $u \leftarrow u + U_i(\mathbf{x})$ 
    end if
end for
return  $\mathbf{x}, w, u$ 

```

---

The notation  $\mathbf{e}(X_i)$  refers to the value specified for the variable  $X_i$  in the assignment  $\mathbf{e}$ .

A large number of samples can be accumulated in such manner. Once enough samples are collected (or the inference engine has run out of time) the resulting posterior distribution for the query variables  $\mathbf{Q}$  is derived by normalising the weighted counts associated with each value of the query variables, as shown in Algorithm 3.1.2. Algorithm 3.1.2 extends the procedure to the calculation of utility distributions. In this case, the utility values are not normalised but correspond to a weighted average of the sampled utilities.

### 3.1.3 Learning

We have so far pushed aside the question of how the distributions in the graphical model are exactly estimated. Early approaches often relied on distributions elicited from human experts based on plausible associations they have observed. Although useful in domains where no data is available, hand-crafted models are unfortunately difficult to scale (only models with a limited number of

---

<sup>5</sup>A partial order on the nodes in the graph can always be found since the network is a directed acyclic graph.

---

**Algorithm 2** : LIKELIHOOD-WEIGHTING ( $\mathcal{B}, \mathbf{Q}, \mathbf{e}, N$ )

---

**Input:** Bayesian/decision network  $\mathcal{B}$  over  $\mathbf{X}$  with topological ordering  $X_1, \dots, X_n$

**Input:** Set of query variables  $\mathbf{Q}$  and evidence  $\mathbf{E} = \mathbf{e}$

**Input:** Number  $N$  of samples to draw

**Output:** Approximate posterior distribution  $P(\mathbf{Q} | \mathbf{e})$  given  $\mathcal{B}$

Let  $\mathbf{W}$  be vectors of weighted counts for each possible value of  $\mathbf{Q}$ , initialised to zero

**for**  $i = 1 \rightarrow N$  **do**

$\mathbf{x}, w \leftarrow \text{GET-SAMPLE}(\mathcal{B}, \mathbf{e})$

$\mathbf{W}[\mathbf{x}(\mathbf{Q})] \leftarrow \mathbf{W}[\mathbf{x}(\mathbf{Q})] + w$

**end for**

Normalise the weighted counts in  $\mathbf{W}$

**return**  $\mathbf{W}$

---

---

**Algorithm 3** : LIKELIHOOD-WEIGHTING-UTILITY ( $\mathcal{B}, \mathbf{Q}, \mathbf{e}, N$ )

---

**Input:** (see above)

**Output:** Approximate utility distribution  $U(\mathbf{Q}, \mathbf{e})$  given  $\mathcal{B}$

Let  $\mathbf{W}, \mathbf{U}$  be vectors of weighted counts for each possible value of  $\mathbf{Q}$ , initialised to zero

**for**  $i = 1 \rightarrow N$  **do**

$\mathbf{x}, w, u \leftarrow \text{GET-SAMPLE}(\mathcal{B}, \mathbf{e})$

$\mathbf{W}[\mathbf{x}(\mathbf{Q})] \leftarrow \mathbf{W}[\mathbf{x}(\mathbf{Q})] + w$

$\mathbf{U}[\mathbf{x}(\mathbf{Q})] \leftarrow \mathbf{U}[\mathbf{x}(\mathbf{Q})] + w \times u$

**end for**

Average the weighted utility counts  $\mathbf{U}(\mathbf{q}) \leftarrow \frac{\mathbf{U}(\mathbf{q})}{\mathbf{W}(\mathbf{q})} \quad \forall \text{ values } \mathbf{q} \text{ of } \mathbf{Q}$

**return**  $\mathbf{U}$

---

parameters can be elicited in such manner), and are vulnerable to human errors and inaccuracies. It is therefore often preferable to automatically estimate these distributions from real world data – in other words, via statistical estimation based on a collection of examples in a training set.

Two distinct types of learning tasks can be distinguished:

1. The most common task is *parameter estimation*. Parameter estimation assumes the general structure of the graphical model (i.e. the dependencies between variables) is known, but not the parameters of the individual CPDs. Most discrete and continuous distributions are indeed “parametrised” – that is, they depend on the specification of particular parameters that define the exact shape of the distribution. A categorical distribution on  $k$  values has for instance  $k$  parameters that assign the relative probability of each outcome. Similarly, a normal distribution  $\mathcal{N}(\mu, \sigma^2)$  is governed by its two parameters  $\mu$  and  $\sigma^2$ .
2. The second possible learning task is *structure learning*. In structure learning, the agent must learn both the structure (i.e. the directed edges) and the parameters of the graphical model, given only the list of variables and the training data. This task is significantly more complex than parameter estimation, as the agent must simultaneously find which variables influence one another and estimate their corresponding conditional dependencies.

We shall concentrate in this thesis on the parameter estimation problem, which is by far the

most common type of learning problem in dialogue management.<sup>6</sup>

### Maximum likelihood estimation

The most straightforward parameter estimation method is maximum likelihood estimation (MLE). Maximum likelihood estimation searches for the parameters values that provide the best “fit” for the provided data set. In other words, the parameters are set to the values that maximise the likelihood of the observed data. Given a data set  $\mathcal{D}$ , a graphical model and a set of parameters  $\theta$  to estimate in this model, the MLE learning objective is to find the values  $\theta^*$  that maximise the probability  $P(\mathcal{D} ; \theta)$ , which is the likelihood of the data set  $\mathcal{D}$  given the specified parameter values for  $\theta$ . This likelihood is often written in logarithmic form:

$$\theta^* = \operatorname{argmax}_{\theta} P(\mathcal{D} ; \theta) = \operatorname{argmax}_{\theta} \log P(\mathcal{D} ; \theta) \quad (3.5)$$

If the data samples cover the complete set of variables in the model, this likelihood can be neatly decomposed in a set of local likelihoods, one for each CPD, and the  $\theta^*$  values can be derived in closed-form. For a categorical distribution, the maximum likelihood estimates will simply correspond to the relative counts of occurrences in the training data.

The learning problem becomes more complex for partially observed domains in which the data samples contain hidden variables. For the Bayesian network in Figure 3.1, an example of partially observed sample is  $\langle \text{Weather} = \text{mild}, \text{Ground} = \text{alarm}, \text{Satellite} = \text{quiet} \rangle$ , where the occurrence of fire is not specified. In such cases, the likelihood function is no longer decomposable, which implies that the maximum likelihood estimate is not amenable to a closed-form solution. The only alternative is to resort to iterative optimisation methods such as gradient ascent (?) and Expectation Maximisation (?).

The main drawback of maximum likelihood estimation is its vulnerability to overfitting when learning from small data sets. For instance, if we only had collected one single data point  $\text{Weather} = \text{cold}$  for the previous example, the maximum likelihood estimate for the distribution of  $P(\text{Weather})$  would be  $\langle 1, 0, 0 \rangle$ . In other words, maximum likelihood estimation does not take into account any prior knowledge about the relative probability of particular parameter hypotheses, which often lead to unreasonable estimates for low frequency events.

### Bayesian learning

An alternative to maximum likelihood estimation is *Bayesian learning*. The key idea of Bayesian approaches to parameter estimation is to view the CPD parameters as random variables and to derive their posterior distributions after observing the data. Bayesian learning starts with an initial prior over the range of parameter values and gradually refines this distribution through probabilistic inference based on the observation of the samples in the training data. Each distribution  $P(X_i | \text{parents}(X_i))$  with unknown parameters is therefore associated with a parent node  $\theta_{X_i|\text{parents}(X_i)}$  that define its parameter distribution.

Parameter distributions are typically continuous (since probabilities are continuous values), and

---

<sup>6</sup>As shall be argued in the next chapters, the dependency relations between state variables can often be derived from prior knowledge of the dialogue domain.

often multivariate. Intuitively, we can think of the variable  $\theta_{X_i|parents(X_i)}$  as expressing a “distribution over possible distributions”.

Based on this formalisation, parameter estimation can be elegantly reduced to a problem of probabilistic inference over the parameters. Given a prior  $P(\boldsymbol{\theta})$  on the parameter values and a data set  $\mathcal{D}$ , the posterior distribution  $P(\boldsymbol{\theta} | \mathcal{D})$  is given by Bayes’ rule:

$$P(\boldsymbol{\theta} | \mathcal{D}) = \frac{P(\mathcal{D}; \boldsymbol{\theta}) P(\boldsymbol{\theta})}{P(\mathcal{D})} \quad (3.6)$$

Equation 3.6 allows us to calculate the posterior distribution  $P(\boldsymbol{\theta} | \mathcal{D})$  through the use of standard inference algorithms for graphical models.

It is often convenient to encode the distributions of the parameter variables as *conjugate priors* of their associated CPD distribution. In such case, the prior  $P(\boldsymbol{\theta})$  and posterior  $P(\boldsymbol{\theta} | \mathcal{D})$  after observing the data points  $\mathcal{D}$  are ensured to remain within the same distribution family. In particular, if the distribution of interest is a categorical distribution, its parameter distribution can be encoded with a Dirichlet distribution, which is known as the conjugate prior of categorical and multinomial distributions. A Dirichlet distribution is a continuous, multivariate distribution of dimension  $k$  (with  $k$  being the size of the multinomial) that is itself parametrised with so-called concentration hyperparameters denoted  $\boldsymbol{\alpha} = [\alpha_1, \dots, \alpha_k]$ . Additional details about the formal properties of Dirichlet distributions can be found in Appendix A.

Figure 3.4 illustrates this Bayesian approach to parameter estimation for the variable *Weather*. As the variable possesses three alternative values, the allowed values for the parameter  $\boldsymbol{\theta}_{\text{Weather}}$  are three-dimensional vectors  $[\theta_{\text{Weather}(1)}, \theta_{\text{Weather}(2)}, \theta_{\text{Weather}(3)}]$ , with the standard constraints on probability values:  $\theta_{\text{Weather}(i)} \geq 0$  for  $i = \{1, 2, 3\}$  and  $\theta_{\text{Weather}(1)} + \theta_{\text{Weather}(2)} + \theta_{\text{Weather}(3)} = 1$ . As we can observe in the figure, these constraints effectively limit the range of possible values to a 2-dimensional simplex. The  $\boldsymbol{\alpha}$  hyperparameters can be intuitively interpreted as “virtual counts” of the number of observations in each category. In Figure 3.4, we can see that the hyperparameters [5, 10, 5] lead to higher probability densities for parameters around the peak  $\langle 0.25, 0.5, 0.25 \rangle$ . As the number of observations increases, the Dirichlet distribution will gradually concentrate on a particular region of the parameter space until convergence.

In the case of completely observed data, Bayesian learning over several parameters can be decomposed into independent estimation problems (one for each parameter variable):

$$P(\mathcal{D}; \boldsymbol{\theta}) = \prod_{\theta_i \in \boldsymbol{\theta}} P(\mathcal{D}; \theta_i) \quad (3.7)$$

As in the maximum likelihood estimation case, the learning task becomes more complicated when dealing with partially observed data, as the posterior distribution can no longer be represented as a product of independent posteriors over each parameter. In this setting, the full posterior is often too complex to be amenable to an analytic solution. Sampling techniques can however be applied to offer reasonable approximations of this posterior. As we shall see in Chapter 5 and 6, the work presented in this thesis is directly grounded in such approximate Bayesian learning methods.

Table 3.1 briefly summarises the parameter estimation methods discussed in this section.

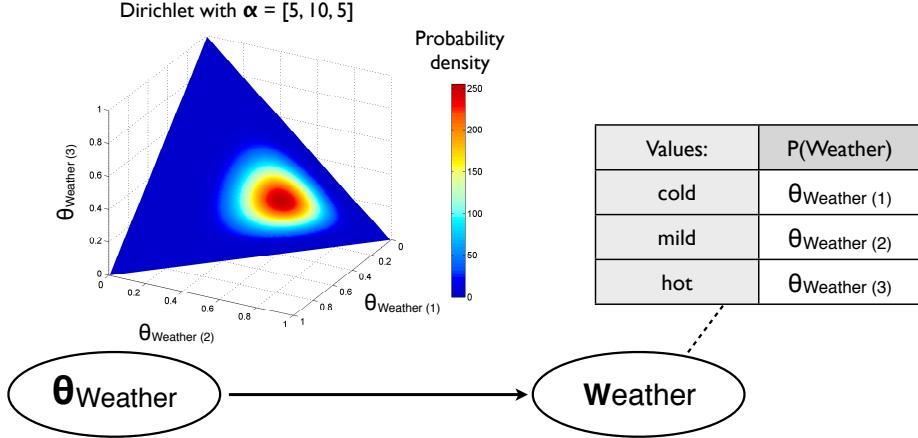


Figure 3.4: Bayesian network with variable *Weather* and associated parameter  $\theta_{\text{Weather}}$ . As *Weather* is a categorical distribution, the distribution  $P(\theta_{\text{Weather}})$  is expressed as a Dirichlet with three dimensions that reflect the relative probabilities for the three values in  $\text{Val}(\text{Weather})$ .

Training data	Maximum likelihood estimation	Bayesian learning
<i>Fully observed</i>	Maximisation of local likelihood functions	Query on local posterior distribution over each parameter
<i>Partially observed (hidden variables)</i>	Iterative optimisation of global likelihood function	Query on full posterior over parameters via sampling

Table 3.1: Summary of parameter estimation approaches for directed graphical models.

## 3.2 Reinforcement learning

Dialogue management is fundamentally a problem of sequential decision-making under uncertainty. The objective of the dialogue system is to select the action that is expected to be “optimal” – that is, that yields the maximum long-term utility for the agent. However, in many domains (including dialogue domains), the agent has no knowledge of the internal dynamics of the environment it finds itself in. It must therefore determine the best action to execute in any given state via a process of trial and error. Such learning process is called *reinforcement learning* (RL). It is worth noting that reinforcement learning effectively strikes a middle ground between supervised and unsupervised learning. Contrary to supervised learning, the framework does not require the provision of “gold standard” examples. However, thanks to the reward function it receives from the environment, the agent is able to get a sense of the quality of its own decisions, an element which is absent in unsupervised learning methods.

We provide here a brief introduction to the central concepts in reinforcement learning, and refer the interested reader to [?\)](#) for more details.

### 3.2.1 Markov Decision Processes

Reinforcement learning tasks are typically based on the definition of a *Markov Decision Process* (MDP), which is a tuple  $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$  where:

- $\mathcal{S}$  is the state space of the domain and represents the set of all (mutually exclusive) states.
- $\mathcal{A}$  is the action space and represents the possible actions that can be executed by the agent.
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$  is the transition function and encodes the probability  $P(s'|s, a)$  of reaching state  $s'$  after executing action  $a$  in state  $s$ .
- $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  is the reward value associated with the execution of action  $a$  in state  $s$ .

Markov Decision Processes can be explicitly represented as dynamic decision networks. As we can see from the graphical illustration in Figure 3.5, the state at time  $t + 1$  is dependent both on the previous state at time  $t$  and the action  $a_t$  performed by the system. After each action, the system received a reward  $r_t = R(s_t, a_t)$  that depends both on the state and selected action.

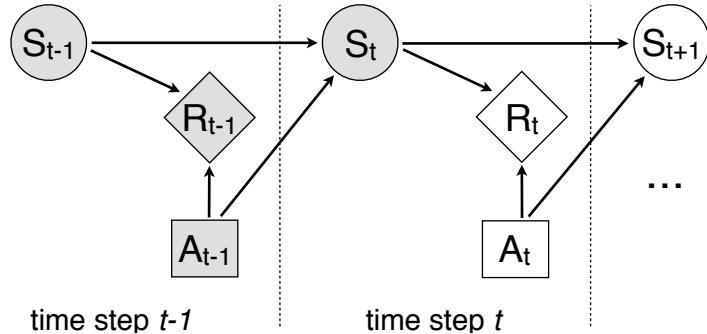


Figure 3.5: Graphical model of a Markov Decision Process (MDP) unfolded on a few time steps. Greyed entities indicate observed variables. In the MDP case, all past states, actions and rewards are observed, as well as the current state.

Given a particular MDP problem, the goal of the learning agent is to find an optimal policy  $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$  that maps each possible state to the best action to execute at that state. The best action is the action that maximises the *expected return* for that state, which is the discounted sum of rewards, starting from the current state up to a potentially infinite horizon. In this sum, a geometric discount factor  $\gamma$  (with  $0 \leq \gamma \leq 1$ ) indicates the relative worth of future rewards in regard to present ones. At one extreme,  $\gamma = 0$  corresponds to a short-sighted agent only interested in its immediate reward, while  $\gamma = 1$  corresponds to an agent for which immediate and distant rewards are valued equally. For a given policy  $\pi$ , the expected return for an arbitrary state  $s$  in  $\mathcal{S}$  is expressed through the value function  $V^\pi(s)$ :

$$V^\pi(s) = E \left\{ r_0 + \gamma r_1 + \gamma^2 r_2 + \dots \mid s_0 = s, \pi \text{ is followed} \right\} \quad (3.8)$$

$$= E \left\{ \sum_{i=0}^{\infty} \gamma^i r_i \mid s_0 = s, \pi \text{ is followed} \right\} \quad (3.9)$$

where  $r_i = R(s_i, \pi(s_i))$  is the reward received at time  $i$  after performing the action specified by the policy  $\pi$  in state  $s_i$ . Equation (3.9) can be rewritten in a recursive form, leading to the well-known Bellman equation (?):

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) V^\pi(s') \quad (3.10)$$

In other words, the expected return in state  $s$  equals its immediate reward plus the expected return of its successor state. The recursive definition offered by the Bellman equation is crucial for many reinforcement learning methods, since it allows the value function to be estimated by an iterative process in which the value function is gradually refined until convergence.

Another useful concept is the action-value function  $Q^\pi(s, a)$  that expresses the return expected after performing action  $a$  in state  $s$  and following the policy  $\pi$  afterwards:

$$Q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^\pi(s') \quad (3.11)$$

The value and action-value functions are closely related, as  $V^\pi(s) = Q^\pi(s, \pi(s))$ .

The objective of the learning process is to find a policy  $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$  which selects the action with highest expected return for all possible states:  $V^{\pi^*}(s) \geq V^\pi(s)$  for any state  $s$  and policy  $\pi$ . For a given MDP, there is at least one policy that satisfies this constraint. The value and action-value functions for this optimal policy are respectively denoted  $V^*$  and  $Q^*$ .

Two families of reinforcement learning approaches can be used to determine this policy:

1. *Model-based* approaches first estimate an explicit model of the MDP and subsequently optimise a policy based on this model. The agent initially collects data by interacting with its environment and recording the reward and successor state following each action. Based on this data, standard parameter estimation methods (as outlined in Section 3.1.3) can be used to fix the MDP parameters, and the resulting model is applied to extract the corresponding optimal policy via dynamic programming (?) or more advanced Bayesian methods (?). The most well-known model-based learning algorithm is *value iteration*, which operates by estimating the value function via a sequence of updates based on Bellman's equation. Given a value function estimate  $V_k$  available at step  $k$ , the estimate at step  $k + 1$  is calculated as:

$$V_{k+1}(s) = \max_a R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) V_k(s') \quad (3.12)$$

Once the iterations have converged, the optimal policy is straightforwardly derived as:

$$\pi^*(s) = \operatorname{argmax}_a Q^*(s, a) \quad (3.13)$$

2. *Model-free* approaches skip the estimation of the underlying MDP model in order to directly learn  $Q^*$  functions from the agent's interaction experience. The most popular model-free techniques are Q-learning (?), SARSA (?) and gradient descent (?). The main idea behind model-free methods is to let the agent try out different actions, observe the effects in terms of rewards and successor state, and use this information to refine the estimate of the  $Q^*$

function. This operation is repeated for a large number of episodes until convergence.

One key question that must be addressed in both model-based and model-free approaches is how to efficiently explore the space of possible actions. The agent should indeed favour the selection of high-utility actions in most cases (since they are the ones of interest to the agent), but should also occasionally explore actions that are currently thought to be less effective to avoid being stuck in a suboptimal behaviour. This trade-off, called the *exploration-exploitation* dilemma, is one of the central research questions in reinforcement learning.

### 3.2.2 Partially Observable Markov Decision Processes

A limitation faced by MDP approaches is the assumption that the dialogue state is fully observable. As we already noted in the previous chapter, this assumption does not hold for most dialogue systems, owing to the presence of multiple sources of uncertainty, due in particular to speech recognition errors. An elegant solution to this problem is to extend the MDP framework by allowing the state to be a hidden variable that is indirectly inferred from observations. Such extension gives rise to a *Partially Observable Markov Decision Process* (POMDP). POMDPs are formally defined as tuples  $\langle \mathcal{S}, \mathcal{A}, T, R, \mathcal{O}, Z \rangle$ . As in a classical MDP,  $\mathcal{S}$  represents the state space,  $\mathcal{A}$  the action space,  $T$  the transition probability  $P(s'|s, a)$  between states, and  $R$  the reward function  $R(s, a)$ . However, the actual state is no longer directly observable. Instead, the process is associated with an observation space  $\mathcal{O}$  that expresses the set of possible observations that can be perceived by the system. The function  $Z$  then defines the probability  $P(o|s)$  of observing  $o$  in the current state  $s$ . Figure 3.6 provides a graphical illustration of the POMDP framework. As we can see, POMDPs can also be expressed as dynamic decision networks in which the state variable is not directly observed.

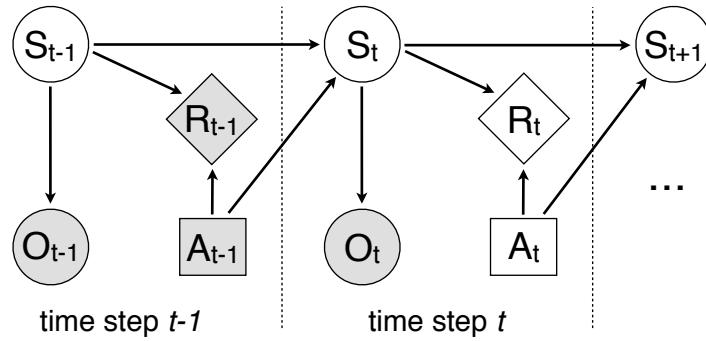


Figure 3.6: Graphical model of a Partially Observable Markov Decision Process (POMDP) unfolded on a few time steps. Compared to Figure 3.5, we notice that the state is no longer directly accessible but must be inferred from the observations and (predicted) state history.

The agent knowledge at a given time is represented by the *belief state*  $b$ , which is a probability distribution  $P(s)$  over possible states. After each system action  $a$  and subsequent observation  $o$ , the belief state  $b$  is updated to  $b'$  in order to incorporate the new information. This belief update is

a simple application of Bayes' theorem:

$$b'(s) = P(s'|b, a, o) = \frac{P(o|s') P(s'|b, a)}{P(o|b, a)} \quad (3.14)$$

$$= \eta P(o|s') \sum_s P(s'|s, a) b(s) \quad (3.15)$$

where  $\eta = P(o|b, a)$  serves as a normalisation constant and is usually never calculated explicitly.

In the POMDP setting, a policy is a function  $\pi = \mathcal{B} \rightarrow \mathcal{A}$  mapping each possible belief state to its optimal action. Mathematically, the belief state space  $\mathcal{B}$  is a  $(|\mathcal{S}| - 1)$ -dimensional simplex (where  $|\mathcal{S}|$  is the size of the state space), which is a continuous and high-dimensional space. The optimisation of the dialogue policy is therefore considerably more complex than for MDPs. The value function  $V^\pi$  for a policy  $\pi$  is the fixed point of Bellman's equation:

$$V^\pi(b) = \sum_{s \in \mathcal{S}} R(s, a)b(s) + \gamma \sum_{o \in \mathcal{O}} P(o|b, \pi(b)) V^\pi(b') \quad (3.16)$$

where  $b'$  is the updated belief state following the execution of action  $\pi(b)$  and the observation of  $o$ , as in Equation (3.15). The optimal value function  $V^*(b)$  for finite-horizon problems is known to be piecewise linear and convex in belief space, as proved by <sup>7</sup>). The value function can therefore be represented by a finite set of vectors, called  $\alpha$ -vectors. Each vector  $\alpha_i$  is associated with a specific action  $a(i) \in \mathcal{A}$ .<sup>7</sup> The vectors are of size  $|\mathcal{S}|$  and  $\alpha_i(s)$  is a scalar value representing the value of action  $a(i)$  in state  $s$ . Given these vectors, the value function simplifies to:

$$V^*(b) = \max_i \alpha_i \cdot b \quad (3.17)$$

And the policy  $\pi^*$  can be rewritten as:

$$\pi^*(b) = a \left( \operatorname{argmax}_i (\alpha_i \cdot b) \right) \quad (3.18)$$

Extracting the  $\alpha$ -vectors associated with a POMDP problem is however computationally challenging given the high-dimensional and continuous nature of the belief space, and exact solutions are intractable beyond toy domains. As shown by <sup>7</sup>), deriving the  $\alpha$ -vectors for a given POMDP – also known as “solving” the POMDP – is a PSPACE-complete problem, which means that the best known algorithms will take time  $2^{\text{poly}(n,h)}$  to solve a problem with  $n$  states and a planning horizon  $h$ .

A number of approximate solutions have however been developed. One simple strategy is to rewrite the  $Q(b, a)$  function in terms of the  $Q$ -values for the underlying MDP, as described by <sup>7</sup>):

$$Q(b, a) = \sum_{s \in \mathcal{S}} Q_{MDP}(s, a)b(s) \quad (3.19)$$

Although this approximation can work well in some settings, it essentially rests on the assumption that the state uncertainty will disappear after one action. It is therefore a poor model for

---

<sup>7</sup>Note that the reverse is not true: each action can be associated with an arbitrary number of vectors.

information-gathering actions – that is, actions that do not change the actual state but might help in reducing state uncertainty.<sup>8</sup>

Many approximations methods focus on simplifying the belief state space, notably through the use of grid-based approximations (?). The idea is to estimate the value function only at particular points within the belief simplex. At runtime, the action-value function for the current belief state  $b$  is then approximated to the value for the closest point in the grid according to some distance measure. Instead of using a fixed grid, most recent solution methods rely on sampling techniques to perform local value updates at specific belief points (??). Belief compression has also proved to be useful (?). Yet another family of optimisation methods directly search in the space of possible policies constrained to a particular form such as finite-state controllers (?).

A final alternative, which we follow in Chapter 6 of this thesis, is to rely on online planning algorithms (??). The idea is to let the agent estimate the  $Q(b, a)$  values at execution time via look-ahead search on a limited planning horizon. Compared to offline policies, the major advantage of online approaches is that the agent only needs to consider the current state to plan instead of enumerating all possible ones. It can also more easily adapt to changes in the reward or transition models, as the policy is not compiled in advance but generated at runtime. Online planning can moreover be used to simultaneously learn or refine these models during the interaction, as demonstrated by (?). The available planning time is however more limited, since planning is in this case interleaved with system execution and must therefore meet real-time constraints.

Despite these recent advances, the optimisation of POMDP policies remains nevertheless to a large extent an open research question in the fields of reinforcement learning and decision-theoretic planning. One important insight that transpires in much of the POMDP literature is the importance of exploiting the problem structure to reduce the complexity of the learning and planning problems (??). As detailed in the next chapter, the work presented in this thesis precisely attempts to transfer this insight into dialogue management.

### 3.2.3 Factored representations

In the previous pages, we modelled the system states and actions as atomic symbols. Such plain representations can unfortunately quickly lead to a combinatorial explosion of the state-action spaces. A more efficient alternative is to apply *factored* representations that decompose the state into distinct variables with possible conditional dependencies between one another. Similarly, action variables can also be split into distinct variables. For a MDP, the state will take the form of a set of variables  $S_t$ , and the transition function  $P(S_t | S_{t-1}, A_{t-1})$  be represented as a dynamic Bayesian network (?). The reward function can also be encoded as a collection of utility variables  $R_t$  connected to relevant sets of state and action variables. In such case, the total utility is defined as the sum of all utilities.

POMDPs can be factored in a similar way, with the inclusion of observation variables  $O_t$  connected to the state variables  $S_t$  through conditional dependencies  $P(O_t | S_t)$ . At time  $t$ , the observation variables  $O_t$  will be observed while the state variables  $S_t$  remain hidden. Figure 3.7 illustrates this factorisation. Both plain and factored (PO)MDPs form specific cases of dynamic decision networks, as illustrated in the figure.

---

<sup>8</sup>A typical example of such action in dialogue is a clarification request about the user intention.

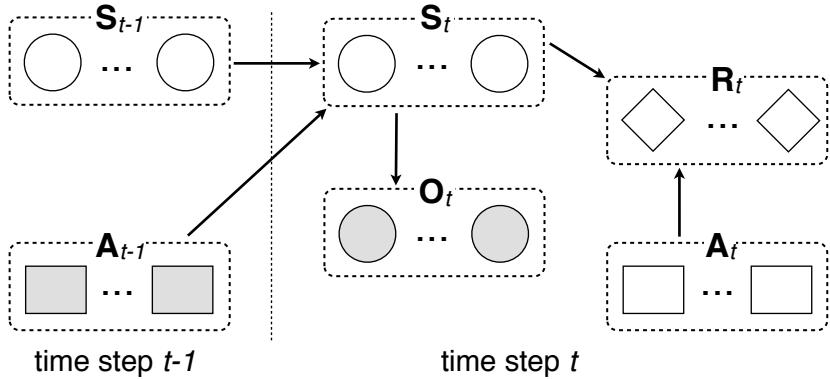


Figure 3.7: Factored representation of a POMDP with state variables  $\mathbf{S}$ , action variables  $\mathbf{A}$ , observations variables  $\mathbf{O}$ , and reward variables  $\mathbf{R}$ .

### 3.3 Application to dialogue management

We have now reviewed the key ideas of probabilistic reasoning and reinforcement learning, and are ready to explain how these ideas can be practically transferred to the dialogue management task. After a brief review of supervised learning approaches, we detail how dialogue can be modelled as a Markov Decision Process or Partially Observable Markov Decision Process, and survey the various optimisation techniques that have been developed to automatically optimise dialogue policies for such models based on real or simulated interaction data.

#### 3.3.1 Supervised learning from Wizard-of-Oz data

The most straightforward use of statistical approaches to dialogue management is to learn dialogue policies in a supervised learning manner, based on collected Wizard-of-Oz data where dialogue management is remotely performed by a human expert. The resulting training data is a sequence  $\{(s_i, a_i) : 1 \leq i \leq n\}$  of state-action pairs, where  $s_i$  is the state at time  $i$  and  $a_i$  the corresponding wizard action, which is assumed to reflect the best action to perform in this state. Most supervised learning approaches encode the dialogue state as a list of feature-value pairs, and the goal of the learning algorithm is to train a classifier  $C : \mathcal{S} \rightarrow \mathcal{A}$  from state to actions that produces the best fit for this training data (modulo regularisation constraints), and will therefore “mimic” the decisions of the wizard in similar situations. Various classifiers can be used for this purpose, such as maximum likelihood classification (?), decision trees (?), Naive Bayes (?), and logistic regression (??).

One important issue in supervised learning approaches is data sparsity, as only a fraction of the possible states can realistically be covered by the Wizard-of-Oz interactions. Several generalisation techniques can be employed to alleviate this problem. The simplest is to encode in a parametric form in which the parameters are associated with a smaller number of features extracted from the state-action pair. The size of the feature set can be further reduced with feature selection (?). Yet another approach put forward by ?) is to couple the classifier with a distance measure between states, thereby allowing the reuse of strategies learned from closely related states.

### 3.3.2 MDP-based optimisation of dialogue policies

Instead of learning a dialogue policy by imitating the behaviour of human experts, the dialogue manager can also learn by itself the best action to perform in each possible conversational situation via repeated interactions with a (real or simulated) user. Dialogue management can indeed be cast as a type of Markov Decision Process  $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$  in which:

- The state space  $\mathcal{S}$  corresponds to the set of possible dialogue states, usually encoded as a list of feature-value pairs that capture relevant aspects of the current conversational context.
- The actions space  $\mathcal{A}$  corresponds to the set of (verbal or extra-verbal) actions that can be executed by the system.
- The transition function  $T$  captures the “dynamics” of the conversation, and indicates how the dialogue state is expected to change as a result of the system actions (and in particular how the user is expected to respond).
- The reward function  $R$  expresses the objectives and costs of the application. A common reward function is to assign a high positive value for the successful completion of the task, a high negative value for a failure, and a small negative value for soliciting the user to repeat or clarify her/his intention.

A common misunderstanding should be clarified at this point. The representation of dialogue as a Markov Decision Process implies that the transition function only considers the previous state and action to predict the next state. It has occasionally been argued in the literature that this formalisation makes the decision-making process oblivious to non-local aspects of the dialogue history. This argument is however invalid, as the dialogue state is not limited to the mere representation of the last dialogue act but may express any features related to the interaction context, including variables recording dialogue histories of arbitrary length, long-term user intentions, and so forth. It is ultimately up to the system designer to decide which features are deemed relevant to describe the current state of the dialogue.

#### Dialogue state representation

As for supervised learning methods, MDP-based reinforcement learning approaches to dialogue management encode the dialogue state in terms of feature-value pairs. The dialogue state is thus factored into a number of independent variables (one for each feature). Most early approaches adopted crude state representations with features limited to the status of the slots to fill and the last user utterance (??). The voice-enabled email client described in ?) captured additional measures related to the overall task progress, history of previous system attempts, confidence thresholds and timing information. There has also been some work on the automatic identification of relevant state variables, using methods from structure learning in decision networks (?) and feature selection (?).

?) are to our knowledge the first to explore the extension of reinforcement learning methods to large state spaces based on rich representations of the conversational context. Inspired by information state approaches to dialogue management, their state space captures detailed information such as complete history of dialogue acts and fine-grained representations of the task status, amounting

to a total of  $10^{386}$  possible states. Such rich state representations allow the dialogue manager to exploit much broader contextual knowledge in its decision-making. However, it also creates important challenges regarding action selection, as generalisation techniques are necessary to scale up the learning procedure to such large state spaces.'

### Policy optimisation (and associated generalisation techniques)

For most dialogue domains, the reward is fixed in advance by the system designer and reflects the task objectives. It can also correspond to a performance metric such as PARADISE (?). The metric is defined in such case as a linear combination of quantitative measures whose weight is empirically estimated via multivariate linear regression from surveys of user satisfaction.<sup>9</sup>

The transition probabilities are however typically unknown. As described in Section 3.2.1, two families of approaches can be followed to optimise dialogue policies: model-based approaches first learn an explicit model of the MDP from interaction data and then extract a policy for this model, while model-free approaches directly estimate an action-value function  $Q^*(s, a)$  from interactions, without explicit model of the state transitions.

While it is possible to follow a model-based strategy and learn explicit distributions for the transition probabilities (?), the majority of recent RL approaches to dialogue management have adopted model-free techniques. Due to the significant amounts of data necessary to reach convergence, it is often impossible to directly learn the value function from interactions with real users for most practical domains. A user simulator is instead used to provide unlimited supplies of interactions to the reinforcement learning algorithm. Several options are available to construct this user simulator. The first option is to design it manually based on specific assumptions about the user behaviour (?). It can also be constructed in a data-driven manner from existing corpora (?).<sup>10</sup> In such case, user simulation can be interpreted as a way to expand the initial data set. The third available option is to exploit Wizard-of-Oz studies to tune the simulator parameters (?). In addition to user modelling, the simulator should also integrate error modelling techniques in order to simulate various types of errors that may appear along the speech recognition and understanding pipeline (?).

The key benefit of user simulation lies in the possibility to explore a large number of possible dialogue trajectories and error types. Such simulation is in particular of great use for prototyping dialogue policies and experimenting with alternative setups. However, simulated interactions are not as valuable as real interactions, and offer no guarantee of matching the behaviour and error patterns of actual users. The practice of evaluating the performance of dialogue policies based on the same simulator as the one used for training has also been put into question (?)

A key problem in reinforcement learning methods is the *curse of dimensionality*: the number of parameters grows exponentially with the size of the state and action spaces (?). In order to scale the learning procedure to larger and more complex domains, factorisation and generalisation techniques are often necessary. An early example of this line of research is the work of (?) on the use of graphical

---

<sup>9</sup>Quantitative measures of dialogue performance are typically classified in three categories: *task success* (e.g. ratio of successfully completed vs. failed tasks,  $\kappa$  agreement for slot-filling applications), *dialogue quality* measures (e.g. number of repetitions, barge-ins, ASR rejection prompts) and *dialogue efficiency* measures (e.g. number of utterances per dialogue, total elapsed time).

<sup>10</sup>The most important corpus of human-machine dialogue is the COMMUNICATOR dataset of telephone conversations in the domain of travel planning, with more than 180,000 utterances (90 hours of recording) transcribed and annotated with various understanding and dialogue-level information (?).

models in dialogue policy optimisation. The dialogue domain used in their experiments was a speech-enabled web browser. Their strategy was to explicitly represent the dialogue management task as a dynamic decision network and learn both the structure and parameters of this network from user simulations. The state space initially included all features that could be automatically logged from the interactions. Based on the simulated dialogues, the learning algorithm was able to automatically discover the subset of state variables that were relevant for decision-making as well as the transition probabilities between these variables. They also experimented with various Markov orders to analyse the impact of longer state histories on the system performance. After the estimation of the decision network, dynamic programming techniques were used to extract a dialogue policy that is optimal with respect to the learned models. Given the complexity of the optimisation, the dynamic programming solution was approximated via forward sampling (?).

?) demonstrated how to reduce the complexity of model-free learning techniques via function approximation. As the large size of their state-action space prevented the use of classical tabular representations for the  $Q(s, a)$  function, they instead relied on function approximation to define the  $Q$ -values as a linear function of state features:

$$Q(s, a) = f(s)^T w_a \quad (3.20)$$

where  $f(s)$  is a feature vector extracted from the state and  $w_a$  a weight vector for action  $a$ . The weight vectors were learned with the SARSA algorithm based on a fixed corpus (thereby avoiding the use of user simulators). To further refine the estimation of  $Q$ -values, they also combined estimates from both supervised and reinforcement learning in their final model.

Hierarchical abstraction is another way to reduce the search space of the optimisation process, as shown by ?). Instead of viewing action selection as a single monolithic policy, the selection is decomposed in their work into multiple levels, each responsible for a specific decision problem. This decomposition is formally expressed via an hierarchical extension of MDPs called Semi-Markov Decision Processes. Each level in the hierarchy is associated with its own subset of state and action variables. This modular approach allows a complex policy to be split into a sequence of simpler decisions. Such hierarchical formalisation is particularly natural for task-oriented dialogues, which are known to exhibit rich attentional and intentional structures, as notably argued in the seminal work of ?). It also bears similarities with the approach presented by ?) on dialogue understanding based on a plan structure.

Finally, it is worth noting that several researchers have attempted to combine the benefits of supervised and reinforcement learning methods by initialising a RL algorithm with a policy estimated via supervised methods (?).

### 3.3.3 POMDP-based optimisation of dialogue policies

To capture the uncertainty associated with many state variables, dialogue can be explicitly modelled as a Partially Observable Markov Decision Process  $\langle \mathcal{S}, \mathcal{A}, T, R, \mathcal{O}, Z \rangle$ . Modelling a dialogue domain as a POMDP is similar in most respects to the MDP formalisation. The observations in  $\mathcal{O}$  typically correspond to the possible N-best lists that can be generated by the speech recogniser and NLU modules, and can also include observations perceived via the other modalities.

## Dialogue state representation

POMDP approaches express state uncertainty through the definition of a belief state  $b$ , which is a probability distribution  $P(s)$  over possible states. After a system action  $a$  in belief state  $b$  followed by observation  $o$ , the belief state  $b$  is updated according to Equation (3.15), repeated here for convenience:

$$b' = P(s'|b, a, o) = \eta P(o|s') \sum_s P(s'|s, a) b(s) \quad (3.15)$$

Belief update requires the specification of two probabilistic models: the observation model  $P(o|s')$  and the transition model  $P(s'|s, a)$ . Many POMDP approaches to dialogue management factor the state  $s$  into (at least) three distinct variables  $s = \langle a_u, i_u, c \rangle$ , where  $a_u$  is the last user dialogue act,  $i_u$  the current user intention(s), and  $c$  the interaction context.<sup>11</sup> Assuming that the observation  $o$  only depends on the last user act  $a_u$ , and that  $a_u$  depends on both the user intention  $i_u$  and the last system action  $a_m$ , Equation (3.15) is then rewritten as:

$$b' = P(a'_u, i'_u, c' | b, a_m, o) \quad (3.21)$$

$$= \eta P(o | a'_u) P(a'_u | i'_u, a_m) \sum_{i_u, c'} P(i'_u | i_u, a_m, c') P(c') b(i_u) \quad (3.22)$$

The transition model is decomposed in this factorisation into two distinct distributions:

1. The distribution  $P(a'_u | i'_u, a_m)$  is called the *user action model* and defines the probability of a particular user action given her/his underlying intention and the last system act. It expresses the likelihood of the user action  $a'_u$  following the system action  $a_m$  and the compatibility of  $a'_u$  with the user intention  $i'_u$  (?).
2. The distribution  $P(i'_u | i_u, a_m, c')$  is the *user goal model* and captures how the user intention is likely to change as a result of the context and system actions.

These two distributions are usually derived from collected interaction data. A graphical illustration of this state factorisation is shown in Figure 3.8.

The observation model  $P(o | a'_u)$  is often rewritten as  $P(\tilde{a}_u)$ , the dialogue act probability in the N-best list provided by the speech recognition and semantic parsing modules (cf. Section 2.2.2), based on the following approximation:

$$P(o | a'_u) = \frac{P(a'_u | o) P(o)}{P(a'_u)} \approx P(a'_u | o) = P(\tilde{a}_u) \quad (3.23)$$

The approximation rests on the assumption of uniform distribution for the user dialogue acts and observation in the absence of further evidence. Since the probabilities  $P(\tilde{a}_u)$  are provided at run-time by the ASR and NLU modules, the approximation does not require an explicit specification of the observation set  $\mathcal{O}$  and observation model  $Z$  at design time. This modelling choice has the advantage of circumventing the statistical estimation of the observation model, a difficult problem since the number of possible N-best lists is theoretically infinite. A few researchers have however

---

<sup>11</sup>Some approaches also define a specific variable for the dialogue history (?).

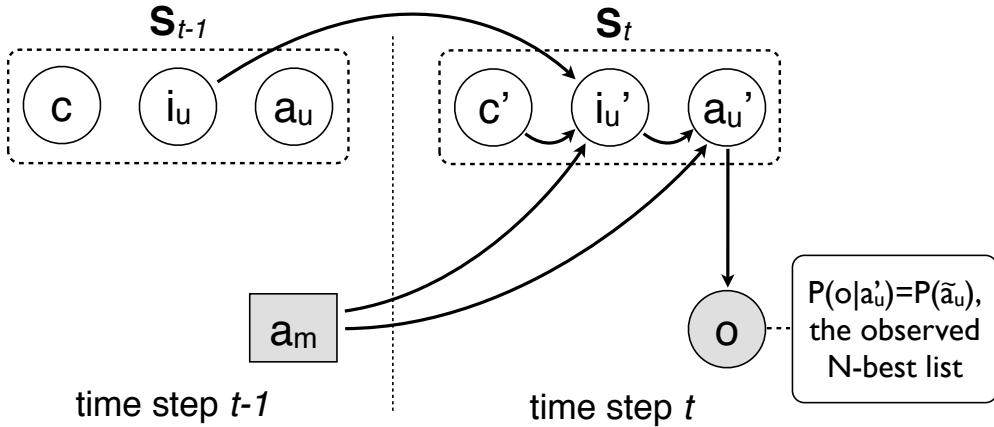


Figure 3.8: Common factorisation of the state space for a POMDP-based dialogue system, where  $c$  represents the dialogue context,  $i_u$  the user intention(s),  $a_u$  the user dialogue act,  $a_m$  the system act, and  $o$  the observed N-best list hypotheses. The representation omits the conditional dependencies and probability distribution for the variable  $c'$ , which are contingent on the particular interaction context in place for the domain.

attempted to estimate explicit observation models for small dialogue domains. ??) investigated how to integrate observations that include continuous-valued ASR confidence scores into a classical POMDP framework and devised ad-hoc density functions for this purpose. ??) showed how an observation model could be empirically estimated from interaction data with a bag-of-words approach.

In the seminal work of ??) that first introduced the POMDP framework to dialogue management, the state is represented by a single variable expressing the user intention, and hand-crafted models were used for the belief update. ??) extended the previous approach by introducing a factored state representation based on Bayesian Networks, where the state includes both the user intention and the system state. ??) further refined this factorisation by decomposing the dialogue state into three distinct variables that respectively represent the last user dialogue act, the user intention and the dialogue history. The related work of ??) used Bayesian networks to encode fine-grained dependencies between the various slots expressed in the user intention. ??) added to this factorisation a specific variable for the user's affective state. Substantial work has also been devoted to the inclusion of non-verbal observations and environmental factors into the dialogue state. In the human-robot interaction domain, ??) have notably applied Bayesian networks for inferring the underlying user intention based on observations arising from both verbal and non-verbal sources.

## Policy optimisation

POMDP solutions methods can in theory be applied to extract the dialogue policy from any given model specification, as shown by ??). Such strategy is however only suitable for relatively small action-state spaces and require the specification of an explicit observation model to extract the  $\alpha$ -vectors corresponding to the optimal policy (?). Most recent POMDP approaches have instead focused on the use of reinforcement learning to derive a dialogue policy from interactions with a user simulator (???).

The need for effective generalisation techniques is heightened in reinforcement learning for POMDPs, as dialogue policies defined in partially observable environments must be defined in a high-dimensional, continuous belief state space. Approximating the  $Q^*(b, a)$  function is thus crucial. One useful approximation method is to reduce the full belief state to a simpler representation such as the “summary state” described by [?](#)). In addition, the estimation of the action-value function can be further simplified by relying on techniques such as grid-based discretisations [?](#)) and linear function approximation [??](#)). Finally, non-parametric methods based on Gaussian Processes have recently been proposed [?](#)).

Reinforcement learning is considerably more difficult for POMDPs than for MDPs due to the partial observability of the state. [?](#)) showed how model-based reinforcement learning can be cast in a Bayesian framework. The key idea of Bayesian reinforcement learning is to maintain an explicit probability distribution over the POMDP parameters. This distribution is then gradually refined as more data is observed. This refinement is done using Bayesian inference, starting with an initial prior. At runtime, the parameter distribution is applied to plan the optimal action, taking into account every source of uncertainty – that is, state uncertainty, stochastic action effects, and uncertainty over the parameters. Our own work on model-based reinforcement learning also follows that line of work, as shall be explained in Chapter 6.

Most POMDP-based approaches to dialogue management assume that the reward model can be encoded in advance by the system designer, with a few notable exceptions. [?](#)) describe a Bayesian approach to estimate a reward model based on gold standard actions provided by an oracle. [??](#)) present alternative approaches based on inverse reinforcement learning (IRL) for POMDPs. Their main idea was to exploit Wizard-of-Oz data for a voice-enabled intelligent wheelchair to automatically infer a reward model. This task of inferring a reward model from expert demonstrations is a prototypical instance of *inverse reinforcement learning*: the agent observes how an expert performs the task and must find the hidden reward model that best explains this behaviour. Inverse reinforcement learning in partially observable domains is however difficult to scale beyond small domains due to the complexity of the optimisation problem [?](#)).

## 3.4 Summary

We have exposed down in this chapter the foundations of probabilistic modelling applied to dialogue, and have reviewed a range of theoretical concepts related to graphical models, reinforcement learning in both fully and partially observable domains, as well as the practical exploitation of these techniques to dialogue management.

The first section of this chapter focused on the use of efficient representations of probability and utility models. We described how directed graphical models could capture various probability and utility distributions. We reviewed in particular the main properties of Bayesian networks, dynamic Bayesian networks, and dynamic decision networks, and described the most important algorithms for inference and parameter estimation that have been tailored for these graphical models. The theoretical appeal of graphical models for the representation of complex stochastic phenomena is elegantly described by [?, p. 1](#)):

“Graphical models, a marriage between probability theory and graph theory, provide a natural tool for dealing with two problems that occur throughout applied mathematics

and engineering – uncertainty and complexity. In particular, they play an increasingly important role in the design and analysis of machine learning algorithms. Fundamental to the idea of a graphical model is the notion of modularity: a complex system is built by combining simpler parts. Probability theory serves as the glue whereby the parts are combined, ensuring that the system as a whole is consistent and providing ways to interface models to data. Graph theory provides both an intuitively appealing interface by which humans can model highly interacting sets of variables and a data structure that lends itself naturally to the design of efficient general-purpose algorithms.”

The second section laid down the core concepts and techniques in the field of reinforcement learning. We described the formal notion of a Markov Decision Process (MDP), composed of a set of states, a set of actions, a transition function describing temporal relations between states, and a reward function encoding the utilities of particular actions. We explained how MDPs can be extended to capture partial observability (leading to POMDPs), and how policies can be optimised for both MDPs and POMDPs using model-based and model-free techniques.

The final section translated these formal representations and optimisation techniques to the practical problem of dialogue management. Three learning strategies were distinguished: supervised learning, reinforcement learning with MDPs, and reinforcement learning with POMDPs. We surveyed a variety of approaches that differ along dimensions such as the representation of the dialogue state, the learning algorithm employed for the optimisation, the type and structure of the models that are to be estimated, and the source of data samples that is employed for this estimation. We discussed the respective merits and limitations of these dialogue optimisation strategies, and stressed in particular the importance of factorisation and generalisation methods to handle the complexity of real-world dialogue domains. The next chapter will now present in detail our own approach to this prominent problem.

# Chapter 4

## Probabilistic rules

The previous chapter fleshed out how dialogue could be represented as a stochastic process and argued for the use of probabilistic graphical models to efficiently encode the probability and utility models employed in dialogue management. Plain graphical models must however face non-trivial scalability issues when applied to dialogue domains associated with rich conversational contexts. The number of parameters necessary to perform dialogue state update and action selection can indeed increase rapidly with the complexity of the domain models. Alas, only small quantities of genuine training data are available in most dialogue domains, and usually cover only a small fraction of the state-action space of interest.

To address this discrepancy between the size of the parameter space and the amount of data available to estimate them, we introduce in this chapter a new approach to probabilistic dialogue modelling, based on the notion of *probabilistic rules*. Probabilistic rules are structured mappings between conditions and effects, and function as *high-level templates* for the construction of a dynamic decision network. The key advantage of this structured modelling approach is the drastic reduction of the number of parameters compared to traditional representations. We also argue that these expressive representations are particularly well suited to encode the probability and utility models of dialogue domains, where substantial amounts of expert knowledge can often be leveraged to structure the relationships between variables.

The chapter is divided in six sections. Section 4.1 exposes in general terms how structural assumptions can be applied to reduce the size and complexity of probabilistic models. Section 4.2 defines the formalism of probabilistic rules and its main theoretical properties. These definitions are then connected in Section 4.3 to the graphical models described in the previous chapter by showing how the rules are practically instantiated in the Bayesian network representing the dialogue state. Section 4.4 explains how this instantiation procedure is incorporated in the processing workflow for updating the dialogue state and selecting the system actions. Finally, Section 4.5 addresses some advanced modelling questions and Section 4.6 relates the approach to previous work.

### 4.1 Structural leverage

The starting point of our approach is the observation that the probability and utility models used in dialogue management often exhibit a fair amount of *internal structure*. We have already discussed in the previous chapter one simple instance of this internal structure, namely factored representations based on conditional independences. However, the internal structure of dialogue domains does not

limit itself to these basic independence assumptions, and much can be gained by exploiting other types of structural properties, as shall be argued in the next pages.

### Latent variables

The number of parameters required to estimate the distributions of a graphical model can often be reduced by introducing *latent variables* (i.e. unobserved or hidden variables) that act as intermediaries between the source and target variables. Indeed, many application domains are often best explained by the combination of a small number of distinct factors or influences, each encoded by a separate random variable and associated with a subset of input and output variables. This layer of latent variables is usually never observed directly but contribute to structuring the model.<sup>1</sup> In the particular case of medical diagnosis, the relations between predisposing factors and observed symptoms are for instance best described by postulating an intermediary layer of variables – possible diseases – that mediates between the predisposing factors and the observed symptoms. Figure 4.1 illustrates how latent variables can be exploited to provide an additional layer of abstraction within a graphical model.

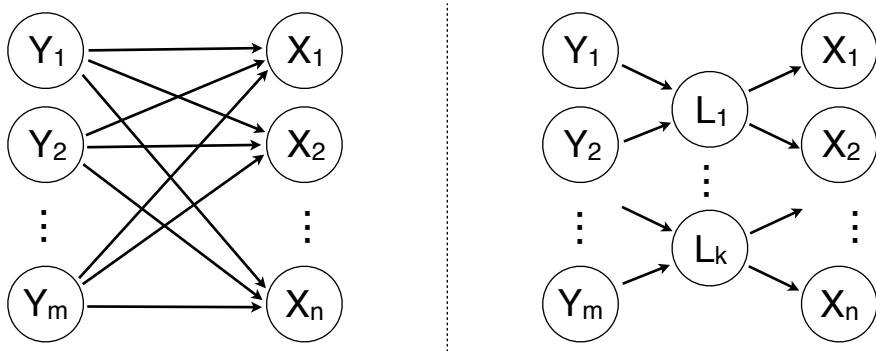


Figure 4.1: Comparison between a model that directly maps variables  $\mathbf{Y}$  to  $\mathbf{X}$  (left side) and one relying on latent variables  $\mathbf{L}$  to serve as intermediaries (right side).

Dialogue models can benefit from the inclusion of such latent variables. The transition function can for instance be modelled in terms of a limited number of latent variables, each responsible for capturing specific aspects of the interaction dynamics. We shall see in the forthcoming sections that probabilistic rules precisely operate as latent variables when instantiated in the dialogue state.

### Partitioning

A random variable  $X$  with parent variables  $Y_1, \dots, Y_m$  must specify a separate probability distribution for every possible assignment of values for the parent variables. In other words, the number of parameters required to specify the distribution  $P(X | Y_1, \dots, Y_m)$  is exponential in the number of parents  $m$ . Fortunately, the values of these parents variables can be grouped into *partitions* yielding similar outcomes for  $X$ . One can therefore directly define the conditional probability distributions on these groups rather than on the full enumeration of combined values for the parent

<sup>1</sup>The construction of layered computational models is one of the most active research topic in artificial intelligence and machine learning, and form in particular the foundations of deep learning approaches (?).

variables. Partitioning is an example of *abstraction mechanism* which can be used to reduce the model complexity and improve its ability to generalise to unseen examples.

Figure 4.2 illustrates such partitioning operation for the conditional probability distribution  $P(Fire | Weather, Rain)$ . The space of possible values for the parent variables is defined in this example as  $Val(Weather) \times Val(Rain)$  and contains 6 possible elements. We can observe that this space can be split in two partitions:  $Rain = true \vee Weather \neq hot$  and  $Rain = false \wedge Weather = hot$ . This partitioning allows a significant reduction of the number of parameters required for the conditional probability distribution. It should be noted that grouping value assignments into partitions corresponds to a modelling choice and can degrade the model accuracy if the partitions do not reflect actual similarities in the predicted outcomes.

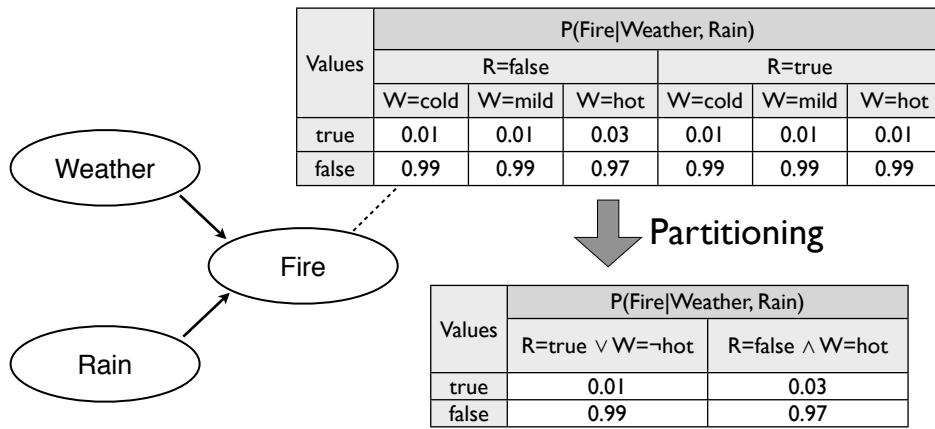


Figure 4.2: Partitioning for the conditional probability distribution  $P(Fire | Weather, Rain)$ .

Partitions must be both exhaustive (each combination of values for the parent variables must belong to one partition) and mutually exclusive (a combination of values can only belong to one partition). As we can observe from the example, partitions can often be concisely expressed via logical conditions on the variable values. A given assignment of values is then grouped in a partition if it satisfies the condition associated with it.

## Quantification

Many dialogue domains are composed of objects or entities related to one another. These domains are often difficult to directly encode by a fixed set of random variables, as the number of entities and relations may vary over time. Examples of relational structures include:

- Collections of physical objects in a visual scene, each described by specific features (colour, shape) and relations with other objects (e.g. spatial relations),
- Indoor environments topologically structured in rooms and spaces in which to navigate.
- Stacks of tasks to complete by the agent, each task being possibly connected to other tasks via precedence or inclusion relationships.
- Linguistic entities employed in the dialogue acts of the conversational partners, linked with one another through multiple syntactic, semantic, referential or pragmatic relations.

*First-order logic* provides an excellent basis for representing and manipulating such relational structures, as it offers a mathematically principled language for (1) referring to objects connected with one another through functions and relations and (2) describing their properties in a concise way through the use of universal and existential quantifiers.<sup>2</sup>

Graphical models can represent relational domains by instantiating one random variable for every possible grounding of the functions and predicates defined for the domain for a collection of objects.<sup>3</sup> A domain with two objects  $o_1$  and  $o_2$  and a relation  $leftOf(x, y)$  will for instance generate the four groundings  $leftOf(o_1, o_2)$ ,  $leftOf(o_2, o_1)$ ,  $leftOf(o_1, o_1)$  and  $leftOf(o_2, o_2)$ . The definition of probability and utility distributions that can handle the relational semantics of such representation is however problematic. In particular, generic properties and constraints such as  $\forall x, \neg leftOf(x, x)$  and  $\forall x, y, z, leftOf(x, y) \wedge leftOf(y, z) \Rightarrow leftOf(x, z)$  can be difficult to enforce at a global level, since classical probabilistic models offer no direct support for quantifiers. Their expressive power is indeed intrinsically limited to propositional logic.

The unification of first-order logic and probability theory has spanned a new research area called *statistical relational learning* (?). A common trait of most approaches to statistical relational learning is the definition of a logic-based description language which is employed as a template to generate classical probabilistic models given a set of constants. The introduction of quantifiers provide an abstraction mechanism to reduce the complexity of probability and utility models by describing constraints or relations that hold for all possible groundings of a given formula and can therefore apply to large sets of random variables.

## 4.2 Formalisation

We now outline a generic description framework for expressing the various types of internal structure we have just detailed. This description framework revolves around the notion of *probabilistic rules*. The framework was originally presented in ??).

The key idea is to represent distributions with the help of *if... then ... else* control structures, based on the following skeleton:

```

if (condition 1 holds) then
    Distribution 1 over possible effects
else if (condition 2 holds) then
    Distribution 2 over possible effects
...
else
    Distribution  $n$  over possible effects
  
```

Each *if... then* branch specifies both a *condition* on particular state variables and an associated distribution over possible *effects*. The *if ... then ... else* structure is read in sequential order, as in programming languages, until a satisfied condition is found, which causes the activation of the

---

<sup>2</sup>We shall not cover in this thesis the mathematical foundations of first-order logic, but the interested reader is invited to refer to e.g. ?) for a formal overview of the logical concepts mentioned throughout this thesis.

<sup>3</sup>Such operation is akin to *propositionalisation* in the terminology of first-order logic.

corresponding probabilistic effects.

We first present how probabilistic rules can express conditional probability distributions in terms of structured mappings between input and output variables. We then show how to generalise the formalism to utility distributions and extend it with quantification mechanisms.

A terminological note is here in order: we shall use the term *probabilistic rules* as an umbrella term that covers all types of rules in this thesis, while *probability rules* will only refer to rules expressing probability distributions over effects, and *utility rules* to rules expressing utility distributions.

### 4.2.1 Probability rules

Probability rules take the form of *if... then ... else* control structures and map a list of conditions on input variables to probabilistic effects on output variables. More formally, a rule is expressed as an ordered list  $\langle br_1, \dots, br_n \rangle$ , where each branch  $br_i$  is a pair  $(c_i, P(E_i))$ ,  $c_i$  is a condition and  $P(E_i)$  an associated distribution over possible effects. The distribution  $P(E_i)$  is a categorical distribution with possible effects  $Val(E_i) = \{e_{(i,1)}, \dots, e_{(i,m_i)}\}$ , where  $m_i$  is the number of alternative effects in  $P(E_i)$ . Each effect  $e_{(i,j)} \in Val(E_i)$  has a particular probability denoted  $p_{(i,j)}$ .

Given these elements, a basic probability rule reads as such:

$$\begin{aligned}
 & \textbf{if } (c_1) \textbf{ then} \\
 & \quad \left\{ \begin{array}{l} P(E_1 = e_{(1,1)}) = p_{(1,1)} \\ \dots \\ P(E_1 = e_{(1,m_1)}) = p_{(1,m_1)} \end{array} \right. \\
 & \textbf{else if } (c_2) \textbf{ then} \\
 & \quad \left\{ \begin{array}{l} P(E_2 = e_{(2,1)}) = p_{(2,1)} \\ \dots \\ P(E_2 = e_{(2,m_2)}) = p_{(2,m_2)} \end{array} \right. \\
 & \quad \dots \\
 & \textbf{else} \\
 & \quad \left\{ \begin{array}{l} P(E_n = e_{(n,1)}) = p_{(n,1)} \\ \dots \\ P(E_n = e_{(n,m_n)}) = p_{(n,m_n)} \end{array} \right.
 \end{aligned} \tag{4.1}$$

In the rest of this thesis, we will often use  $P(e_{(i,j)})$  as notational convenience for  $P(E_i = e_{(i,j)})$ .

### Conditions

The conditions  $c_i$  are expressed as logical formulae grounded in a subset of random variables defined in the dialogue state. This subset of state variables are the *input variables* of the rule, which we shall denote as  $I_1, \dots, I_k$ . Conditions can be arbitrarily complex logical formulae connected by conjunction, disjunction and negation, and (as we shall see in Section 4.2.3) can also include universally quantified variables. The examples ( $Rain = true \vee Weather \neq hot$ ) and

$(Rain = \text{false} \wedge Weather = \text{hot})$  in Figure 4.2 are instances of valid conditions on the two input variables *Rain* and *Weather*.

Given that a rule is defined through a *if ... then ... else* control structure, the partitioning is guaranteed by construction to be exhaustive and mutually exclusive (only one branch will be followed). When provided with an assignment of values on the input variables, the conditions are tested in sequential order until one is satisfied. When no terminating **else** block is explicitly specified at the end of a rule, the framework assumes a final **else** block associated with a void effect to ensure that the partitioning is exhaustive. The last condition  $c_n$  is thus guaranteed to be always trivially satisfied irrespective of the input variable values.

The conditions on the input variables offer a compact partitioning of the state space to mitigate the dimensionality curse. Without this partitioning in alternative conditions, a rule ranging over input variables  $I_1, \dots, I_k$  each containing  $q$  possible values would need to enumerate  $q^k$  possible assignments. Partitioning this space reduces this number to  $n$  mutually exclusive partitions, where  $n$  corresponds to the number of conditions for the rule and is usually small.

## Effects

Associated to each condition  $c_i$  stands a collection of mutually exclusive effects  $e_{(i,1)}, \dots, e_{(i,m_i)}$ . Each effect  $e_{(i,j)}$  defines a specific assignment of values for a set of variables called the *output variables* of the rule. An effect is represented as a conjunction of (variable,value) pairs  $O_1 = o_1 \dots \wedge O_l = o_l$  where  $O_1, \dots, O_l$  are the output variables (which may already exist or yet to be created) and  $o_1, \dots, o_l$  the corresponding values assigned to these variables. In the partitioning example from Figure 4.2, the output variable is unique and corresponds to *Fire*. We shall however encounter examples of rules with more than one output variable.

Effects can be void – that is, represent an empty assignment. In such case, the effect does not lead to any change in the distribution of the output variables for the rule.

## Probabilities

Each effect  $e_{(i,j)}$  is assigned with a probability  $p_{(i,j)} = P(E_i = e_{(i,j)})$  that must satisfy the usual probability axioms  $p_{(i,j)} \geq 0 \forall i, j$  and  $\sum_{j=1}^{m_i} p_{(i,j)} = 1 \forall i$ . The probabilities can be either fixed by hand or correspond to parameters to estimate from data. Chapters 5 and 6 detail how Dirichlet distributions can be exploited to estimate probability parameters.

## Example

Rule  $r_1$  illustrates a simple example of probability rule:

$$\begin{aligned}
r_1 : & \quad \mathbf{if} (Rain = \text{false} \wedge Weather = \text{hot}) \mathbf{then} \\
& \quad \begin{cases} P(Fire = \text{true}) = 0.03 \\ P(Fire = \text{false}) = 0.97 \end{cases} \\
& \quad \mathbf{else} \\
& \quad \begin{cases} P(Fire = \text{true}) = 0.01 \\ P(Fire = \text{false}) = 0.99 \end{cases}
\end{aligned}$$

Rule  $r_1$  has two input variables: *Rain* and *Weather* as well as one output variable *Fire*. The rule specifies that the probability of a fire is 0.03 in case of no rain and a hot weather and 0.01 in all other cases. The rule structure enables the conditional probability distribution for *Fire* to be specified with only four probabilities in comparison to twelve for the original CPD (Figure 4.2).

### 4.2.2 Utility rules

The rule-based formalism we have outlined can also be used to express utility distributions with only minor notational changes. Utility rules essentially retain the same form as probability rules, with one important exception, namely that the probabilistic effects are replaced by utility distributions over particular assignments of decision variables.

Formally, a utility rule is an ordered list  $\langle br_1, \dots, br_n \rangle$ , where each branch  $br_i$  is a pair  $(c_i, U_i)$  where  $c_i$  is a condition and  $U_i$  an associated utility distribution over possible assignments of decision variables. The utility distribution  $U_i$  specifies a set of possible decisions  $d_{(i,1)}, \dots, d_{(i,m_i)}$ . Each decision  $d_{(i,j)}$  has a particular utility value denoted  $u_{(i,j)}$ . Utility rules can be expressed in the following manner:

$$\begin{aligned}
& \text{if } (c_1) \text{ then} \\
& \quad \left\{ \begin{array}{l} U_1(d_{(1,1)}) = u_{(1,1)} \\ \dots \\ U_1(d_{(1,m_1)}) = u_{(1,m_1)} \end{array} \right. \\
& \text{else if } (c_2) \text{ then} \\
& \quad \left\{ \begin{array}{l} U_2(d_{(2,1)}) = u_{(2,1)} \\ \dots \\ U_2(d_{(2,m_2)}) = u_{(2,m_2)} \end{array} \right. \\
& \quad \dots \\
& \text{else} \\
& \quad \left\{ \begin{array}{l} U_n(d_{(n,1)}) = u_{(n,1)} \\ \dots \\ U_n(d_{(n,m_n)}) = u_{(n,m_n)} \end{array} \right.
\end{aligned} \tag{4.2}$$

A utility rule assigns utility values to particular system decisions depending on conditions on the state variables. As for probability rules, the conditions  $c_i$  are defined as arbitrary logical formulae on input variables  $I_1, \dots, I_k$ . The decisions  $d_{(i,j)}$  are assignments  $A_1 = a_1 \dots \wedge A_l = a_l$  where the variables  $A_1, \dots, A_l$  are decision variables and  $a_1, \dots, a_l$  possible values for these variables. The utility values  $u_{(i,j)}$  are real numbers (which may be positive or negative).

Although most utility rules only include one single decision variable, the possibility to integrate multiple decision variables is helpful in domains where the system can execute multiple actions in parallel. Such situations can arise in human-robot interaction and multi-modal applications, as the system can communicate through both verbal and non-verbal channels and is often able to perform physical actions in addition to communicative acts.

## Example

Rule  $r_2$  provides a simple example of utility rule:

$$r_2 : \begin{array}{l} \textbf{if } (Fire = true) \textbf{ then} \\ \quad \left\{ \begin{array}{l} U(Tanker = drop-water) = 5 \\ U(Tanker = wait) = -5 \end{array} \right. \\ \textbf{else} \\ \quad \left\{ \begin{array}{l} U(Tanker = drop-water) = -1 \\ U(Tanker = wait) = 0 \end{array} \right. \end{array}$$

Rule  $r_2$  stipulates the respective utilities of the two possible utility values for the decision variable  $Tanker$  depending on the variable  $Fire$ .

### 4.2.3 Quantification

Quantification is a powerful mechanism to abstract over particular relational aspects of the domain structure. Logical variables can be included in the specification of both the conditions and effects of a given rule, and are universally quantified on top of the rule.<sup>4</sup> A rule containing the quantified variables  $y_1 \dots y_p$  in its conditions and/or effects is therefore formalised as:

$$\begin{aligned} & \forall y_1, y_2, \dots, y_p : \\ & \quad \textbf{if } (c_1) \textbf{ then} \\ & \quad \quad \left\{ \begin{array}{l} P(E_1 = e_{(1,1)}) = p_{(1,1)} \\ \dots \\ P(E_1 = e_{(1,m_1)}) = p_{(1,m_1)} \end{array} \right. \\ & \quad \textbf{else if } (c_2) \textbf{ then} \\ & \quad \quad \left\{ \begin{array}{l} P(E_2 = e_{(2,1)}) = p_{(2,1)} \\ \dots \\ P(E_2 = e_{(2,m_2)}) = p_{(2,m_2)} \end{array} \right. \\ & \quad \dots \\ & \quad \textbf{else} \\ & \quad \quad \left\{ \begin{array}{l} P(E_n = e_{(n,1)}) = p_{(n,1)} \\ \dots \\ P(E_n = e_{(n,m_n)}) = p_{(n,m_n)} \end{array} \right. \end{aligned} \tag{4.3}$$

The formalisation allows specific elements  $y_1, \dots, y_p$  in the conditions and effects to be *under-*

---

<sup>4</sup>These variables are variables in the sense of first-order logic, and are not to be confused with the random variables of the probabilistic model.

*specified.* The mapping between conditions and effects specified by the rule holds for every possible assignment of the underspecified variables. Based on this quantification mechanism, probabilistic rules can cover large portions of the state space in a highly compact manner, based on a reduced number of parameters. One of the key advantages of such representation is that it allows for powerful forms of *parameter sharing*, as the effect probabilities  $p_{(i,j)}$  in the above rule are made independent of the various possible instantiations of the variables  $y_1, \dots, y_p$ . Quantification also applies to utility rules in the same manner.

### Example

Rule  $r_3$  provides a simple example of probability rule including a quantified variable:

$$r_3 : \forall y :$$

**if** ( $shape(y) = sphere$ ) **then**

$$\begin{cases} P(graspable(y) = true) = 0.9 \\ P(graspable(y) = false) = 0.1 \end{cases}$$

**else if** ( $shape(y) = cone$ ) **then**

$$\begin{cases} P(graspable(y) = true) = 0.2 \\ P(graspable(y) = false) = 0.8 \end{cases}$$

Rule  $r_3$  specifies how the graspability of a given object  $y$  depends on its shape (a sphere being easier to grasp than a cone). Similarly, rule  $r_4$  defines the utility of a grasping action depending on the task and object graspability:

$$r_4 : \forall y :$$

**if** ( $task = grasp(y) \wedge graspable(y) = true$ ) **then**

$$\begin{cases} U(a_m = grasp(y)) = 2 \end{cases}$$

**else**

$$\begin{cases} U(a_m = grasp(y)) = -2 \end{cases}$$

Rule  $r_4$  associates an utility of 2 to the action of grasping an object  $y$  when it corresponds to the task and is feasible. Grasping the object in any other case results in a negative utility of -2.

## 4.3 Rule instantiation

We represent the dialogue state as a Bayesian network over state variables, in line with other dialogue management approaches such as [??](#)). Rules are then applied at runtime on this dialogue state. The instantiation is performed by creating a distinct node for every rule to apply. These rule nodes are essentially latent variables that serve as intermediaries between input and output variables. Albeit the presence of these rules is never directly observed, they help structuring the relations between variables and enable the system designer to decompose complex probability and utility models into smaller parts.

We describe below the instantiation procedure for each type of rule. For the sake of clarity, we shall first limit our discussion to rules without quantifiers, and then demonstrate how quantifiers can be accounted for in the instantiation process.

### 4.3.1 Probability rules

Let  $\mathcal{B}$  be the Bayesian network representing the current dialogue state, and  $\mathcal{R}$  a set of rules to apply to this dialogue state. For each rule  $r \in \mathcal{R}$ , a distinct chance node is created.<sup>5</sup> This chance node represents a random variable defined on the possible effects of the rule. The node is conditionally dependent on the input variables of the rule (i.e. the set of all variables that are mentioned in the rule conditions), and is also connected via outgoing edges to its output variables (i.e. the set of all variables that are mentioned in the rule effects).

Figure 4.3 illustrates this construction process on a constructed example composed of the two rules  $r_5$  and  $r_6$ . To simplify the rule representation, we shall usually omit the explicit specification of the probability for the empty effect in the effect distributions. The remaining probability mass in the rules is thus by default assigned to the empty effect.

The two rules  $r_5$  and  $r_6$  are applied on the state variables  $A, B, C$  and  $D$ . The application of the two rules results in an update of the variable  $A$  and the creation of a new variable  $E$ . The nodes corresponding to the output variables are by convention denoted with a prime to distinguish them from the input nodes.

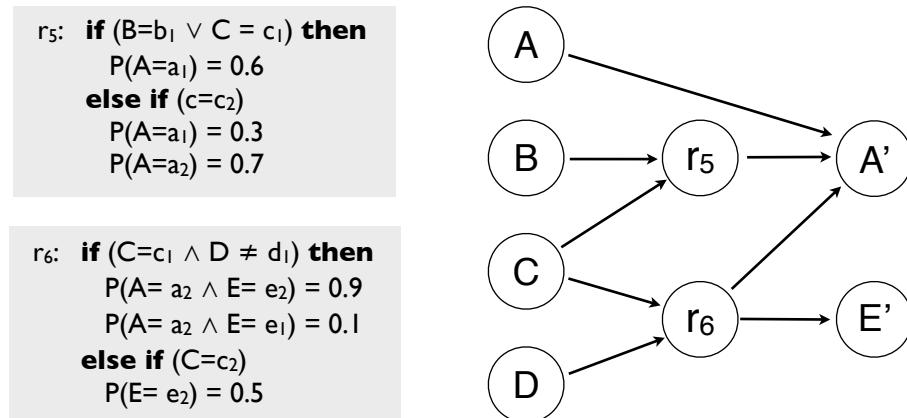


Figure 4.3: Example of instantiation for the two probability rules  $r_5$  and  $r_6$ .

The random variable represented by the node  $r_5$  has three possible values, reflecting the effects described in the rule:  $Val(r_5) = \{\{A=a_1\}, \{A=a_2\}, \{\cdot\}\}$ , where  $\{\cdot\}$  denotes the empty effect. Similarly, the random variable  $r_5$  has four alternative effects:  $Val(r_6) = \{\{A=a_2 \wedge E=e_2\}, \{A=a_2 \wedge E=e_1\}, \{E=e_2\}, \{\cdot\}\}$ .

We shall also adopt the following terminology to denote the probability distributions created through the instantiation procedure:

<sup>5</sup>The original instantiation algorithm presented in (?) included two separate nodes: one for the rule condition and one for the effect. The formalism was later simplified to one single node.

- The conditional probability distribution associated with rule nodes such as  $r_5$  and  $r_6$  given their inputs is a *rule distribution*.
- The conditional probability distribution associated with output variables such as  $A'$  and  $E'$  given the rule nodes that determine it is an *output distribution*.

## Rule distributions

The rule distributions directly reflect the rule semantics. Formally, the conditional probability distribution of a rule node  $r$  given its input variables  $I_1, \dots, I_k$  is defined as:

$$P(r = e | I_1 = i_1, \dots, I_k = i_k) = P(E_i = e) \quad (4.4)$$

where  $i = \min_i(c_i \text{ is satisfied with } I_1 = i_1 \wedge \dots \wedge I_k = i_k)$

Formally speaking, a condition  $c_i$  is said to be satisfied iff the input assignment logically entails that the condition is true, that is  $(I_1 = i_1 \wedge \dots \wedge I_k = i_k) \vdash c_i$ . The rule conditions are checked in sequential order until one condition is found to be satisfied. The rule distribution is then simply determined as the effect distribution  $P(E_i = e)$  associated with the first satisfied condition  $c_i$ . As the last condition  $c_n$  corresponds to the final **else** block and is therefore always trivially true, there will always be at least one satisfied condition.

As an example, the rule distribution  $P(r_5 | B = b_1, C = c_1)$  for the node  $r_5$  in Figure 4.3 is defined as:

- $P(r_5 = \{A = a_1\} | B = b_1, C = c_1) = 0.6$
- $P(r_5 = \{\cdot\} | B = b_1, C = c_1) = 0.4$

Similarly, the distribution  $P(r_6 | C = c_1, D = d_1)$  is a distribution with the empty effect  $\{\cdot\}$  assigned to a probability 1.

## Output distributions

An output node  $X'$  is conditionally dependent on all the rule nodes that refer to the variable  $X$  in their effects. In addition, output nodes that correspond to the updated version of existing nodes (such as  $A'$  in the example of Figure 4.3) also include a conditional dependence on these existing nodes. The output distribution is a reflection of the combination of effects specified in the parent rules. The conditional probability distribution  $P(X' | r_1 = e_1, \dots, r_n = e_n)$  for an output variable  $X'$  with  $n$  incoming rule nodes is defined in the following manner:

$$P(X' = x' | r_1 = e_1, \dots, r_n = e_n) = \begin{cases} \frac{\sum_{v \in \mathbf{e}(X)} \mathbf{1}(x' = v)}{|\mathbf{e}(X)|} & \text{if } \mathbf{e}(X) \neq \emptyset \\ \mathbf{1}(x' = \text{None}) & \text{otherwise} \end{cases} \quad (4.5)$$

where the following notation is used:

- $\mathbf{e}$  is the conjunction of all effects, i.e.  $\mathbf{e} = e_1 \wedge \dots \wedge e_n$ . This conjunction can include more than one assigned value for a particular variable.

- $\mathbf{e}(X)$  denotes the (possibly empty) list of values specified for the variable  $X$  in  $\mathbf{e}$ .
- $\mathbf{1}(b)$  is the indicator function for the Boolean  $b$ , with  $\mathbf{1}(b) = 1$  if  $b$  is true and 0 otherwise.

Equation (4.5) stipulates that the distribution for  $X'$  will follow the values assigned in the effect(s) provided that at least one effect specifies a value for it. If the effects include conflicting assignments, the distribution is spread uniformly over the alternative values. If no effects  $e_1, \dots, e_n$  specifies a value for  $X$ , the value for  $X'$  is set to a default *None* value with probability 1.

If the node  $X'$  is an update of an existing node  $X$ , the procedure remains essentially the same as for (4.5), except in the case where all the effects specify empty assignments for the variable. In such case, the distribution for  $X'$  will fall back to the value defined for the existing node  $X$  instead of being assigned a *None* value:

$$P(X' = x' | r_1 = e_1, \dots, r_n = e_n, X = x) = \begin{cases} \frac{\sum_{v \in \mathbf{e}(X)} \mathbf{1}(x' = v)}{|\mathbf{e}(X)|} & \text{if } \mathbf{e}(X) \neq \emptyset \\ \mathbf{1}(x' = x) & \text{otherwise} \end{cases} \quad (4.6)$$

As an example, the output distribution  $P(A' | r_5 = \{\cdot\}, r_6 = \{A = a_2 \wedge E = e_2\}, A = a_3)$  in Figure 4.3 results in a deterministic distribution with a unique value  $a_2$  with probability 1, since  $\mathbf{e}(A) = \{a_2\}$ . If the two rules generate conflicting assignments, the probability mass is divided equally over the alternative values.<sup>6</sup> The output distribution  $P(A' | r_5 = \{A = a_1\}, r_6 = \{A = a_2 \wedge E = e_2\}, A = a_3)$  provides two alternative values for  $A$ :  $\mathbf{e}(A) = \{a_1, a_2\}$ . The output distribution is thus a uniform distribution with two values:  $a_1$  and  $a_2$ , each with probability 0.5. Finally, if all effects are empty, the output distribution is a simple copy of the distribution for the existing variable:  $P(A' | r_5 = \{\cdot\}, r_6 = \{\cdot\}, A = a_3)$  has a unique value  $a_3$  with probability 1.

Output distributions are directly derived from the effects in the rule nodes and are thus entirely parameter-free. The ordering of the parent rules in the conditional probability distribution is arbitrary. The resulting distribution bears resemblance to the probabilistic Independence of Causal Influence (pICI) described by ?).

## Instantiation algorithm

The procedure for instantiating a rule in a given dialogue state is detailed in Algorithm 4.

The first steps of the instantiation process are to extract in the Bayesian network the input variables of the rule (line 1), create a node corresponding to the rule (line 2) and include its dependency edges in the network (line 3). The algorithm then checks whether at least one effect in  $r$  is non-empty given its conditional dependences (lines 4). If all effects are empty, the rule node is irrelevant and can be directly pruned (line 5). Otherwise, the output variables are extracted (line 7), and output nodes that do not already exist in the network are created (line 10-11). The final step is to establish dependency edges between the rule node and these output variables (line 13).

---

<sup>6</sup>This uniform division of the probability mass reflects that, in the absence of further knowledge, all rules are assumed to have the same “weight”. If two or more rules generate contradictory effects, there is therefore no way to tell which rule is to take precedence. This assumption could however be refined in the future by e.g. associating numerical weights to particular rules or specifying a dominance hierarchy between the rules.

---

**Algorithm 4 : INSTANTIATEPROBRULE ( $\mathcal{B}, r$ )**


---

**Input:** Bayesian network  $\mathcal{B}$  for the current state

**Input:** Probability rule  $r$  to instantiate in network

```

1:  $I_1, \dots, I_k \leftarrow$  input variables for  $r$ 
2: Create chance node  $r$  with the rule distribution in Eq. (4.4)
3: Add node  $r$  and dependency edges  $I_1, \dots, I_k \rightarrow r$  to  $\mathcal{B}$ 
4: if  $Val(r) = \{\cdot\}$  then
5:   Prune  $r$  from  $\mathcal{B}$ 
6: else
7:    $O_1, \dots, O_l \leftarrow$  output variables mentioned in the effects of  $r$ 
8:   for all variable  $O \in O_1, \dots, O_l$  do
9:     if  $O'$  not already in  $\mathcal{B}$  then
10:      Create chance node  $O'$  with the output distribution in Eq. (4.5)-(4.6)
11:      Add node  $O'$  and (in case  $O$  exists) dependency edge  $O \rightarrow O'$  to  $\mathcal{B}$ 
12:    end if
13:    Add dependency edge  $r \rightarrow O'$  to  $\mathcal{B}$ 
14:  end for
15: end if
```

---

### 4.3.2 Utility rules

Utility rules are instantiated in the Bayesian network according to a similar procedure, with two notable differences compared to probability rules:

- As utility rules define utility distributions, their instantiation correspond to utility nodes instead of chance nodes.
- Instead of output variables, the rules are associated with decision variables. The association direction is inverted, as the decision node must be input to the utility node.

The result of the instantiation process is a decision network that incorporates chance nodes (corresponding to the state variables), utility nodes (corresponding to the utility rules) and associated decision nodes. Figure 4.4 illustrates the instantiation of two utility rules  $r_7$  and  $r_8$ .

The utility distribution associated with each rule is a direct translation of the *if ... then ... else* rule. Formally, the utility distribution generated by a rule  $r$  with input variables  $I_1, \dots, I_k$  and decision variables  $A_1, \dots, A_l$  is defined as:

$$U_r(I_1=i_1, \dots, I_k=i_k, A_1=a_1, \dots, A_l=a_l) = U_i(A_1=a_1 \wedge \dots \wedge A_l=a_l) \quad (4.7)$$

where  $i = \min_i(c_i \text{ is satisfied with } I_1=i_1 \wedge \dots \wedge I_k=i_k)$

If no utility is explicitly specified for  $A_1=a_1 \wedge \dots \wedge A_l=a_l$ , the default value is zero.

As is conventionally assumed in decision networks, the total utility for a given assignment of decision variables is defined as the *sum* of all utilities. In case  $A=a_1, B=b_1$  and  $C=c_1$ , we can therefore calculate the total utility for the actions  $D'=d_1 \wedge E'=e_1$  to be equal to  $2 - 4 = -2$ .

```

r7: if (A=a1  $\vee$  B = b2) then
    U(D=d1) = 2
    U(D=d2) = -1
else if (A=a2)
    U(D=d1) = -2

```

```

r8: if (B=b1  $\wedge$  C ≠ c1) then
    U(D=d1  $\wedge$  E= e2) = 1.5
    U(D=d2) = -0.5
else if (C=c1)
    U(E=e1) = -4

```

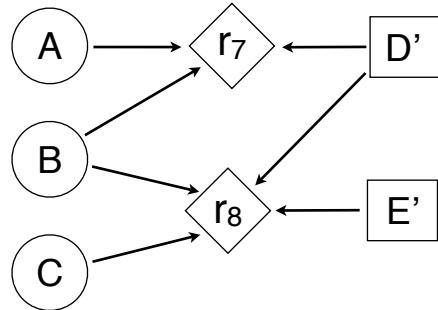


Figure 4.4: Example of instantiation for the two utility rules  $r_7$  and  $r_8$ .

### Instantiation algorithm

The procedure for instantiating a utility rule is similar in most respects to the one already outlined for probability rules. Algorithm 5 details the procedure, starting from the extraction of the input variables, the creation of the rule node, and the inclusion of conditional dependences (line 1-3). The algorithm then checks if the utility distribution stipulates a non-zero utility for at least one decision (line 4). If the answer is negative, the node is essentially irrelevant and can be pruned (line 5). The decision variables associated with the rule are extracted (line 7), and a corresponding decision node is created if it does not already exist (line 10). Finally, the possible values specified for the decision variable are integrated to the node (line 12), and a dependency edge is established between the decision node and the utility node for the rule (line 13).

---

#### Algorithm 5 : INSTANTIATEUTILRULE ( $\mathcal{B}, r$ )

---

**Input:** Bayesian network  $\mathcal{B}$  for the current state

**Input:** Utility rule  $r$  to instantiate in network

- 1:  $I_1, \dots, I_k \leftarrow$  input variables for *rule*
  - 2: Create utility node  $r$  with the utility distribution in Eq. (5.7)
  - 3: Add node  $r$  and dependency edges  $I_1, \dots, I_k \rightarrow r$  to  $\mathcal{B}$
  - 4: **if** utility distribution is empty for all inputs **then**
  - 5:     Prune  $r$  from  $\mathcal{B}$
  - 6: **else**
  - 7:      $A_1, \dots, A_l \leftarrow$  decision variables mentioned in the effects of  $r$
  - 8:     **for all** variable  $A \in A_1, \dots, A_l$  **do**
  - 9:         **if**  $A'$  not already in  $\mathcal{B}$  **then**
  - 10:             Create decision node  $A'$  and add it to  $\mathcal{B}$
  - 11:         **end if**
  - 12:         Add in  $Val(A')$  the action values specified in the effects of  $r$
  - 13:         Add dependency edge  $A' \rightarrow r$  to  $\mathcal{B}$
  - 14:     **end for**
  - 15: **end if**
-

### 4.3.3 Quantification

We saw in Section 4.2.3 that conditions and effect could include universally quantified variables, but have not yet discussed how such underspecified rules could be practically instantiated in the Bayesian network. The general instantiation principle remains unchanged: to each rule corresponds a distinct rule node responsible for the mapping between input and output variables (or decision variables for utility rules). The instantiation procedure must be however extended to accommodate the presence of quantified variables. The key idea is to find all relevant *groundings* for the quantified variables, and then calculate the effect distribution for each grounding. This method of handling quantifiers by extracting all possible groundings and reasoning at the propositional level is an instance of *ground inference* (?).

#### Extraction of input variables

Universally quantified rules may underspecify both the names and values of random variables, as we saw in the examples of Section 4.2.3. Rule  $r_3$  includes for instance a reference to an underspecified random variable  $shape(y)$ . In order to instantiate the rule, the system must therefore first determine all random variables included in the model that match the underspecified description. If rule  $r_3$  is instantiated on a state containing two objects  $o_1$  and  $o_2$ , the resulting input variables will therefore be  $shape(o_1)$  and  $shape(o_2)$ .

Algorithm 6 outlines how this search for matching input variables can proceed. The first step is to extract the initial input variables associated with the rule, which may include underspecified descriptions such as  $shape(y)$ . The algorithm then loops on each underspecified description to find possible groundings in the random variables of the Bayesian network. The final result corresponds to the combination of the fully specified input variables for the rule and the groundings for the underspecified variables.

---

#### Algorithm 6 : GETINPUTVARIABLES ( $\mathcal{B}, r$ )

---

**Input:** Bayesian network  $\mathcal{B}$  for the current state

**Input:** Probability or utility rule  $r$

```

1:  $\mathcal{I}_r \leftarrow$  Initial (possibly underspecified) input variables for  $r$ 
2:  $\mathcal{U}_r \leftarrow$  Subset of variable names in  $\mathcal{I}_r$  that are underspecified
3:  $\mathcal{G}_r \leftarrow$  Set of possible groundings for  $\mathcal{U}_r$ , initially empty
4: for underspecified variable name  $u \in \mathcal{U}_r$  do
5:   for random variable  $X$  in  $\mathcal{B}$  do
6:     if  $X$  matches  $u$  then
7:        $\mathcal{G}_r \leftarrow \mathcal{G}_r \cup [X]$ 
8:     end if
9:   end for
10: end for
11: return  $(\mathcal{I}_r / \mathcal{U}_r) \cup \mathcal{G}_r$ 

```

---

#### Extraction of relevant groundings

Once the input variables for the rules are retrieved, the next step is to establish the set of relevant groundings  $G$  for the universally quantified variables. The groundings are always determined

relative to a particular assignment of values for the (grounded) input variables  $I_1, \dots, I_k$ .

Given an input assignment  $\{I_1 = i_1 \wedge \dots \wedge I_k = i_k\}$ , the set of groundings  $G$  are derived in a heuristic manner, by checking which ground terms (constants and functions of constants) from the input assignment can function as proper substitutions for the quantified variables. These ground terms define the domain of discourse for the application of the universal quantifier. The collection of relevant ground terms is then combined into subsets of size  $p$ , where  $p$  corresponds to the number of universally quantified variables  $y = y_1, \dots, y_p$ . These combinations form the groundings  $G$  for the input assignment.

### Quantified probability rules

As we have seen, probability rules are instantiated as chance nodes associated with a rule distribution. For rules containing universal quantifiers, each grounding in  $G$  gives rise to a particular distribution over effects. The instantiation procedure generates a distinct effect distribution for each grounding  $g_j$ :

$$P(r_{g_j} = e' \mid I_1 = i_1, \dots, I_k = i_k) = P(E_i = e) \quad (4.8)$$

where  $i = \min_i(c_i[y/g_j]$  is satisfied with  $I_1 = i_1 \wedge \dots \wedge I_k = i_k)$   
and  $e' = e[y/g_j]$

The expression  $\phi[a/b]$  denotes (as in formal logic) the formula  $\phi$  where all instances of  $a$  are substituted by  $b$ .

Empty and redundant effect distributions are discarded. The grounding procedure results in a collection of distributions  $\langle P(r_{g_1}), \dots, P(r_{g_p}) \rangle$ . The final conditional probability distribution for the rule node is then defined as the joint distribution over these grounding-specific distributions:

$$P(r = [e_1 \wedge \dots \wedge e_q] \mid I_1 = i_1, \dots, I_k = i_k) = \prod_{j=1}^q P(r_{g_j} = e_j) \quad (4.9)$$

Figure 4.5 shows how the rule  $r_3$  is instantiated in a state with two objects  $o_1$  and  $o_2$ , each associated with a random variable describing its shape.

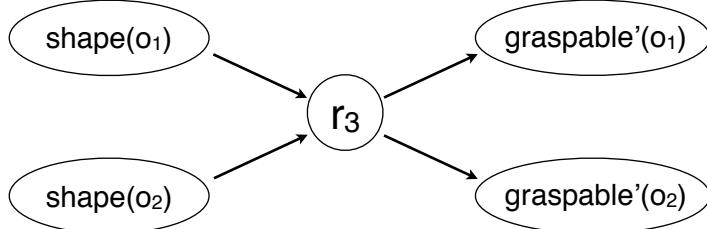


Figure 4.5: Instantiation of the probability rule  $r_3$  on a state with two objects  $o_1$  and  $o_2$ .

As an example, the probability distribution  $P(r_3 \mid \text{shape}(o_1) = \text{sphere}, \text{shape}(o_2) = \text{cone})$  has two relevant groundings  $y = o_1$  and  $y = o_2$ , from which four alternative effects are derived:

- $\{graspable(o_1)=true \wedge graspable(o_2)=true\}$  with probability  $0.9 \times 0.2 = 0.18$
- $\{graspable(o_1)=true \wedge graspable(o_2)=false\}$  with probability  $0.9 \times 0.8 = 0.72$
- $\{graspable(o_1)=false \wedge graspable(o_2)=true\}$  with probability  $0.1 \times 0.2 = 0.02$
- $\{graspable(o_1)=false \wedge graspable(o_2)=false\}$  with probability  $0.1 \times 0.8 = 0.08.$

The effects specified for  $r_3$  mention two output variables:  $graspable(o_1)$  and  $graspable(o_1)$ . These two variables are thus created as children of the node  $r_3$ .

### Quantified utility rules

Utility rules can be similarly extended to accommodate quantified variables. As for probability rules, the instantiation of universally quantified utility rules proceeds by determining a set of groundings and generating a particular utility distribution for each.<sup>7</sup> For a utility rule with input variables  $I_1, \dots, I_k$ , decision variables  $A_1, \dots, A_l$  and quantified variables  $y$ ,

$$U_{g_j}(I_1=i_1, \dots, I_k=i_k, A_1=a'_1, \dots, A_l=a'_l) = U_i(A_1=a_1 \wedge \dots \wedge A_l=a_l) \quad (4.10)$$

where  $i = \min_i(c_i[y/g_j]$  is satisfied with  $I_1=i_1 \wedge \dots \wedge I_k=i_k$ )  
and  $a'_1 = a_1[y/g_j], \dots, a'_l = a_l[y/g_j]$

After discarding empty and redundant distributions, the result is a set of utility distributions  $\langle U_{g_1}, \dots, U_{g_p} \rangle$ . The total utility distribution for the rule is then constructed by adding up the grounding-specific utility distributions:

$$U_r(I_1, \dots, I_k, A_1, \dots, A_l) = \sum_{j=1}^q U_{g_j}(I_1, \dots, I_k, A_1, \dots, A_l) \quad (4.11)$$

The result of instantiating rule  $r_4$  on a state with two objects  $o_1, o_2$  with associated random variables  $graspable(o_1)$  and  $graspable(o_2)$  is shown in Figure 4.6.

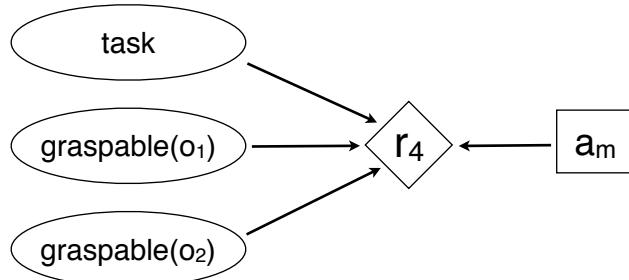


Figure 4.6: Instantiation of the quantified utility rule  $r_4$  on a state with two objects  $o_1$  and  $o_2$ .

---

<sup>7</sup>The extraction of groundings is slightly modified for utility rules in order to integrate the ground terms appearing in both the input and decision assignments.

The utility distribution  $U_{r_4}(\text{task} = \text{grasp}(o_1), \text{graspable}(o_1) = \text{true}, \text{graspable}(o_2) = \text{false}, a_m)$  assigns the action  $a_m = \text{grasp}(o_1)$  to a utility of 2, while the action  $a_m = \text{grasp}(o_2)$  is assigned to a utility of -2.

## Tractability aspects

Although the use of universally quantifiers greatly improves the expressivity of probabilistic rules, they also tend to increase the in- and out-degrees of rule nodes (that is, the cardinality of their parents and children nodes). Approximate inference techniques are thus necessary to handle this conditional structure in a tractable manner. Sampling methods such as likelihood weighting have in practice proved to work well in this setting (cf. Section 3.1.2).

It should be stressed that the groundings are always extracted *given a specific value assignment* for the input variables. By restricting the groundings to this limited domain of discourse, we ensure that the number of alternative effects remains bounded and avoid the generation of spurious effects. This instantiation procedure was found to be much more efficient than copying the rule in distinct nodes, as investigated in earlier implementations of the formalism (?). This procedure departs from other frameworks such as Markov Logic Networks, where the functions and predicates are duplicated for every possible grounding of variables in the domain (?).

## 4.4 Processing workflow

The two previous sections detailed how probability and utility rules are internally defined, and how they can be instantiated as nodes of a graphical model. We are now ready to explain how collections of rules are practically applied at runtime to update the dialogue state and perform action selection. The general workflow is strongly inspired by information-state approaches to dialogue management (?), as the dialogue state serves as a central blackboard monitored by various groups of rules that are “triggered” upon relevant changes.

The section first describes how dialogue domains are organised in collections of rules called “models”, and then goes on to explain how these models are applied to update state variables and express the utility of particular actions. We also demonstrate the general processing workflow on a detailed example.

### 4.4.1 Domain representation

Dialogue domains can consist of multiple probability and utility rules. These rules are internally grouped in collections of rules called *models*. A model is simply a collection of rules that is associated with one or more “trigger” variables that specify when the model is to be instantiated. Each model is attached the dialogue state and monitors it to detect changes affecting their trigger variables. When these trigger variables are changed at runtime by another module, they lead to the instantiation of the model rules. Formally, a model  $m$  is defined as a pair  $\langle \mathcal{T}_m, \mathcal{R}_m \rangle$  where  $\mathcal{T}_m$  corresponds to the trigger variables and  $\mathcal{R}_m$  to the rules included in the model.

A dialogue domain is represented as a pair  $\langle \mathcal{B}_0, \mathcal{M} \rangle$ , where  $\mathcal{B}_0$  is the initial dialogue state and  $\mathcal{M}$  the set of rule-based models attached to it. The organisation of rules into models allows the system designer to structure the application pipeline in a modular manner. Each model can be intuitively viewed as a distinct component responsible for a particular inference or decision step.

Section 7.1.2 explains how dialogue domains (and the models that compose them) are practically encoded in the openDial architecture, based on an XML specification.

#### 4.4.2 Update algorithm

The software architecture adopted in this thesis takes the form of an event-driven, blackboard architecture revolving around a dialogue state  $\mathcal{B}$  represented as a Bayesian network.

The update procedure is shown in Algorithm 7. The procedure is started upon the reception of new variables to incorporate in the state, such as new user inputs processed by the speech recognition and natural language understanding components. The first step is to insert the variables in the dialogue state and possibly relate them to their predicted values (lines 2-3). The algorithm then triggers the instantiation of the relevant domain models (line 4), leading to a recursive chain of updates. If the expanded dialogue state contains decision and utility variables, the algorithms searches for the optimal action, selects it, and activates the models that are triggered as a result (lines 6-8). Finally, the updated state is reduced by pruning away unnecessary nodes and incorporating the evidence (line 10).

---

**Algorithm 7 : UPDATESTATE ( $\mathcal{B}, \mathbf{X}$ )**


---

**Input:** Bayesian network  $\mathcal{B}$  for the current state

**Input:** New random variables  $\mathbf{X}$  to insert in the state

- 1: Initialise evidence  $e \leftarrow \emptyset$
  - 2: Insert  $\mathbf{X}'$  to the current state  $\mathcal{B}$
  - 3:  $\mathcal{B}, e \leftarrow \text{INTEGRATEPREDICTIONS}(\mathcal{B}, e, \mathbf{X}')$
  - 4:  $\mathcal{B}, e \leftarrow \text{TRIGGERMODELS}(\mathcal{B}, e, \mathbf{X}')$
  - 5: **while**  $\mathcal{B}$  contains decision variables **do**
  - 6:    $a^* \leftarrow \text{SELECTACTION}(\mathcal{B}, e)$
  - 7:   Assign  $\mathbf{A}' = a^*$
  - 8:    $\mathcal{B}, e \leftarrow \text{TRIGGERMODELS}(\mathcal{B}, e, \mathbf{A}')$
  - 9: **end while**
  - 10:  $\mathcal{B} \leftarrow \text{PRUNESTATE}(\mathcal{B}, e)$
- 

We now describe each of these steps in detail.

#### Connecting predictions and observations

Rules are sometimes used to provide predictions on variables that will be observed in the next time steps.<sup>8</sup> In order to distinguish random variables that express a prediction on a future outcome from those that reflect an actual (although possibly uncertain) observation, we denote predictive variables with a subscript  $p$ . A variable  $X_p$  represents therefore a random variable on the predicted values of the variable  $X$  to be observed in the future.

Prediction and observation variables must be connected with one another at runtime. In the case where the observation

<sup>8</sup>This is notably the case for the user action model  $P(a_u | i_u, a_m)$ , which estimate the relative probabilities for the next dialogue action from the user. The prediction provide a prior on the future observation of the user action.

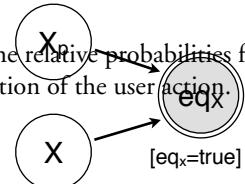


Figure 4.7: Equivalence node

is known with certainty, this connection can simply be represented as an assignment of evidence values. However, dialogue often include observations that are themselves uncertain and represent “soft” or virtual evidence. Several techniques are available to practically encode this evidence. The method adopted in this thesis is to add a new boolean-valued chance node, subsequently called the *equivalence node*  $eq_X$ , that is conditionally dependent on both  $X$  and  $X_p$ , as shown in Figure 4.7. The conditional probability distribution of  $eq_X$  is deterministic (graphically depicted by a double circle around the chance node):

$$P(eq_X = \text{true} | X = x, X_p = x_p) = \begin{cases} 1 & \text{if } x = x_p \\ 0 & \text{otherwise} \end{cases} \quad (4.12)$$

The use of a distinct node to express the evidence is motivated by the fact that  $X$  and  $X_p$  can have arbitrary incoming and outgoing edges with other variables.

The assignment  $eq_X = \text{true}$  is added to the evidence. The posterior distribution given the evidence allows the prediction to act as a prior for the observed distribution:

$$\begin{aligned} P(X = x | eq_X = \text{true}) \\ = \alpha P(X = x) \sum_{x_p \in Val(X_p)} P(eq_X = \text{true} | X = x, X_p = x_p) P(X_p = x_p) \end{aligned} \quad (4.13)$$

$$= \alpha P(X = x) P(X_p = x) \quad (4.14)$$

The inclusion of an equivalence node between  $X$  and  $X_p$  with evidence [ $eq_X = \text{true}$ ] modifies the distribution of the variables  $X$  and  $X_p$  as well as their respective parents/children nodes. Algorithm 8 illustrates the process of integrating predictions for the variables  $Vars$ .

---

**Algorithm 8 : INTEGRATEPREDICTIONS ( $\mathcal{B}, e, Vars$ )**


---

```

1: for all  $X \in Vars$  do
2:   if there is a corresponding prediction variable  $X_p \in \mathcal{B}$  then
3:     Create equivalence node  $eq_X$  with distribution in Eq. (4.12)
4:     Insert  $eq_X$  in  $\mathcal{B}$  with parents  $X$  and  $X_p$ 
5:     Add assignment [ $eq_X = \text{true}$ ] to evidence  $e$ 
6:   end if
7: end for
8: return  $\mathcal{B}, e$ 

```

---

## Model instantiation

After inserting the new variables in the dialogue state and connecting them to their predicted values, the next step in the processing workflow is to trigger the relevant domain models .

Algorithm 9 summarises the steps involved in the instantiation of the domain models. The algorithm takes three arguments: a dialogue state  $\mathcal{B}$  represented as a Bayesian network, an assignment

of evidence values and a list of random variables that have been recently updated in the dialogue state. The algorithm loops on all domain models and instantiates the ones that are triggered by the updated variables. The rules are instantiated one by one, following the procedure we have outlined in the previous section. Once all models are traversed, the output variables of the instantiated rules become updated variables themselves, and the procedure is repeated until no more models can be applied. To avoid the occurrence of infinite triggering cycles, models are limited to one instantiation per update. The algorithm returns both the dialogue state expanded with new variables, and the evidence assignment attached to the equivalence nodes.

---

**Algorithm 9 : TRIGGERMODELS ( $\mathcal{B}, \mathbf{e}, UpdatedVars$ )**


---

```

1: while  $UpdatedVars \neq \emptyset$  do
2:    $NewVars \leftarrow \emptyset$ 
3:   for all models  $m$  do
4:     if  $UpdatedVars \cap \mathcal{T}_m \neq \emptyset$  and  $m$  has not yet been applied then
5:       for all rule  $r \in \mathcal{R}_m$  do
6:         if  $r$  is a probability rule then
7:            $\mathcal{B} \leftarrow \text{INSTANTIATEPROBRULE}(\mathcal{B}, r)$ 
8:         else if  $r$  is a utility rule then
9:            $\mathcal{B} \leftarrow \text{INSTANTIATEUTILRULE}(\mathcal{B}, r)$ 
10:        end if
11:        Let  $\mathcal{O}_r$  be the new output variables created by rule  $r$ 
12:         $NewVars \leftarrow NewVars \cup \mathcal{O}_r$ 
13:         $\mathcal{B}, \mathbf{e} \leftarrow \text{INTEGRATEPREDICTIONS}(\mathcal{B}, \mathbf{e}, \mathcal{O}_r)$ 
14:      end for
15:    end if
16:   end for
17:    $UpdatedVars \leftarrow NewVars$ 
18: end while
19: return  $\mathcal{B}, \mathbf{e}$ 

```

---

### Action selection

Whenever the new dialogue state contains utility and decision nodes, the system must decide on the action to perform. Algorithm 10 illustrates how actions can be selected on the basis of the current dialogue state augmented with the decision and utility nodes created by the utility rules. The algorithm searches for the assignment of action values that maximise the current utility given the dialogue state and the evidence and returns it. This utility maximisation is based on standard inference algorithms for decision networks such as likelihood weighting (cf. Section 3.1.2).

The utility nodes are removed from the state once the decision is made. The action selection procedure described here only takes into account the current (immediate) utility and does not rely on any forward planning. Chapter 6 demonstrates how this procedure can be extended to perform online planning on a limited horizon.

---

**Algorithm 10** : SELECTACTION ( $\mathcal{B}, \mathbf{e}$ )

---

- 1: Let  $\mathbf{A}'$  be the set of all decision variables in  $\mathcal{B}$
  - 2: Find optimal value  $\mathbf{a}^* = \operatorname{argmax}_{\mathbf{a}} U(\mathbf{A}' = \mathbf{a}, \mathbf{e})$
  - 3: Remove utility nodes from the state  $\mathcal{B}$
  - 4: **return**  $\mathbf{a}^*$
- 

**State pruning**

The instantiation of the domain models results in the inclusion of numerous new nodes in the dialogue state. However, many nodes in this expanded Bayesian network only serve as intermediaries and do not directly express meaningful information about the current state of the dialogue. The last step is therefore to reduce the dialogue state to its minimal size, by removing all intermediary nodes – including rule nodes, outdated versions of state variables, equivalence nodes and predictive nodes that are attached to them – in order to only retain current state variables. The accumulated evidence is also integrated in the posterior distribution of the state variables.

The procedure is outlined in Algorithm 11. The first step is to determine which nodes to keep (line 1-6). Only the most recent versions of state variables are retained. The nodes are then added one by one in a new dialogue state  $\mathcal{B}'$ . The parents of each variable is determined, and its conditional probability distribution is calculated given the evidence. The parents of a state variable are the closest ancestors of the variable within the subset of nodes in  $NodesToKeep$ , and its conditional probability distribution is determined as  $P_{\mathcal{B}}(N | Parents, \mathbf{e})$ . This posterior distribution is calculated via sampling techniques. This is done by sampling all nodes in  $\mathcal{B}$ , then deriving the distributions  $P_{\mathcal{B}}(N | Parents, \mathbf{e})$  on the basis of the collected samples.

---

**Algorithm 11** : PRUNESTATE ( $\mathcal{B}, \mathbf{e}$ )

---

- 1:  $NodesToKeep \leftarrow \emptyset$
  - 2: **for all** node  $N \in \mathcal{B}$  **do**
  - 3:   **if**  $N$  is a state variable and  $\nexists N' \in \mathcal{B}$  **then**
  - 4:      $NodesToKeep \leftarrow NodesToKeep \cup [N]$
  - 5:   **end if**
  - 6: **end for**
  - 7: Create new state  $\mathcal{B}' \leftarrow \emptyset$
  - 8: **for all** node  $N \in NodesToKeep$  **do**
  - 9:   Add node  $N$  to  $\mathcal{B}'$  (with primes removed from node name)
  - 10:    $Parents \leftarrow \{M \in NodesToKeep : M \text{ is an ancestor of } N \text{ and there is a path } M \rightarrow^+ N \text{ without node in } NodesToKeep\}$
  - 11:   Add dependency edges between  $Parents$  and  $N$  in  $\mathcal{B}'$
  - 12:   Assign distributions  $P_{\mathcal{B}'}(N | Parents) \leftarrow P_{\mathcal{B}}(N | Parents, \mathbf{e})$
  - 13: **end for**
  - 14: **return**  $\mathcal{B}'$
- 

Figure 4.8 illustrates the input and output of the pruning process. Note that the primes attached to the labels of output variables are deleted from the random variable names.

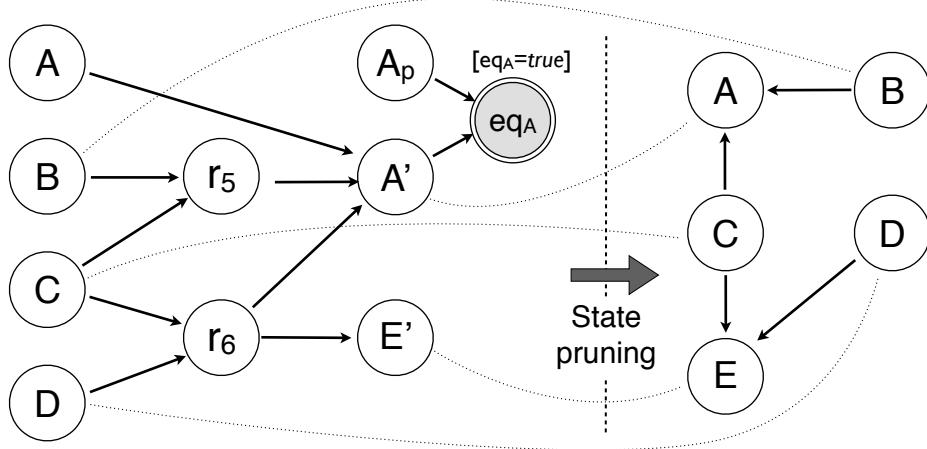


Figure 4.8: Illustration of the state pruning process. Only the nodes  $A'$ ,  $B$ ,  $C$ ,  $D$  and  $E'$  are retained. The dotted lines denote the correspondence between nodes.

#### 4.4.3 Detailed example

To make the above procedure more concrete, we describe below a minimal but complete example of workflow for a short interaction, detailing step-by-step how the dialogue state is practically updated and employed to select the system actions.

##### Description

Assume a domain similar to the one shown in Figure 2.3, where a user can request a robot to move forward, backward, left, right, or stop. The set of dialogue acts  $a_u$  that can be recognised by the system in this minimal example is the following:

$$\{Request(Forward), Request(Backward), Request(Left), \\ Request(Right), Request(Stop), Other\}.$$

The corresponding system actions  $a_m$  are:

$$\{Move(Forward), Move(Backward), Move(Left), \\ Move(Right), Stop, AskRepeat\}.$$

The objective of the system is to fulfil the user command if it is reasonably confident regarding which action to execute. Otherwise, the system asks the user to repeat.

##### Domain specification

The domain specification for this constructed example is constituted of an empty initial state and contains two rule-structured models  $m_1$  and  $m_2$ :

- Model  $m_1$  is triggered by  $a_u$  and includes the two utility rules  $r_9$  and  $r_{10}$ :

$$r_9 : \forall y :$$

$$\begin{aligned} & \text{if } (a_u = Request(y)) \text{ then} \\ & \quad \left\{ \begin{aligned} U(a_m = Move(y)) &= 2 \\ \text{else} & \\ & \quad \left\{ \begin{aligned} U(a_m = Move(y)) &= -2 \\ r_{10} : & \quad \left\{ \begin{aligned} U(a_m = AskRepeat) &= 0.5 \end{aligned} \right. \end{aligned} \right. \end{aligned} \right. \end{aligned}$$

Rule  $r_9$  specifies that the utility of executing the action corresponding to the user command is 2, with a penalty of  $-2$  when the wrong action is executed. Rule  $r_{10}$  assign a utility of 0.5 for asking a clarification question.<sup>9</sup>

- Model  $m_2$  is triggered by  $a_m$  and has one single predictive rule  $r_{11}$ :

$$r_{11} : \forall y :$$

$$\begin{aligned} & \text{if } (a_m = AskRepeat \wedge a_u = y) \text{ then} \\ & \quad \left\{ P(a_{u-p} = y) = 0.9 \right. \end{aligned}$$

Rule  $r_{11}$  specifies that the probability that the user will repeat his last utterance when asked by the system to do so is expected to be 0.9.

## Processing workflow

We now detail the processing workflow associated with the following constructed interaction:

USER : Now move forward

$$\tilde{a}_u = \langle (Request(Forward), 0.6), (Request(Backward)), 0.4 \rangle$$

SYSTEM : Could you please repeat?

USER : Please move forward!

$$\tilde{a}_u = \langle (Request(Forward), 0.7), (Other, 0.2), (Request(Backward), 0.1) \rangle$$

SYSTEM : OK, moving forward!

The (constructed) recognition hypotheses  $\tilde{a}_u$  produced by the ASR/NLU components are written underneath the user utterance.

Figure 4.9 details the steps involved in the state update procedure that follows from the reception of dialogue act hypotheses from the natural language understanding component.

Step 1 inserts the new dialogue act hypotheses on the dialogue state. This insertion triggers the utility model  $m_1$ . The instantiation results in Step 2 in the creation of two utility nodes and

---

<sup>9</sup>As the action selection process presented thus far does not perform forward planning, the utilities provided in this example correspond to long-term expected utilities (Q-values in the reinforcement learning terminology).

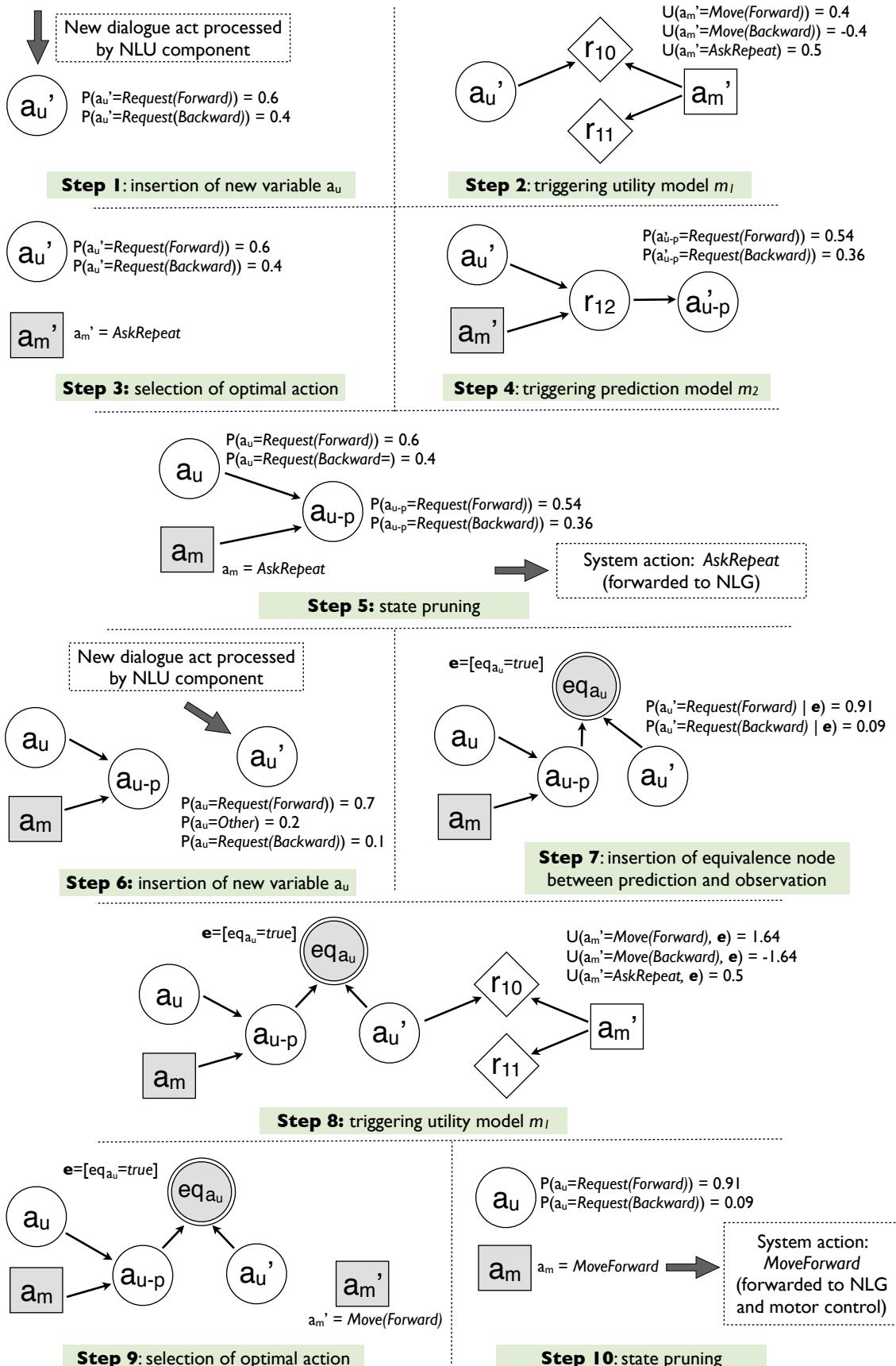


Figure 4.9: Detailed example of processing workflow.

one decision node. The optimal action to perform in such case is *AskRepeat*, which is selected by the system in Step 3. The action selection triggers model  $m_2$  in Step 4, which creates a prediction node  $a'_{u-p}$  expressing the expected probability distribution for the next user dialogue act. The state is finally pruned of the intermediary rule node in Step 5. System components such as NLG can react on the updated state and generate the proper linguistic realisation of the system action. The system then waits for the user input, which is shown in Step 6. The relation between the predicted and actual user response leads in Step 7 to the creation of an equivalence node, and the inclusion of the assignment  $eq_{a_u} = true$  in the evidence. We notice that the combination of the prior distribution over predicted values and the actual distribution over dialogue act hypotheses increases the probability of  $a'_u = Request(Forward)$ . Step 8 triggers the model  $m_1$  based on the new user input. The optimal action is this case is *Move(Forward)*, which is selected in Step 9. This selection triggers model  $m_2$ , but rule  $r_{11}$  only generates in this case an empty effect and is therefore directly deleted. Finally, the state is pruned of its intermediary nodes in Step 10, retaining only the last user and system actions  $a_u$  and  $a_m$ .

In comparison to the finite-state solution present in Figure 2.3, we observe that the rule-structured approach defined by models  $m_1$  and  $m_2$  allows the dialogue manager to accumulate evidence over time and prime the recognition hypotheses of the user dialogue act  $a_u$  based on the previous dialogue act. This accumulation of evidence is absent from the FSA, due its rigid state representation and lack of memory.

## 4.5 Advanced modelling

Dialogue domains often include random variables with values expressed via specific data structures such as lists or strings. The rule-based formalism described in the previous sections can be easily complemented with special-purpose tools to efficiently operate on these data structures. We first explain how conditions and effects can be defined on variables that represent lists, and then discuss how rules can manipulate strings.

### 4.5.1 Operations on lists

Some state variables are best represented as lists of elements. For instance, the dialogue state may include random variables that enumerate the  $n$  most recent dialogue acts in the interaction history, the stack of tasks that remain to perform, or the list of visual objects perceived by the system. The range of values for such state variables is the power set of its possible elements.

Special-purpose operators for the manipulation of such lists can be integrated in both the conditions and effects of probabilistic rules:

- Rule conditions can include operators to check the presence or absence of particular elements in a list, such as  $a \in A$  or  $a \notin A$ .
- Rule effects can also be augmented to manipulate elements from a list. Three new types of effects are created to this end, in addition to the traditional assignment of output values: *add effects* (adding an element to a list), *delete effects* (deleting an element from a list) and *clear effects* (clearing all elements of a list). The resulting lists are sorted by insertion order.

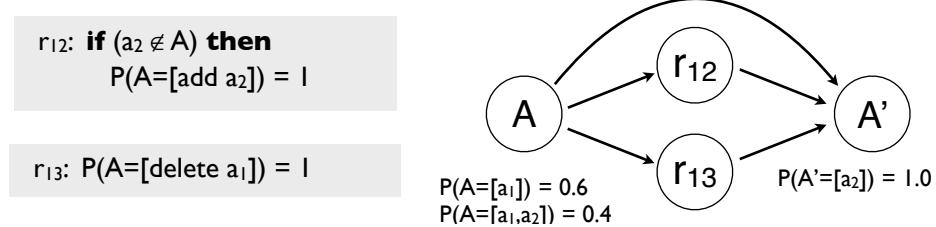


Figure 4.10: Example of rules using add/delete effects to manipulate lists.

Figure 4.10 illustrates two rules that apply these new effects to update a state variable  $A$ .

These new effects can be incorporated to the framework through a simple modification of the output distribution. Let  $e$  denote as before the conjunction of all effects  $e_1 \wedge \dots \wedge e_n$ . In addition to the previously defined set of values  $e(X)$  assigned for the variable  $X$ , we construct two new sets of values  $e_{add}(X)$  and  $e_{del}(X)$  that represent the values that are respectively added and deleted for the variable  $X$  through the new effects we just described. Note that  $e_{del}(X)$  may include all values for  $X$  if the clear effect is applied.

The output distribution in Equation (4.6) is then rewritten as:

$$P(X' = x' | r_1 = e_1, \dots, r_n = e_n, X = x) = \begin{cases} \frac{\sum_{v \in e(X)} \mathbf{1}(x' = v)}{|e(X)|} & \text{if } e(X) \neq \emptyset \\ \mathbf{1}(x' = (e_{add}(X) \cup (x / e_{del}(X)))) & \text{otherwise} \end{cases} \quad (4.15)$$

The output distribution associated for a new variable (cf. Equation (4.5)) can be rewritten in a similar manner.

## 4.5.2 Operations on strings

Many of the data structures present in the dialogue state are strings – the most prominent ones being the last user utterance  $u_u$  and the last system utterance  $u_m$ . The integration of special-purpose functionalities for manipulating strings within the conditions and effects of probabilistic rules is therefore desirable. In particular, rules can be extended to perform template-based string matching operations. The idea is to include a new type of conditions that checks whether a string matches a given template. Both full and partial matching can be employed. Templates are allowed to include slots to fill. These slots are conceptually similar to the quantified variables discussed in Sections 4.2.3 and 4.3.3. A successful match will thus generate values for the filled slots, which will be included as part of the groundings for the rule.

Figure 4.11 illustrate how such rules are applied in practice.  $\{OBJ\}$  denotes a slot that is to be filled through matching the template with the value specified in  $u_u$ .

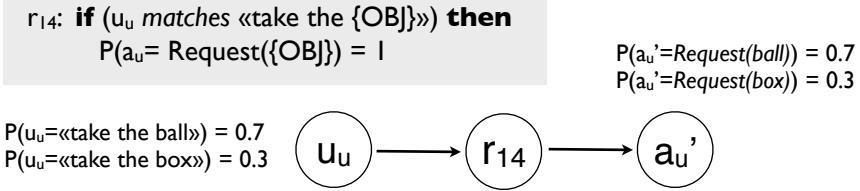


Figure 4.11: Example of rule using string matching operations.

## 4.6 Relation to previous work

The idea of using structural knowledge in probabilistic models has been explored in many directions, both in the fields of decision-theoretic planning and reinforcement learning (?????) and in statistical relational learning (??). The introduced structure may be hierarchical, relational, or both. As in our approach, most of these frameworks rely on the use of expressive representations serving as templates for the generation of classical probabilistic models. The surveys of ??) provide a complete overview of relational and first-order logical approaches for reinforcement learning in Markov decision processes, covering both model-free and model-based methods. While the formalisation presented in this thesis and the aforementioned approaches share many insights, they also reveal several interesting differences:

- Probabilistic rules are primarily tailored for dialogue management tasks and seek to capture dialogue domains by striking a balance between propositional and first-order logic. The formalism deliberately eschews the complexity of full-scale first-order probabilistic inference to ensure that the domain models can be applied under real-time constraints. This design choice sets it apart from other frameworks such as Markov Logic Networks which can express arbitrary first-order formulae but are often tedious to instantiate due to the size and complexity of the resulting models.<sup>10</sup>
- Probabilistic rules are also designed to operate under partially observable settings, as state uncertainty is a pervasive and unavoidable aspect of verbal interactions. By contrast, most previous work on relational probabilistic models are limited to fully observable environments, with the exception of some limited theoretical studies by ??).
- Finally, the presented framework posits that the *if ... then ... else* structures of probabilistic rules are best encoded by the system designers based on their expert knowledge of the domain, while the rule parameters can be estimated empirically. We therefore exclude the problem of structure learning from the scope of this thesis, as opposed to several approaches in which the domain rules and constraints are extracted via machine learning techniques (??).

Probabilistic rules also bear similarities with planning description languages such as the Planning Domain Description Language (PDDL, see ?) and its probabilistic extension, the Probabilistic Problem Description Language (PPDDL, see ?). These languages are structured through

---

<sup>10</sup>See however ?) for an approach that attempts to apply Markov Logic Networks to natural language understanding tasks.

action schemas that specify how (parametrised) actions can yield particular effects under various conditions. As in probabilistic rules, these languages try to carefully balance between the language expressivity and the complexity of the planning algorithm, based on a subset of first-order logic. A relational extension of PDDL, named RDDL, has also been introduced in recent planning competitions (?). The learning techniques presented by ?) to estimate transition functions based on noisy indeterministic deictic rules is directly related to our approach, as is the recent work ?) on probabilistic noisy planning rules. Both frameworks define conditions associated with probabilistic distributions over effects. Their approaches are however restricted to fully observable settings.

In the dialogue management literature, most structural approaches rely on a clear-cut task decomposition into goals and sub-goals (??), where the completion of each goal is assumed to be fully observable, discarding any remaining uncertainty. Our own work on multi-policy dialogue management in ?) relaxes the assumption of perfect knowledge of task completion, handling multiple policies as a problem of probabilistic inference over activation variables. Probabilistic rules can be considered an extension of this early work, where the structural knowledge is not confined to task decomposition but is extended to generic rules over state variables.

The formalism presented in this chapter is strongly inspired by information-state approaches to dialogue management (??), which are also based on a shared state representation that is updated according to a rich repository of rules. ?) also models conversational phenomena by way of update operations that are encoded with rules mapping conditions to effects. However, contrary to the framework presented here, the rules specified in these approaches are generally deterministic and do not include learnable parameters. The action selection mechanism is also conceptualised slightly differently, as information-state frameworks rely on rules that directly select the most appropriate action given the current state. Probabilistic rules adopt by contrast a decision-theoretic approach that decomposes action selection in two stages: the instantiation of utility rules first determines the utility distributions associated with the possible system actions, and the system then searches for the action that yields the maximum expected utility on the basis of these distributions.

The literature on dialogue policy optimisation with reinforcement learning also contains several approaches dedicated to dimensionality reduction for large state-action spaces, such as function approximation (?), hierarchical reinforcement learning (?) and summary POMDPs (?). Many of these techniques have already been discussed in Section 3.3 and will therefore not be repeated here. Most current approaches in dialogue policy optimisation focus on large but weakly structured state spaces (generally encoded as large lists of features), which are suited for slot-filling dialogue applications but are difficult to transfer to more open-ended or relational domains. The idea of state space partitioning, implemented here via high-level conditions, has also been explored in recent papers (see e.g. ?). ?) explored the introduction of complex user goal states including disjunction and negation operators. ?) describes a policy optimisation approach based on logic-based representations of the state-action space for relational MDPs. The main difference with our approach lies in his reduction of the belief state to fully observable variables whereas we retain the partial observability associated with each variable. The work of ??) demonstrated how tree-structured Bayesian networks called probabilistic ontology trees can improve belief tracking performance. The tree structure is derived in their work from a hierarchical concept structure . Finally, ?) describe a procedure for the selection of dialogue strategies based on probabilistic logic programming.

## 4.7 Conclusion

This chapter presented the formalism of probabilistic rules, which forms the core of the modelling approach developed in this thesis. We started by arguing that dialogue models are often highly structured, and that this structure can be leveraged by introducing latent variables, partitioning value assignments for the parent variables, and making use of quantification. We then explained how these structural insights can be transferred into a new framework – probabilistic rules – that combines concepts borrowed from both first-order logic and probability theory in order to get “the best of both worlds”, i.e. a representation formalism that is both richly expressive and capable of capturing uncertain knowledge. These rules are practically defined as *if...then...else* control structures that associate high-level conditions on input variables to probabilistic effects on output variables. Multiple extensions of the formalism have been developed to e.g. encode utility distributions, enclose universal quantifiers, and efficiently manipulate data structures such as lists and templates.

At runtime, these rules are instantiated in the Bayesian network representing the current dialogue state. The instantiation procedure creates a latent node for each rule, which is conditionally dependent on the input variables of the rule. For probability rules, this node is a chance node that expresses a probability distribution over the possible effects of the rule. Utility rules are similarly instantiated with a utility node expressing the utility distribution for specific decision variables. Universally quantified variables can be included in the conditions and effects of the rules, allowing particular aspects of the rule to be underspecified. The rules are grouped into models that are attached to the dialogue state and are triggered upon relevant state updates.

Probability and utility rules effectively function as high-level templates for the definition of a dynamic decision network. The expressive power of these rules allows them to efficiently encode complex relations between variables, and thereby reduce the number of parameters to estimate. We have however not yet detailed how this parameter estimation is practically performed. The next two chapters provide answers to this important question.

# Chapter 5

## Learning from Wizard-of-Oz data

The previous chapter outlined the formalism of probabilistic rules and their instantiation in the dialogue state. Probabilistic rules are generally associated to a number of parameters, which may either express probabilities over effects (for probability rules) or utilities associated with particular decisions (for utility rules). But where do these probabilities and utilities exactly come from?

We have developed in this thesis two distinct approaches to this question. The present chapter concentrates on the first approach, which is grounded in supervised learning techniques. We demonstrate how rule parameters can be optimised to best imitate the decision choices of a human expert through a process of statistical estimation based on Wizard-of-Oz data. The next chapter will then detail an alternative, reinforcement learning approach to the same task.

The chapter is divided in three sections. Section 5.1 describes how uncertainty regarding the value of rule parameters can be explicitly represented through prior distributions. Section 5.2 goes on to explain how these distributions can be gradually refined through Bayesian learning on data gathered from Wizard-of-Oz interactions. The learning algorithm is used to progressively narrow down the spread of the parameter distributions to the values providing the best fit for the training data. Finally, Section 5.3 presents experimental results in a human-robot interaction domain for which small amounts of Wizard-of-Oz data were recorded. The experiment compared the learning performance of a utility model structured with probabilistic rules against two baselines respectively encoded with plain utility tables and with linear models. The evaluation showed that the rule-structured model was able to imitate the dialogue policy followed by the wizard significantly better than its unstructured counterparts.

### 5.1 Parameters of probabilistic rules

#### 5.1.1 Generalities

The examples of probabilistic rules seen so far all relied on fixed probability and utility values. These values can however be replaced by parameters that reflect unknown values that are to be estimated empirically, on the basis of training data.

The overall structure of parametrised rules remains essentially identical to the one outlined in the previous chapter. Parametrised probability rules are once more defined in terms of conditions  $c_i$  associated to probability distributions  $P(E_i)$  over effects. The probability of each effect  $e_{(i,j)}$  is however no longer fixed but is instead represented by a parameter  $\theta_{(i,j)}$ , giving rise to the following

rule skeleton:

$$\begin{aligned}
 & \textbf{if } (c_1) \textbf{ then} \\
 & \quad \left\{ \begin{array}{l} P(E_1 = e_{(1,1)}) = \theta_{(1,1)} \\ \dots \\ P(E_1 = e_{(1,m_1)}) = \theta_{(1,m_1)} \end{array} \right. \\
 & \quad \dots \\
 & \textbf{else} \\
 & \quad \left\{ \begin{array}{l} P(E_n = e_{(n,1)}) = \theta_{(n,1)} \\ \dots \\ P(E_n = e_{(n,m_n)}) = \theta_{(n,m_n)} \end{array} \right.
 \end{aligned} \tag{5.1}$$

As the  $\theta$  values represent probability values, they must satisfy as before the two probability axioms  $\theta_{(i,j)} \geq 0 \forall i, j$  and  $\sum_{j=1}^{m_i} \theta_{(i,j)} = 1 \forall i$ .

Parametrised utility rules analogously replace utility values with unknown parameters:

$$\begin{aligned}
 & \textbf{if } (c_1) \textbf{ then} \\
 & \quad \left\{ \begin{array}{l} U_1(d_{(1,1)}) = \theta_{(1,1)} \\ \dots \\ U_1(d_{(1,m_1)}) = \theta_{(1,m_1)} \end{array} \right. \\
 & \quad \dots \\
 & \textbf{else} \\
 & \quad \left\{ \begin{array}{l} U_n(d_n^1) = \theta_{(n,1)} \\ \dots \\ U_n(d_{(n,m_n)}) = \theta_{(n,m_n)} \end{array} \right.
 \end{aligned} \tag{5.2}$$

We shall focus in this work on the problem of parameter estimation given a known rule structure. The methodological stance adopted in this thesis rests on the idea that the structure of probabilistic rules is best defined by the system designer, while the rule parameters is best determined by statistical optimisation techniques. Based on our own practical experience with various dialogue systems, we believe that such division of labour between the human designer and the learning algorithm is a sensible one, as system designers generally have a good grasp of the domain structure and relations between variables, but are often unable to quantify the precise probability of an effect or utility of an action.<sup>1</sup>

It should nevertheless be noted that some approaches have been recently developed in the literature on statistical relational learning (see e.g. ??) to automatically extract both the structure and parameters of stochastic rules from raw data. These methods are however generally confined

---

<sup>1</sup>Humans are indeed notoriously poor at estimating probabilities and are prone to multiple cognitive biases when doing so, as evidenced by numerous studies in behavioural psychology. The interested reader is invited to consult e.g. ?? for more details on the psychological aspects of the human perception of uncertainty and the difficult problem of probability elicitation from experts.

to domains of limited size and full observability and are therefore difficult to apply to the types of domains investigated in this thesis.

### 5.1.2 Parameter priors

We follow in this thesis a Bayesian approach to parameter estimation and associate the rule parameters with explicit prior distributions over their range of possible values. The benefits of Bayesian approaches compared to traditional maximum likelihood methods are multiple:

1. Bayesian methods allow domain knowledge to be directly included into the learning cycle through the use of informative priors (cf. discussion below). The system designer is thus free to bias the initial prior distributions according to her/his domain expertise.
2. Bayesian methods explicitly capture the model uncertainty both before and after learning. The outputs of a Bayesian learning cycle are indeed full posterior distributions over the possible parameter values instead of being reduced to point estimates (as for maximum likelihood estimation). The dialogue agent can therefore explicitly account for parameter uncertainty at runtime and use it to simultaneously learn, reason and act in its environment (?). Furthermore, the reliance on full parameter distributions facilitates the combination of multiple learning processes, as the posterior distribution generated by one learner can be passed on as the prior distribution of another.

Prior parameter distributions have a continuous range of values and can be either univariate or multivariate. We first review prior parameter distributions for probability rules and then discuss the case of utility rules.

#### Probability parameters

The distributions over the effects of probability rules are categorical probability distributions. As discussed in Section 3.1.3, the parameter priors of categorical and multinomial distributions are best expressed using *Dirichlet* distributions.

Each distribution over effects  $P(E_i)$  is associated with its own Dirichlet distribution of dimension  $m_i$ , where  $m_i$  is the number of alternative effects (including the empty effect if appropriate). Rule  $r_1$  provides an example of parametrised probability rule:

$$r_1 : \begin{aligned} & \textbf{if } (Rain = \text{false} \wedge Weather = \text{hot}) \textbf{ then} \\ & \quad \begin{cases} P(Fire = \text{true}) = \theta_{r_1(1,1)} \\ P(Fire = \text{false}) = \theta_{r_1(1,2)} \end{cases} \\ & \textbf{else} \\ & \quad \begin{cases} P(Fire = \text{true}) = \theta_{r_1(2,1)} \\ P(Fire = \text{false}) = \theta_{r_1(2,2)} \end{cases} \end{aligned}$$

The parameters of rule  $r_1$  can be expressed with two Dirichlets  $\boldsymbol{\theta}_{r_1(1,\cdot)} = \langle \theta_{r_1(1,1)}, \theta_{r_1(1,2)} \rangle$  and  $\boldsymbol{\theta}_{r_1(2,\cdot)} = \langle \theta_{r_1(2,1)}, \theta_{r_1(2,2)} \rangle$  respectively associated with the effect distributions for the first and

second condition. The Dirichlet distributions in this example have two dimensions (since two alternative effects are mentioned in the rule), which make them equivalent to *Beta* distributions.

Dirichlet distributions are continuous, multivariate distributions defined by the meta-parameters  $\boldsymbol{\alpha} = [\alpha_1, \dots, \alpha_k]$ , where  $k$  corresponds to the dimensionality of the categorical distribution of interest. Dirichlet distributions over the parameters  $\theta_1, \dots, \theta_k$  are formally expressed by the following probability density function:

$$P(\theta_1, \dots, \theta_k; \boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{i=1}^k \theta_i^{\alpha_i-1} \quad \text{with } B(\boldsymbol{\alpha}) = \frac{\prod_{i=1}^k \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^k \alpha_i)} \quad (5.3)$$

where  $B(\boldsymbol{\alpha})$  serves as normalisation factor and builds upon on the gamma function  $\Gamma$ .<sup>2</sup> We remind the reader that the notation  $P(\theta_1, \dots, \theta_k; \boldsymbol{\alpha})$  refers to the joint density function of the random variables  $\theta_1, \dots, \theta_k$  given the specified (hyper-)parameters  $\boldsymbol{\alpha}$ .

Figure 5.1 depicts the shape of the probability density functions for several variants of the two-dimensional Dirichlet distribution  $P(\theta_1, \theta_2; \boldsymbol{\alpha})$ . The second dimension  $\theta_2$  is not explicitly shown on the figure but can be directly derived from the first dimension, since  $\theta_1 + \theta_2 = 1$ . The figure illustrates how the  $\boldsymbol{\alpha}$  hyper-parameters determine the shape of the distribution. As the  $\boldsymbol{\alpha}$  counts grow larger, the density function becomes increasingly focused on a particular region of the parameter space. The  $\boldsymbol{\alpha}$  counts can therefore be tuned to skew the distribution in a particular way based on prior domain knowledge. Such prior distributions are called “informative” priors, since their shape is influenced by expert information. In the absence of such information, non-informative distributions such as Dirichlet(1, 1) can also be employed.

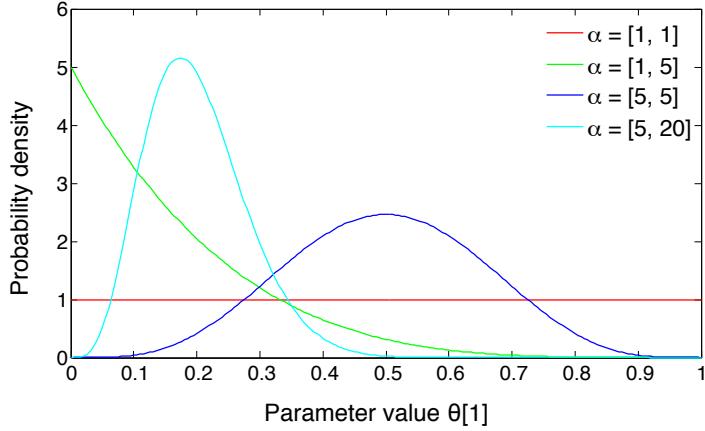


Figure 5.1: Probability density functions for the Dirichlet distribution  $P(\theta_1, \theta_2; \boldsymbol{\alpha})$  with various values for the  $\boldsymbol{\alpha}$  hyper-parameters.

As Dirichlet distributions are conjugate priors of categorical distributions, their posterior distribution after the observation of their corresponding variable remains a Dirichlet distribution with updated counts. As an illustrative example, the posterior distribution of  $\theta_{r_1(1,\cdot)}$  after observing a

<sup>2</sup>The gamma function is a generalisation of the factorial for real numbers, and is defined as  $\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt$ .

fire when  $Rain = \text{false} \wedge Weather = \text{hot}$  can be derived from Bayes' rule:

$$\begin{aligned} P(\boldsymbol{\theta}_{r_1(1,\cdot)} \mid Fire = \text{true}, Rain = \text{false}, Weather = \text{hot}) \\ = \eta P(Fire = \text{true} \mid Rain = \text{false}, Weather = \text{hot}, \boldsymbol{\theta}_{r_1(1,\cdot)}) P(\boldsymbol{\theta}_{r_1(1,\cdot)}) \\ = \eta \theta_{r_1(1,1)} \text{Dirichlet}(\alpha_1, \alpha_2) \\ = \eta' \theta_{r_1(1,1)} [\theta_{r_1(1,1)}^{\alpha_1-1} \times \theta_{r_1(1,2)}^{\alpha_2-1}] = \eta' \text{Dirichlet}(\alpha_1 + 1, \alpha_2) \end{aligned}$$

where  $\eta, \eta'$  are normalisation factors. Given a prior distribution  $P(\boldsymbol{\theta}_{r_1(1,\cdot)}) \sim \text{Dirichlet}(\alpha_1, \alpha_2)$ , the posterior distribution for  $\boldsymbol{\theta}_{r_1(1,\cdot)}$  after the observation  $Fire = \text{true}$  is thus another Dirichlet distribution  $\sim \text{Dirichlet}(\alpha_1 + 1, \alpha_2)$ . As explained in Section 3.1.3, this property is however contingent on the full observability of the domain variables.

## Utility parameters

The parameters of utility rules are also defined by probability density functions. However, contrary to probability values, the values in a utility distribution are independent of one another and need not satisfy the probability axioms (their range of possible values is arbitrary). Each utility value  $u_{(i,j)}$  assigned to decision  $d_{(i,j)}$  is therefore associated with its own, univariate distribution.

Rule  $r_2$  illustrates a utility rule with four independent parameters:

$$\begin{aligned} r_2 : & \text{if } (Fire = \text{true}) \text{ then} \\ & \begin{cases} U(Tanker = \text{drop-water}) = \theta_{r_2(1,1)} \\ U(Tanker = \text{wait}) = \theta_{r_2(1,2)} \end{cases} \\ & \text{else} \\ & \begin{cases} U(Tanker = \text{drop-water}) = \theta_{r_2(2,1)} \\ U(Tanker = \text{wait}) = \theta_{r_2(2,2)} \end{cases} \end{aligned}$$

Several types of density functions can be applied to define the prior distributions over these utility values. This thesis concentrates on two specific families of priors, one non-informative (uniform distributions) and one informative (normal distributions):

1. Continuous uniform distributions are defined on an interval  $[a, b]$  which corresponds to the allowed range of utility values, and have the following density:

$$P(\theta ; a, b) = \begin{cases} \frac{1}{b-a} & \text{for } \theta \in [a, b] \\ 0 & \text{otherwise} \end{cases} \quad (5.4)$$

2. Normal (also called Gaussian) distributions are defined by a probability density function revolving around a mean  $\mu$  and variance  $\sigma^2$ :

$$P(\theta ; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left\{ -\frac{(\theta - \mu)^2}{2\sigma^2} \right\} \quad (5.5)$$

The range of possible values can be further constrained by truncating the density function.

Normal distributions are well suited to represent utility values for which rough initial estimates are available. If a particular utility value is expected to lie in the vicinity of a particular value, its probability distribution can be expressed via a normal distribution with a mean centered on this value and a variance reflecting the confidence in the provided estimate.

Figure 5.2 illustrates three instances of probability density functions for a parameter  $\theta$ . The first distribution corresponds to a uniform distribution on the interval  $[-2, 4]$ , while the second and third distributions are truncated normal distributions with mean  $\mu = 2$  and variances respectively assigned to  $\sigma^2 = 4$  and  $\sigma^2 = 1$ . The normal distributions illustrate how prior knowledge about the utility value can be incorporated in the prior – in this case, the distributions rest on the assumption that the true utility value is likely to revolve around the value 2.

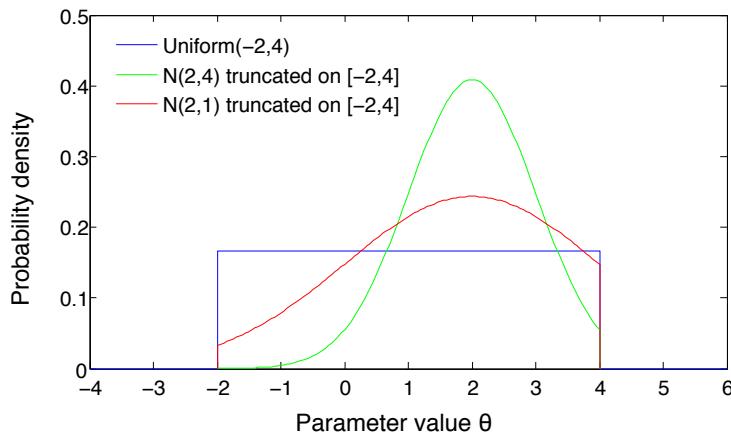


Figure 5.2: Probability density functions  $P(\theta)$  over the interval  $[-2, 4]$  using an uniform distribution, a truncated normal distribution  $\mathcal{N}(2, 4)$  and a truncated normal distribution  $\mathcal{N}(2, 1)$ .

### 5.1.3 Instantiation

Parameters are instantiated in the dialogue state as distinct chance nodes. One chance node is created for each (univariate or multivariate) parameter distribution and included as parents of its corresponding rule node. The rule distributions must be slightly adapted to factor these parameters in the parent nodes of the rule. The rest of the instantiation process remains unchanged.

#### Parameters of probability rules

A parametrised probability rule  $r$  structured with  $n$  conditions is associated with  $n$  multivariate parameter nodes  $\boldsymbol{\theta}_{r(1,\cdot)}, \dots, \boldsymbol{\theta}_{r(n,\cdot)}$ . Each distribution  $P(\boldsymbol{\theta}_{r(i,\cdot)})$  is a Dirichlet distribution of dimension  $m_i$ , where  $m_i$  is the number of effects associated with the condition  $c_i$  (including the empty effect). Figure 5.3 illustrates this instantiation procedure with two probability rules.

The conditional probability distribution of a rule node  $r$  given its input variables  $I_1, \dots, I_k$  and parameters  $\boldsymbol{\theta}_{r(1,\cdot)}, \dots, \boldsymbol{\theta}_{r(n,\cdot)}$  is a straightforward adaptation of Equation (4.4):

$$P(r=e | I_1=i_1, \dots, I_k=i_k; \boldsymbol{\theta}_{r(1,\cdot)}, \dots, \boldsymbol{\theta}_{r(n,\cdot)}) = P(E_i = e; \boldsymbol{\theta}_{r(i,\cdot)}) \quad (5.6)$$

where  $i = \min_i(c_i \text{ is satisfied with } I_1=i_1 \wedge \dots \wedge I_k=i_k)$

## Parameters of utility rules

The parameters of utility rules are instantiated in a similar manner, as shown in Figure 5.4. The corresponding utility distribution is adapted from Equation 5.7 as follows:

$$\begin{aligned}
 U_r(I_1=i_1, \dots, I_k=i_k, A_1=a_1, \dots, A_l=a_l; \theta_{r(1,\cdot)}, \dots, \theta_{r(n,\cdot)}) \\
 = U_i(A_1=a_1, \dots, A_l=a_l; \theta_{r(i,\cdot)}) \\
 \text{where } i = \min_i(c_i \text{ is satisfied with } I_1=i_1 \wedge \dots \wedge I_k=i_k)
 \end{aligned} \tag{5.7}$$

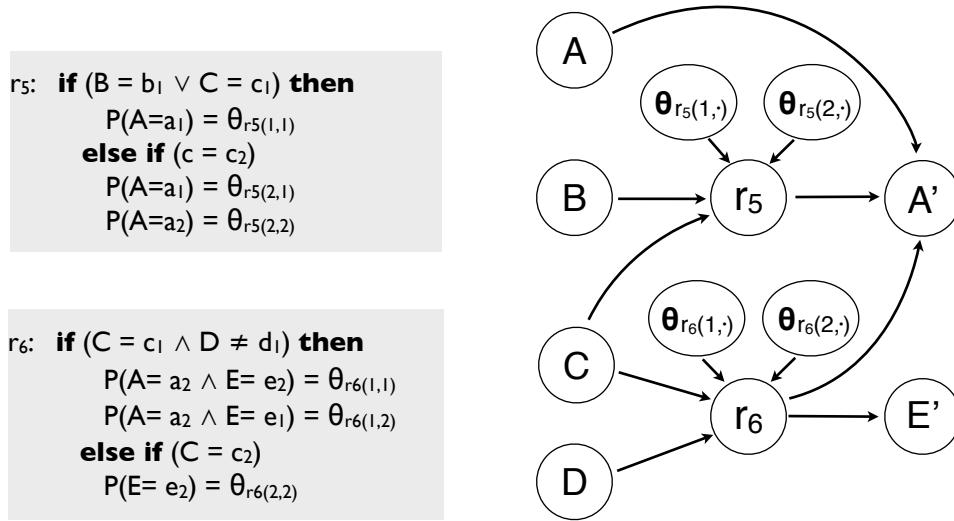


Figure 5.3: Example of instantiation for two parametrised probability rules  $r_5$  and  $r_6$ .

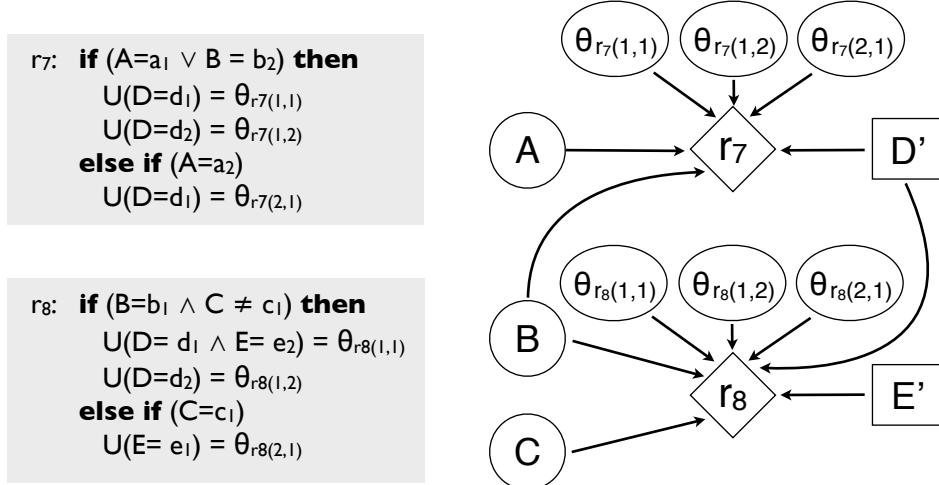


Figure 5.4: Example of instantiation for two parametrised utility rules  $r_7$  and  $r_8$ .

## 5.2 Supervised learning of rule parameters

Once the rule parameters are instantiated as nodes in the dialogue state, one can calculate their posterior distribution  $P(\theta | \mathcal{D})$  after observing a particular data set  $\mathcal{D}$  using standard algorithms for probabilistic inference. The estimation of rule parameters corresponds to a learning problem with partial data (cf. Section 3.1.3), since some of the nodes in the dialogue state (such as rule nodes) are not directly observed. The posterior distribution  $P(\theta | \mathcal{D})$  is thus no longer guaranteed to remain in the same distribution family as their prior distributions. The solution adopted here is to approximate the parameter distributions via sampling techniques.

The estimation of rule parameters from data is practically achieved by cycling through the data sample one after the other and gradually refining the parameter distributions in the light of the observed values. We describe in the next pages the generic representation of the training data, the as well as the learning algorithm that exploits them for optimising the rule parameters of the domain given some reasonable assumptions about the wizard behaviour.

### 5.2.1 Wizard-of-Oz training data

This chapter focus on parameter estimation from a specific type of training data, namely Wizard-of-Oz interactions. Wizard-of-Oz interactions are interactions between human users and a dialogue system which are remotely controlled by a human expert “behind the curtains”. They constitute a simple and efficient method to collect realistic conversational behaviour for a particular domain in the absence of a fully implemented or optimised system.<sup>3</sup>

#### Representation format

For the specific purpose of estimating the parameters of dialogue management models, we can represent Wizard-of-Oz interactions as a sequence of state-action pairs  $\mathcal{D} = \{(\mathcal{B}_i, a_i) : 1 \leq i \leq n\}$ , where  $\mathcal{B}_i$  corresponds to the dialogue state at time  $i$ , and  $a_i$  is the associated action performed by the wizard. The number  $n$  corresponds to the total number of recorded actions.

The dialogue state  $\mathcal{B}_i$  represents the current conversational situation at time  $i$  as it was perceived by the wizard. Its representation usually includes the recent dialogue history as well as important contextual features. As explained in the previous chapter, the dialogue state can be encoded as a Bayesian network to reflect state uncertainty and dependences amongst state variables. Associated to each dialogue state is the corresponding action  $a_i$  selected by the wizard at that state. This action can be void if the wizard decides to take no action at that specific step in the dialogue.

Many conversational situations allow for multiple, equally “correct” system responses. This characteristic of verbal interactions transpires in the state-action pairs of Wizard-of-Oz data sets, as one can occasionally observe similar states mapped to different wizard actions. The wizard actions should therefore be viewed as an indication of good conversational behaviour, but do not constitute absolute gold standards in the traditional sense of being the uniquely appropriate output for the given dialogue state. The existence of multiple responses also entails that the accuracy of the learned models remains contingent on the degree of internal consistency of the wizard actions.

---

<sup>3</sup>Developing spoken dialogue systems can indeed lead to a classical “chicken-and-egg” dilemma: in order to build up a particular system, system designers often need to know what types of user utterances and behaviours are expected – but in order to collect such data, one must first have an integrated dialogue system with which the users can interact. Wizard-of-Oz interactions are a way to circumvent this dilemma.

## Assumptions about the wizard behaviour

Parameter estimation on Wizard-of-Oz data rests on the assumption that the wizard is a rational agent and will tend to select actions that are deemed most useful in their respective dialogue state. It should be stressed that the agent is only assumed to act rationally *given* the perceived (uncertain) dialogue state. The wizard must indeed act on the basis of “noisy” inputs (including e.g. speech recognition errors) and may as a consequence select suboptimal actions if the provided inputs contain erroneous hypotheses. It should hence be stressed that the assumption of rationality does not equate to an assumption of omniscience on the part of the wizard.

Practically, the presupposed rationality of the wizard implies that the likelihood  $P_{\mathcal{B}_i}(a_i ; \boldsymbol{\theta})$  of a wizard action  $a_i$  in a particular dialogue state  $\mathcal{B}_i$  under the parameters  $\boldsymbol{\theta}$  will be proportional to the corresponding utility of action  $a_i$  in  $\mathcal{B}_i$  relative to other actions (as formalised below).

### 5.2.2 Learning cycle

The goal of the learning process is to estimate the posterior distribution  $P(\boldsymbol{\theta} | \mathcal{D})$  over the rule parameters given the collected Wizard-of-Oz data set. The procedure operates in an incremental fashion by traversing the (state,action) pairs one by one and re-estimating the posterior distribution after each pair.

#### Likelihood distribution

One key element of the learning cycle is the definition of the probability  $P_{\mathcal{B}_i}(a_i ; \boldsymbol{\theta})$ , which specifies the likelihood of the wizard action  $a_i$  in a dialogue state  $\mathcal{B}_i$  given the parameters  $\boldsymbol{\theta}$ . The purpose of this likelihood is intuitively to favour the parameter values that provide a good fit for the wizard action choices.

In practice, this is achieved by representing the likelihood of the wizard action  $a_i$  in terms of the relative utility of action  $a_i$  compared to the alternative action choices. The first step is to calculate the utility  $U_{\mathcal{B}_i}(a ; \boldsymbol{\theta})$  for all possible actions  $a$ , and to subsequently rank the actions in descending order of utility. As the wizard is presupposed to act rationally in most cases, the wizard action will thus most likely appear at the top of this ranked list of actions. Formally, the likelihood of action  $a_i$  given the parameters  $\boldsymbol{\theta}$  is expressed as:

$$P_{\mathcal{B}_i}(a_i ; \boldsymbol{\theta}) = \begin{cases} \alpha p & \text{if } a_i \text{ is the action with highest utility} \\ \alpha(1-p)p & \text{if } a_i \text{ is the second-highest action} \\ \alpha(1-p)^2p & \text{if } a_i \text{ is the third-highest action} \\ \dots & \end{cases}$$

$$= \alpha(1-p)^x p \quad \text{where } x \text{ is the position of action } a_i \text{ in the ranked list of actions} \quad (5.8)$$

The  $\alpha$  factor in Eq. (5.8) represents as usual the normalisation factor, while the probability  $p$  represents the learner confidence in the judgement of the wizard. A value  $p = 0.7$  will hence indicate that the wizard is assumed to act rationally – and thus select the highest-utility action – with probability 0.7. The probability  $p$  indirectly defines the learning rate of the estimation process:

the higher the probability, the faster the learner will converge to a policy that imitate the wizard actions. A high value for  $p$  makes the learner however more vulnerable to the occasional errors and inconsistencies on the part of the wizard.

The probability distribution represented by Equation (5.8) is a particular instance of a *geometric* distribution.<sup>4</sup> Geometric distributions correspond the number of Bernoulli trials required to get one success. The reliance on geometric distributions to formalise the likelihood of the observed actions is motivated by the posited rationality of the wizard decisions. In other words, the formalisation is grounded in the assumption that the wizard will select the highest utility action with probability  $p$ , the second-highest action with probability  $(1 - p)p$ , and so forth. The geometric distribution is monotonically decreasing, which ensures that the probability of a high-utility action will always be higher than its lower-utility alternatives. Figure 5.5 illustrates an example of likelihood distribution over four action values  $[a_1, a_2, a_3, a_4]$ , where  $U_{\mathcal{B}_i}(a_1; \theta) > U_{\mathcal{B}_i}(a_2; \theta) > \dots > U_{\mathcal{B}_i}(a_4; \theta)$ .

## Posterior parameter distribution

Given the aforementioned likelihood distribution, the posterior distribution over the parameters is defined via Bayes' rule:

$$P_{\mathcal{B}_i}(\theta | a_i) = \eta P_{\mathcal{B}_i}(a_i; \theta) P(\theta) \quad (5.9)$$

The factor  $\eta$  is used for normalisation. The value range for parameter distributions is typically continuous and has therefore an infinite number of possible values. Inference in hybrid graphical models featuring both continuous and discrete variables is known to be a non-trivial problem. Two types of solutions can be distinguished:

- The first strategy is to discretise the range of parameter values into distinct, mutually exclusive buckets, and thereby transform continuous variables into discrete variables, with a number of values equivalent to the number of buckets employed for the discretisation.
- The second strategy is to retain the continuous nature of the parameters, but approximate the inference process through the use of sampling techniques.

Although both solutions are implemented in the openDial toolkit, sampling techniques such as likelihood weighting have proved to be in practice more efficient and scalable than discretisation.

---

<sup>4</sup>Technically speaking, the definition in Equation (5.8) corresponds to a truncated version of the geometric distribution, as the support of the distribution is finite (the number of possible actions is bounded). The reduction of the distribution to a finite support is achieved via the normalisation factor  $\alpha$ .

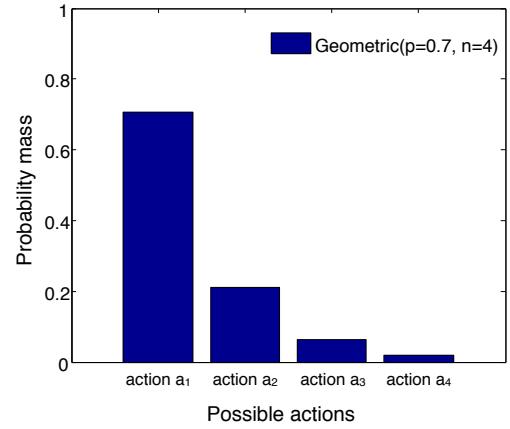


Figure 5.5: Distribution  $P_{\mathcal{B}_i}(a_i; \theta)$  for four action values  $[a_1, a_2, a_3, a_4]$  ranked by decreasing utility, with  $p = 0.7$ .

After sampling, the full joint distribution  $P_{\mathcal{B}_i}(\boldsymbol{\theta} \mid a_i)$  is factored into its individual parameter variables. This factoring is a simplifying assumption, since parameter independence is theoretically no longer guaranteed when handling partially observed data.

### Representation of the posterior

The result of the posterior calculation in Equation (5.9) is a collection of sampled values. Two approaches are possible to define probability density functions from these samples. One can either follow a so-called parametric approach and seek to reconstruct the underlying parametric distributions (such as Gaussian or Dirichlet distributions) that best fit the data, or adopt a non-parametric strategy and directly represent the posterior as a function of the collected samples. Parametric approaches, albeit interesting, are difficult to apply in this setting, for two main reasons:

1. The distribution family of the posterior is hard to determine, as the posterior is no longer ensured to remain in the same family as the prior when learning from partial data.
2. Additionally, fitting multivariate distributions such as Dirichlets based on sampled values is a laborious computational process with no closed-form solutions.<sup>5</sup>

We have consequently adopted a non-parametric representation of the posterior distributions, based on *Kernel Density Estimation* (KDE). The kernel density estimator for a continuous variable  $X$  for which a set of samples  $x_1, \dots, x_n$  is available is given by:

$$P(x) = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x - x_i}{h}\right) \quad (5.10)$$

where  $K(\cdot)$  is a *kernel function* and  $h$  is a smoothing parameter called the *bandwidth*. Multiple kernel functions can be used, but a common choice is to adopt a Gaussian kernel. The kernel density estimator corresponds in this case to a combination of  $n$  Gaussians, where each Gaussian is centered on a sample point  $x_i$ . This combination of Gaussians is smoothed proportionally to the bandwidth parameter. Figure 5.6 illustrate the use of kernel density estimation for a continuous variable based on a set of 50 samples and a Gaussian kernel. The figure shows the influence of the bandwidth parameter on the shape of the resulting density function. Kernel density estimators do not necessitate any particular assumption about the nature of the underlying distribution, and are therefore well-suited to represent distributions of indeterminate type – as is the case for posterior distributions over the parameters of probabilistic rules.

Multivariate distributions are encoded with a multivariate extension of KDE based on product kernels (?). All estimators in our experiments employ Gaussian kernels with a bandwidth tuned from the sample variance using Silverman's rule of thumb (?).

### Learning algorithm

Algorithm 12 presents the general procedure for estimating model parameters from Wizard-of-Oz data. The algorithm loops on each instance pair in the training data. The posterior parameter distribution is estimated via sampling, based on the likelihood of the wizard action (line 2). The

---

<sup>5</sup>Numerical methods based on fixed-point and Newton-Raphson iterations do however exist (?).

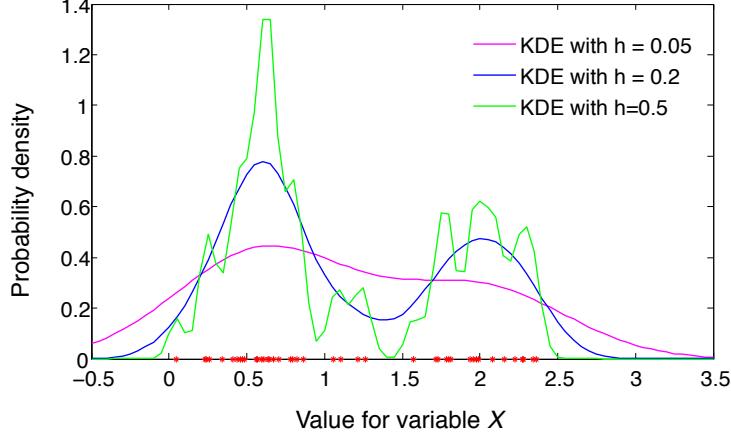


Figure 5.6: Kernel density estimators (KDEs) for a continuous variable  $X$  based on 50 samples (shown on the X axis). The density function is shown for three possible bandwidths  $h$ .

individual posterior distributions for each parameter are then reconstructed via kernel density estimation on the sampled values (line 3-5), and the process is repeated.

---

**Algorithm 12 : WoZ-LEARNING ( $\mathcal{M}, \boldsymbol{\theta}, \mathcal{D}, N$ )**


---

**Input:** Rule-structured models  $\mathcal{M}$  for the domain  
**Input:** Model parameters  $\boldsymbol{\theta}$  with prior distribution  $P(\boldsymbol{\theta})$   
**Input:** Wizard-of-Oz data set  $\mathcal{D} = \{\langle \mathcal{B}_i, a_i \rangle : 1 \leq i \leq n\}$   
**Input:** Number  $N$  of samples to draw for each learning example  
**Output:** Posterior distribution  $P(\boldsymbol{\theta} | \mathcal{D})$  for the parameters

- 1: **for all**  $\langle \mathcal{B}_i, a_i \rangle \in \mathcal{D}$  **do**
  - 2:   Draw  $N$  samples  $\mathbf{x}_1, \dots, \mathbf{x}_N$  from posterior  $P_{\mathcal{B}_i}(\boldsymbol{\theta} | a_i) = \eta P_{\mathcal{B}_i}(a_i; \boldsymbol{\theta}) P(\boldsymbol{\theta})$
  - 3:   **for all** parameter variable  $\theta \in \boldsymbol{\theta}$  **do**
  - 4:     Set  $P(\theta) \leftarrow \text{KDE}(\mathbf{x}_1(\theta), \dots, \mathbf{x}_N(\theta))$
  - 5:   **end for**
  - 6: **end for**
  - 7: **return**  $P(\boldsymbol{\theta})$
- 

### 5.3 Experiments

We evaluated the learning approach outlined in this chapter in the context of a dialogue policy learning task for a human-robot interaction scenario. The goal of the experiment, originally presented in ?), was to evaluate whether the parameters of a rule-structured utility model could be efficiently optimised from small amounts of Wizard-of-Oz data. The evaluation metric was defined in this experiment as the proportion of actions corresponding to the wizard selections. The rule-structured model was compared to two baselines where the action utilities were represented using classical representations (respectively utility tables and linear functions).

It should be stressed that the purpose of the experiment is limited to the evaluation of the *learning* performance of the model. The evaluation of the model in terms of e.g. qualitative

and quantitative metrics of interaction success (and user satisfaction) constitutes an important but separate question, which will be addressed in Chapter 8.

We first describe the dialogue domain used for the experiment, after which we detail the data collection procedure and experimental setup, and finally present and analyse the empirical results.

### 5.3.1 Dialogue domain

The scenario for the Wizard-of-Oz experiment involved a human user and a Nao robot (nicknamed “Lenny”), which is a programmable humanoid robot developed by Aldebaran Robotics. Figure 5.7 shows a human user interacting with the robot during a data collection experiment.

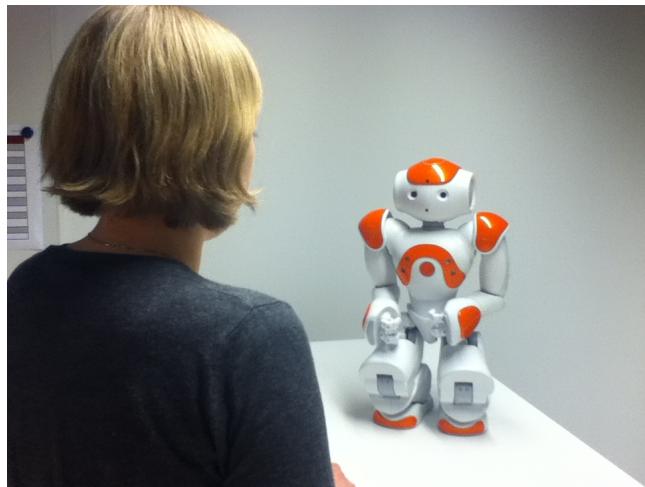


Figure 5.7: Human user interacting with the Nao robot during the Wizard-of-Oz data collection.

The users were instructed to teach the robot a sequence of body movements such as lifting arms, stepping forward, backward, kneeling down, etc. The movements could be performed either in consecutive order or in parallel (provided the movements were not in conflict). The users were free to decide on the movements to perform, and communicated their intention using spoken commands (no gesture recognition was here involved). The robot was programmed to memorise the instructed sequence and could “replay” it at any time.

The list of possible dialogue acts for the user is shown in Table 5.1, and includes a total of 16 dialogue act templates (expanding into 41 dialogue acts when counting all possible argument instantiations). The set of user dialogue acts contain both task-specific dialogue moves to convey user commands as well as conversational actions for feedbacks, acknowledgements, corrections and engagement. To respond to these user inputs, the robot/wizard had at its disposal a repository of 12 possible actions (expanding into 41 alternative actions when counting all possible instantiations of the action arguments). The actions included both physical and verbal actions. The verbal actions available to the system comprised various types of clarification requests and grounding acts. The list of system actions is given in Table 5.2.

- MoveArm( $x, y$ )
  - where  $x = \{\text{Left, Right, Both}\}$
  - and  $y = \{\text{Up, Down, Lateral, Forward, Folded}\}$
- MoveHead( $y$ )
  - where  $y = \{\text{Up, Left, Down, Right}\}$
- MoveFoot( $x, y$ )
  - where  $x = \{\text{Left, Right}\}$
  - and  $y = \{\text{Forward, Backward}\}$
- Turn( $y$ )
  - where  $y = \{\text{Left, Right}\}$
- Kneel
- StandUp
- SitDown
- DoMovements( $y$ )
  - where  $y = \{\text{InParallel, InSequence}\}$
- RepeatAll
- ForgetAll
- Confirm
- Disconfirm
- Say( $x$ )
  - where  $x = \{\text{Hello, Compliment, ThankYou, Goodbye}\}$
- GoToInitPose
- FollowMe
- Stop

Table 5.1: List of user actions  $a_u$

- Demonstrate( $z$ )
  - where  $z = \{\text{MoveArm}(x, y), \text{MoveHead}(y), \text{Kneel}, \text{StandUp}, \text{SitDown}, \text{MoveFoot}(x, y), \text{Turn}(y)\}$
  - and  $x, y$  take the same values as for the user actions
- Say( $x$ )
  - where  $x = \{\text{Hello, ThankYou, Goodbye}\}$
- AskConfirmation
- RegisterMove
- UndoMove
- AskRepeat
- Acknowledgement
- AskIntention
- DemonstrateAll
- ForgetAll
- StopMove
- FollowUser

Table 5.2: List of system actions  $a_m$

### 5.3.2 Wizard-of-Oz data collection

#### System platform

An integrated dialogue system was developed in order to collect Wizard-of-Oz interactions for the human-robot interaction domain described above. The dialogue system is equipped with all standard processing modules for speech understanding, generation and robot control. Figure 5.8 illustrates the general system architecture and its connection to the robotic platform. As evidenced by the figure, the general system architecture is directly inspired by information-state approaches. At the centre of the architecture lies a shared dialogue state to which multiple system components are attached. These components monitor the dialogue state for relevant changes and read/write to it as they process their data flow. The dialogue management module is naturally replaced by the wizard during data collection.

The modules used for the experiments include in particular an off-the-shelf speech recogniser (Vocon 3200 from Nuance) connected to four microphones placed on the robot head. The language model of the speech recogniser is encoded with a small hand-crafted recognition grammar, while the acoustic model was set to U.S. English. The corresponding user dialogue acts are then derived from the ASR hypotheses via a template-based recognition model. On the generation side,

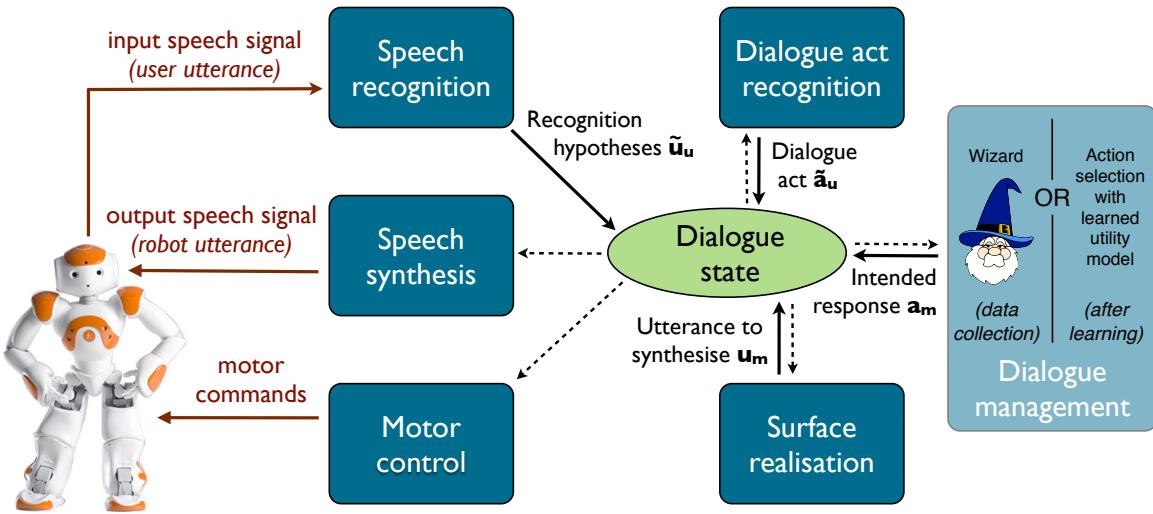


Figure 5.8: System architecture used for the experiment.

a shallow generation model is in charge of the surface realisation of the verbal actions. The surface string is synthesised via a speech synthesis module embarked on the robot. Finally, the execution of physical actions is delegated to a separate component, responsible for planning the robot movements and controlling its motors in real-time, based on the software libraries available on the robotic platform.

### Data collection procedure

Each recorded dialogue involved one human subject interacting with the Nao robot in a shared visual scene. The dialogues were relatively short, with an average duration of about four minutes. We collected a total of 20 interactions with 7 distinct users, for a total of 1020 system turns, summing to around 1h of interaction. The users were recruited amongst the students and researchers in the Department of Informatics at the University of Oslo, while the role of the wizard was taken by the author of the present thesis. All the interactions were performed in English. All but one users were non-native speakers of English.<sup>6</sup> As the speech recogniser relied on a language model of limited coverage, the users were briefed before each experiment about the comprehension capabilities of the robot in order to adjust their expectations about what the robot could and could not understand, and thereby limit the number of out-of-coverage utterances.

Wizard-of-Oz experiments should ideally place the wizard in the same types of decision contexts as the ones encountered by the dialogue manager. To this end, the wizard was prevented from listening directly to the spoken utterances of the user and was instead provided with the N-best list  $\tilde{a}_u$  generated by the dialogue act recognition (which is itself based on the speech recogniser outputs). The N-best lists appeared on the wizard control screen as lists of hypotheses accompanied with their respective probabilities. On the basis of these inputs (and the interaction context), the wizard could then select the action to perform from a list of alternatives.

<sup>6</sup>This mismatch between the acoustic model trained on native American speakers and the foreign accents of the participants may partly explain the relatively high level of speech recognition errors.

Transcripts 1 and 2 present two recorded excerpts of Wizard-of-Oz interactions. The user utterances are displayed as N-best lists of speech recognition hypotheses.

### **Dialogue state**

Each selected action  $a_m$  was recorded along with the complete dialogue state  $\mathcal{B}$  in effect at the time of the selection. The dialogue state variables are represented with their full probability distributions.

The dialogue state designed for the experiment consisted of five independent variables:

1. The last user dialogue act  $a_u$  (with the values given in Table 5.1)
2. The last system action  $a_m$  (with the values given in Table 5.2)
3. The recorded sequence of (confirmed) movements, encoded as a list (cf. Section 4.5)
4. The last physical movement demonstrated by the robot
5. Finally, a periodically updated variable expressing the number of seconds elapsed since the last user or system action. This variable is used to determine when the system should ask for an explicit confirmation from the user.

The dialogue state recorded right before the wizard registers the first movement in Transcript 1 corresponds for instance to the following specification:

$$\mathcal{B} = \begin{cases} a_u = \langle (\text{Confirm}, p=0.88), (\text{SayThankYou}, p=0.10), (\text{None}, p=0.02) \rangle \\ a_m = \langle (\text{AskConfirmation}, p=1) \rangle \\ moveSequence = \langle (\emptyset, p=1) \rangle \\ lastMove = \langle (\text{MoveArm(Left, Up)}, p=1) \rangle \\ silenceTime = \langle (3 \text{ seconds}, p=1) \rangle \end{cases}$$

### **5.3.3 Experimental setup**

The central question investigated in this experiment is the following: does the encoding of dialogue management models in terms of probability and utility rules really improve the efficiency of the parameter estimation process compared to more traditional representations? And if yes, how significant is the difference? The experiment focused therefore on the learning performance of various utility models based on limited amounts of training data gathered from Wizard-of-Oz interactions. The parameters correspond here to the utilities of the various system actions depending on the current state.

### **Baseline models**

The experiment relied on two distinct baselines that express the utility model of the domain based on traditional representations:

1. The first baseline is a plain utility table that maps every combination of state values and actions to a particular utility. For a given set of state variables  $X_1, \dots, X_n$ , the utility for the

---

**Transcript 1** User interaction with wizard-controlled robot, first excerpt

---

HUMAN:  $\begin{bmatrix} \text{hello Lenny} & 0.22 \\ \text{follow me} & 0.20 \\ \text{hi Lenny} & 0.53 \end{bmatrix}$

ROBOT: Hello there!

HUMAN:  $\begin{bmatrix} \text{move your left arm up} & 0.41 \\ \text{move the left arm up} & 0.27 \\ \text{put your left arm up} & 0.30 \end{bmatrix}$

ROBOT: Sure!

[ROBOT executes the movement MoveArm(Left, Up)]  
[ROBOT waits a few seconds]

ROBOT: Was that correct?

HUMAN:  $\begin{bmatrix} \text{yes} & 0.88 \\ \text{thanks} & 0.10 \end{bmatrix}$

[ROBOT registers the movement as confirmed]

ROBOT: Great

HUMAN:  $\begin{bmatrix} \text{move your right arm up at the same time} & 0.26 \\ \text{put the right arm up at the same time} & 0.32 \\ \text{move the right arm up at the same time} & 0.38 \end{bmatrix}$

ROBOT: Let's see

[ROBOT executes movement MoveArm(Both, Up)]

ROBOT: Was that correct?

HUMAN:  $\begin{bmatrix} \text{kneel} & 0.25 \\ \text{thanks} & 0.14 \\ \text{no} & 0.57 \end{bmatrix}$

[ROBOT discards last movement and goes back to previous position]

ROBOT: What should I do then?

[...]

---

---

**Transcript 2** User interaction with wizard-controlled robot, second excerpt

---

[...]

HUMAN:	stop	0.25
	stand up	0.41
	no	0.25

ROBOT: Let's see

[ROBOT executes movement StandUp]

HUMAN:	put your right arm forward	0.31
	move your right arm forward	0.36
	move the right arm forward	0.30

[ROBOT registers movement StandUp as implicitly confirmed]

ROBOT: Let's see

[ROBOT executes movement MoveArm(Right, Forward)]

HUMAN:	move forward the left arm	0.31
	lower the left arm	0.35
	lower both arms	0.29

ROBOT: Sorry, could you repeat?

HUMAN:	raise your left arm	0.28
	lift your right arm	0.31
	lift your left arm	0.37

ROBOT: Sure

[ROBOT executes movement MoveArm(Left, Up)]

[ROBOT waits a few seconds]

ROBOT: Should I do it like this?

HUMAN:	kneel	0.23
	wrong	0.22
	no	0.47

[ROBOT discards last movement and goes back to previous position]

ROBOT: What should I do then?

HUMAN:	move your left arm up	0.29
	move the left arm up	0.34
	move the left arm down	0.28

[...]

---

system action  $a'_m$  is therefore defined by:

$$U(X_1 = x_1, \dots X_n = x_n, a'_m) = \theta_{(x_1, \dots x_n, a'_m)} \quad (5.11)$$

where the  $\theta_{(x_1, \dots x_n, a'_m)}$  value corresponds to the utility encoded in the table for the state-action pair. The total number of required parameters is therefore  $|Val(X_1)| \times \dots |Val(X_n)| \times |Val(a'_m)|$ . In order to keep the model tractable, the utility table was factored in the experiments in three parts, each responsible for a subset of the possible system actions. These utility tables comprised a total of 8962 independent parameters.

2. The second baseline defines the utility of a given action as a linear combination of values – one for each state variable. The total utility is thus determined as:

$$U(X_1 = x_1, \dots X_n = x_n, a'_m) = \sum_{i=1}^n \theta_{(x_i, a'_m)} \quad (5.12)$$

where  $\theta_{(x_i, a'_m)}$  corresponds to the utility weight of the variable value  $x_i$  for the action  $a'_m$ . Note that the weights are specific to a given action value. The number of required parameters is here  $|Val(a'_m)| \times (|Val(X_1)| + \dots + |Val(X_n)|)$ . As a consequence, the size of the linear model is reduced to 581 independent parameters. This baseline hinges however on the assumption that the total utility of a given action can be decomposed as a linear combination of weights for each state variable value.

## Rule-structured model

The two baselines were compared to a utility model structured with 15 utility rules. The interested reader is invited to browse the specification of these rules in Appendix B. The rule structure was designed by hand, while the parameters (in this case, utility values) remained unknown. The rules were associated to a total of 24 parameters.

## Parameter estimation

Figure 5.9 offers a graphical comparison of the utility models produced for the two baselines and the rule-structured approach. The two baselines are essentially “flattened” or unstructured versions of the rule-based model. The input and output variables remain identical in all three models. However, the two baselines directly associate each state-action combination to a single utility value, while the rule-structured approach defines this overall utility in a more indirect manner, through the instantiation of multiple utility rules.

The parameter estimation procedure followed the Bayesian learning approach detailed in Section 5.2.2. For the baseline models, the function `TRIGGERSMODELS(...)` resulted in the creation of one single utility node connected to the system action, as illustrated in Figure 5.9. The prior distributions on the utility values were initialised with uniform priors on the interval  $[-1, 6]$  in all three models.

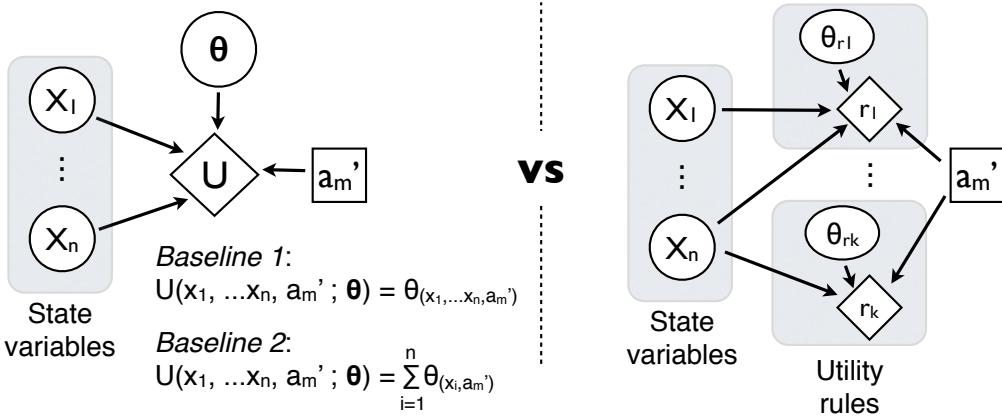


Figure 5.9: Baseline utility models (left) compared to the rule-structured utility model (right).

## Evaluation

Given the utility models defined above, the action to execute corresponds to the action associated with the maximum utility in the current state. In this particular experiment, utility maximisation only considered the current (immediate) utility and did not perform forward planning.

The data collected from the Wizard-of-Oz interactions was split into a training set composed of 765 state-action pairs (75 % of the gathered data) and a held-out test set with 255 actions (remaining 25 %). The same training set was used to estimate the utility parameters for the three models. The resulting utility models were then evaluated on the basis of their agreement with the wizard actions – that is, on the percentage of actions corresponding to the action selected by the wizard in the held-out test set. The agreement results for the three models were evaluated at various stages of the estimation process in order to analyse and compare their learning performance. The results were calculated by sampling over the parameters, performing inference over the resulting models, and finally averaging over the inference results.

### 5.3.4 Empirical results and analysis

Table 5.3 presents the agreement results for the three utility models. The differences between the rule-structured model and the two baselines are statistically significant using Bonferroni-corrected paired  $t$ -tests, with  $p$ -value  $< 0.0001$ .

We performed an error analysis on the 17% of actions that deviate from the wizard behaviour.<sup>7</sup> The analysis revealed that the discrepancy is mainly due to two factors. The first factor is the lack of complete consistency on the part of the wizard, who occasionally decided to follow distinct strategies in similar situations (especially regarding the use of clarification requests). The second factor is the presence of a non negligible number of spurious and noisy data points, notably caused by e.g. interruptions in the middle of the experiments and technical issues with the control of the robotic platform (e.g. movements that had to be repeated due to motor failures).

The learning curves for the three models are shown in Figure 5.10. Note that since the parameters are initially uniformly distributed, the agreement is already non-zero before learning, since a

<sup>7</sup>One should be wary of labelling these actions as “incorrect”, since they are in most cases relevant dialogue moves, but simply result from slightly different decision strategies than the one followed by the wizard.

Type of model	Agreement (in %)
Plain model	67.35
Linear model	61.85
Rule-structured model	<b>82.82</b>

Table 5.3: Agreement results for the three models on a held-out test set.

random assignment of parameters has a low but non-zero chance of leading to the right action.

Thanks to its considerably reduced number of parameters, the rule-structured model is able to converge to near-optimal values after observing only a small fraction of the training set. The incorporation of domain knowledge via the rule structure has a clearly beneficial effect on the learning performance and on the generalisation capacity of the model. As the figure shows, the two baseline models do also improve their accuracies over time, but at a much slower rate. The linear model is comparatively faster than the plain model, but levels off towards the end. The suboptimal learning performance of the linear model is most likely due to the non-linearity of some dialogue strategies. The plain model continues its convergence and would probably reach an agreement level similar to the rule-structured model if given much larger amounts of training data.

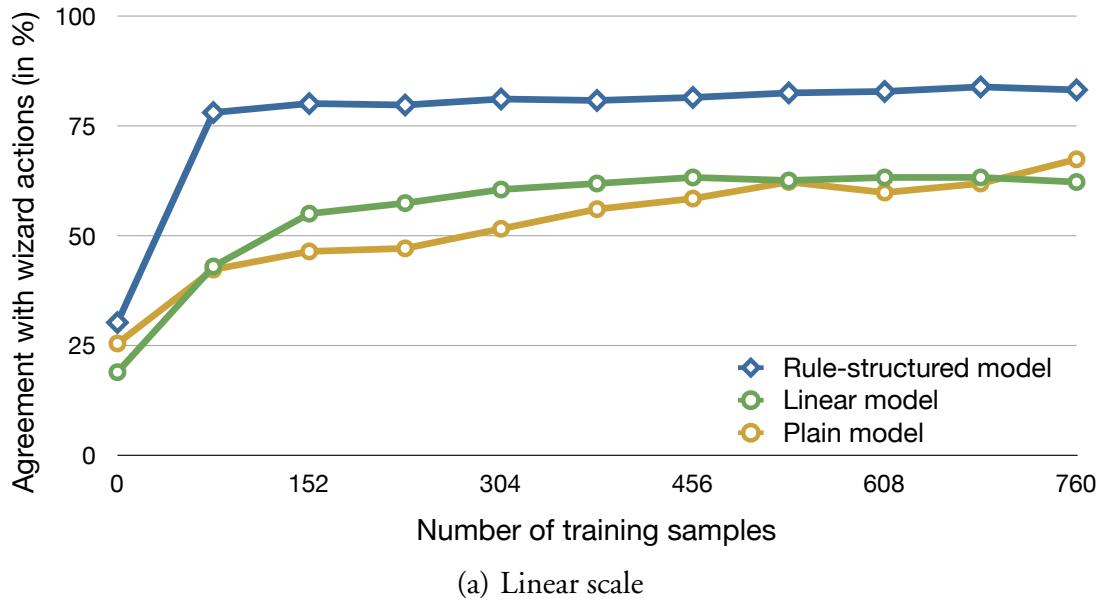
The learning results are in line with our expectations based on the respective sizes of the parameter space for the three utility models, and are not *per se* highly surprising. The main lesson to draw from this experiment is however not the exact difference in agreement or learning rates for each particular model, but the fact that probabilistic rules can be successfully applied to structure a small but non-trivial dialogue domain and derive its parameters from collected interaction data. Through the use of abstraction mechanisms such as partitioning and quantification, the experiment demonstrates that the utility rules specified for the domain (see Appendix B) can cover large regions of the problem space without degrading the model performance. The utility model can hence be optimised with only modest amounts of in-domain data.

## 5.4 Conclusion

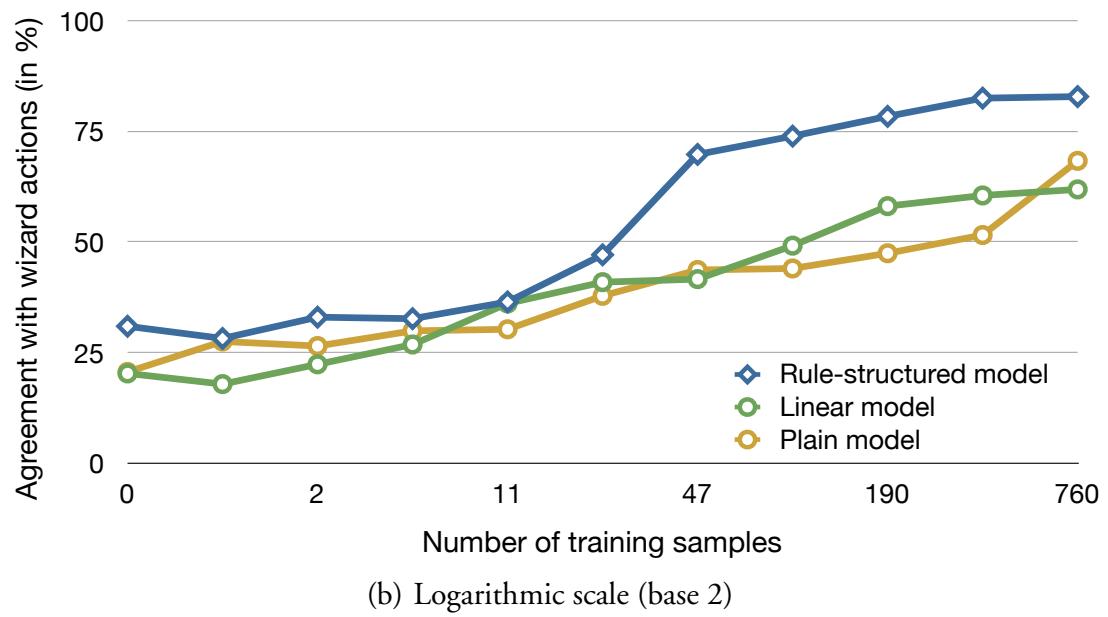
The present chapter described how parameters could be integrated in the specification of probabilistic rules and estimated via Bayesian learning techniques. Rule parameters can represent either probability or utility values. The estimation process operates by calculating posterior probability distributions over possible parameter values given the observed data set. The process is initiated with prior distributions such as Dirichlet priors for probability values and Gaussian or uniform priors for utilities.

The main focus of the chapter was on supervised learning of rule parameters based on Wizard-of-Oz training data. We described how Wizard-of-Oz data can be practically collected and processed to yield data points encoded as pairs  $\langle \text{dialogue state } \mathcal{B}_i, \text{wizard action } a_i \rangle$ . These data points are used to progressively narrow down the spread of the posterior distributions to the values that provide the best fit for the observed wizard actions.

This learning approach has been implemented in a spoken dialogue system for human-robot interaction and validated in a proof-of-concept experiment. The goal of the experiment was to estimate the utility values of various actions on the basis of a Wizard-of-Oz data set. Three utility



(a) Linear scale



(b) Logarithmic scale (base 2)

Figure 5.10: Learning curves for the learned utility models on a held-out test set of 255 actions as a function of the number of processed data points. The agreement results are given for the plain, linear and rule-structured utility models, using both linear (top) and logarithmic scales (bottom).

models were compared: a plain utility table, a linear model, and a model structured with utility rules. The analysis of the empirical results shows that the rule-structured model outperforms the two baselines in regards to learning rate and generalisation performance.

The outcome of the experiment corroborates one of the central claims of this thesis – namely, that hybrid approaches to dialogue modelling (and in particular the formalism of probabilistic rules presented in this thesis) are well suited to model dialogue domains that must simultaneously confront high levels of uncertainty and limited availability of in-domain dialogue data. In such situation, which is commonplace in the field of spoken dialogue systems, neither purely symbolic nor purely data-driven approaches are alone sufficient to harness the complex and stochastic nature of the interactions. Hybrid approaches provide ways to combine expert knowledge and statistically optimised parameters in a single, unified framework, thereby allowing dialogue models to be tuned from even small amounts of training data.

As shown in the experiments, Wizard-of-Oz interaction data can be a useful and interesting source of domain knowledge for the estimation of probabilistic models of dialogue. The data collection procedure can however be a tedious endeavour, as it requires:

1. The availability of an expert (the wizard) that can control the system and provide examples of appropriate behaviour for the domain.
2. The technical setup of a Wizard-of-Oz environment from which the wizard can perceive the user inputs, monitor contextual features, select possible actions to execute, and get all relevant information recorded and stored in a generic format.

A natural alternative to supervised learning from Wizard-of-Oz data is to let the dialogue system learn the best conversational behaviour via trial-and-error from its own interaction experience (that is, through reinforcement learning), without relying on the provision of external examples. The next chapter demonstrates how such strategy can be practically implemented, based once again on the formalism of probabilistic rules to structure the domain models.



# Chapter 6

## Learning from interactions

This chapter extends the parameter estimation approach to a reinforcement learning context. As explained in the first part of this thesis, a reinforcement learning agent learns how to act through a process of trial and error in a given environment. In our case, the environment is represented by verbal interactions with human users, and the system behaviour to learn corresponds to dialogue policies mapping dialogue states to relevant system responses.

The optimisation process is in many respects similar to the one outlined in the previous chapter. Bayesian inference remains the basic instrument for updating distributions over the model parameters on account of the collected evidence. However, the evidence is no longer represented by examples of expert behaviour as in supervised learning. The learning agent instead actively gathers experience through repeated interactions with (real or simulated) users, and receives feedback on its actions in the form of new observations and rewards. The parameter distributions are gradually refined on the basis of this feedback and subsequently used to select the next actions to execute. To minimise the number of parameters, the domain models are structured through probability and utility rules. This structured modelling approach allows the learning agent to escape the “curse of dimensionality” that often characterises dialogue domains. As a consequence, the number of interactions required to reach dialogue policies of high quality can be greatly reduced.

The chapter is divided in three sections. After a short survey of the core concepts of Bayesian reinforcement learning in Section 6.1, we expose in Section 6.2 our own reinforcement learning approach to the estimation of rule parameters. More specifically, we present how the parameters of probabilistic rules can be automatically optimised from interactions using either model-based or model-free strategies. Finally, Section 6.3 describes a practical experiment conducted in a human-robot interaction domain. The purpose of the experiment was to analyse the learning performance of rule-structured models compared to standard categorical distributions. Empirical results on a user simulator show that the rule-structured models converge to optimal parameters – and hence achieve higher returns – in a much shorter time than unstructured representations.

### 6.1 Bayesian reinforcement learning

Bayesian reinforcement learning has recently emerged as a generic framework for learning and acting in uncertain environments (???). As in other types of Bayesian learning methods, the core idea of Bayesian reinforcement learning is to maintain explicit probability distributions over the domain parameters and gradually narrow down the spread of these distributions as more experience

is collected. However, in contrast to supervised approaches, reinforcement learning agents are no longer mere passive observers in the interaction. The agent must indeed actively decide how to act after each turn in order to move the interaction forward.

As explained in Section 3.2, the actions selected by the agent must strike a balance between exploration (trying out new actions) and exploitation (preferring actions that are most likely to yield higher rewards). Bayesian reinforcement learning can offer principled solutions to the exploration-exploitation dilemma, as model uncertainty is explicitly accounted for in the action selection process (??). A Bayesian agent will therefore select actions that are expected to provide the highest long-term return given the current model uncertainty. When the uncertainty is high, information-gathering actions are preferred since they lead to a better understanding of the environment dynamics and are therefore more likely to result in higher future rewards. This inclination to explore gradually fades away as the learning agent develops a better grasp of its domain and becomes more confident about the relative merits of its own actions.

As for other families of reinforcement learning methods, Bayesian reinforcement learning can be divided in model-based and model-free methods.

## Model-based methods

Model-based methods learn an explicit model of the domain in the form of transition, observation and reward models. One benefit of model-based approaches is the relative simplicity of parameter estimation, as the model parameters can be directly updated upon the reception of each observation and reward using standard Bayesian inference. The policy is however complex to optimise, due to the combination of state uncertainty, stochastic action effects, and uncertainty over the parameters. The result is an augmented POMDP where the state also includes random variables expressing the model parameters in addition to traditional state variables. (??).

For domains of small to medium size, approximate dynamic programming methods can be applied to generate the  $\alpha$ -vectors for this augmented POMDP. Point-based solvers (??) have notably shown reasonable performance on a variety of domains. These solution methods are however difficult to scale to more complex models due to the computational intractability of the optimisation process.

An alternative to offline approaches is online planning. Instead of compiling a policy for all possible states (as done in dynamic programming), online planning concentrates on selecting the best action for the current state at runtime. This selection is typically implemented through the construction of a search tree representing possible actions and their effects in terms of rewards and new observations. This tree is gradually expanded until a particular planning horizon is reached. Several approximate methods have recently been developed, based on e.g. point-based value iterations (?) and Monte Carlo tree search (?). The action leading to the highest return on the search tree is then executed by the agent.

One important benefit of online approaches is the possibility to dynamically adapt the domain models at runtime. This characteristic is useful for dialogue domains where the domain models can vary in the course of the interaction – in order to e.g. adapt to shifting user preferences. Offline approaches must in comparison recalculate their policies after every modification or extension of the internal models for the domain. This advantage comes however at a price, namely the fact that planning must be performed at execution time, while the interaction is taking place. Planning must

therefore meet real-time constraints.

Interestingly, offline and online approaches are not mutually exclusive, but can be combined together to take full advantage of both strategies. The idea is to perform offline planning to pre-compute an initial rough policy, and use this policy as a heuristic approximation to guide the search of an online planner (?). These heuristic approximations can for instance be used to provide lower and upper bounds on the values associated with the dialogue states of the search tree. Based on these bounds, the planning algorithm can concentrate its computational efforts in the most fruitful regions of the search space and quickly discard irrelevant actions.

Model-based Bayesian reinforcement learning has been applied to dialogue management in several recent papers. ?) describe a generic Bayes-Adaptive POMDP framework and illustrate its use in simulated interactions. ?) present a similar POMDP framework with model uncertainty combined with active learning. Action selection is formalised in their paper by sampling possible POMDP models and extracting a solution for each sample. A related strategy is employed in a less principled manner by ?). Finally, ??) demonstrate the estimation of observation and reward models for dialogue POMDPs in an healthcare application. One interesting aspect of their work is the use of inverse reinforcement learning to automatically derive a reward model from expert policies.

## Model-free methods

Model-free methods adopt a different learning strategy and directly optimise a dialogue policy from experience, without attempting to construct explicit internal models of the domain. One simple method, formalised by e.g. ?), is to assign prior distributions to the  $Q$  value estimates associated with state-action pairs, and iteratively refine these distributions upon the completion of each action. This update generally relies on Bellman's equation, since the  $Q$  values are never directly observed (only the observations and rewards are available to the agent). The optimal action is then simply defined as the one that maximises the  $Q$  values for the current state, modulo an "exploration bonus" added at learning time to favour exploratory strategies.

Other Bayesian model-free approaches rely on Gaussian processes, which extend the above approach to problems with continuous state and actions spaces (?), and policy gradient methods, which directly optimise a parametrised policy by gradient ascent (??).

? ) present a framework for dialogue policy optimisation based on Gaussian processes. One of the main benefits of their approach is the tremendous acceleration of the optimisation procedure. As a result, the dialogue policy can be optimised via live interactions with human users instead of being confined to simulation. ?) describe a related approach based on Kalman Temporal Differences. As their approach is grounded in Kalman filtering instead of full Bayesian filtering, it only estimates the first and second moment of the parameter distributions – i.e. its mean and variance – instead of the full posterior distribution (?). Finally, Bayesian learning approaches have also been applied to partially observable dialogue domains which necessitate the estimation of a transition model to update the dialogue state, even though the dialogue policy itself is optimised in a model-free manner (?).

## Scalability

Bayesian reinforcement learning has been the subject of much recent research in the last decade, based on both model-based and model-free paradigms. This research focus led to the development

of powerful optimisation methods (see e.g. ?, for a detailed survey). Scalability remains nevertheless a important concern when porting these methods to real applications. The sizes of the parameter and action spaces are in particular major bottlenecks for many learning methods, especially in partially observable domains.

As argued in the next section, probabilistic rules can contribute to addressing by reducing the number of parameters and filtering out irrelevant actions from the planning process.

## 6.2 Optimisation of rule parameters

We developed in this thesis two distinct approaches to the optimisation of rule parameters from unannotated interactions. Both employ Bayesian reinforcement learning as theoretical framework and probabilistic rules as representation formalism, but follow distinct optimisation strategies:

- The first approach follows a model-based strategy. In this approach, the rule-structured models  $\mathcal{M}$  of the domain correspond to transition, observation and reward models. The models are associated to a collection of parameters  $\theta$  with prior distributions  $P(\theta)$ . These parameter distributions are updated on the basis of the observations and rewards received by the system during the interactions. Forward planning is used at runtime to calculate the expected cumulative utilities of possible actions and select the one yielding the maximum utility given the current dialogue state and rule parameters.
- The second approach is a model-free strategy. The transition and reward models are here replaced by a collection of parametrised utility rules representing the estimated  $Q$  value for the system actions. In contrast to the model-based strategy, the utility parameters are here updated via a temporal-difference learning method. The actions to execute are determined through an  $\epsilon$ -greedy policy that strikes a balance between the selection of known high-utility actions and the exploration of new actions.

The two sections below flesh out these two approaches in more detail.

### 6.2.1 Model-based approach

The model-based approach relies on the specification of probabilistic rules that describe:

- the *transition model* for the domain, i.e. how the dialogue state is likely to change as a result of the system actions
- the *observation model* for the domain, i.e. what are the likely observations associated with a given dialogue state;
- the *reward model* for the domain, i.e. what are the immediate utilities (reflecting the system objectives) that result from the execution of particular system actions.

Figure 6.1 depicts a dynamic decision network where transition model is encoded in the figure as a probability distribution  $P(s_t | s_{t-1}, a_{t-1} ; \theta_T)$ , the observation model by a probability distribution  $P(o_t | s_t ; \theta_O)$  and the reward model by the utility distribution  $R_t(s_t, a_t ; \theta_R)$ . For the sake of clarity, the figure abstracts away from the rule nodes mediating between the variables in the network, and upon which the parameters are attached.

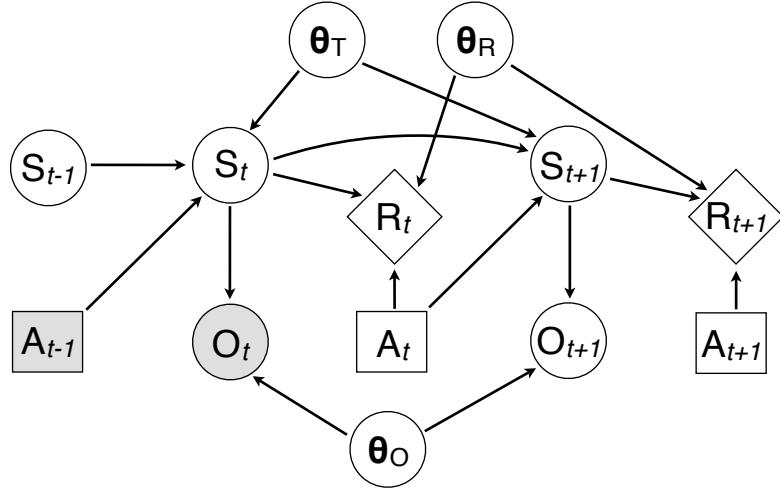


Figure 6.1: Dynamic decision network for a model-based learning strategy.

### Parameter estimation

The probabilistic rules corresponding to these three models may all include unknown parameters that must be estimated from data. Fortunately, this worst case scenario where  $\theta_T$ ,  $\theta_R$  and  $\theta_O$  must all be learned from scratch rarely happens in practice. As exemplified in the experiment described at the end of this chapter, the reward and observation models can often be defined by the system designer prior to learning.

Two types of information sources are available to the agent to refine its parameters during the interaction: the observations and the rewards. In the model-based setting, parameter update is relatively straightforward. The key idea is to include the parameter variables as part of the dialogue state. The probability distributions of these parameters are then automatically updated as part of the state update process (see Algorithm 7 in Section 4.4). There is therefore no need for special purpose mechanisms beyond standard Bayesian update.

### Example of parameter update

Let us illustrate this process on the domain example from Section 4.4.3, if we assume the effect distribution associated with the predictive rule  $r_{11}$  to be unknown and replaced by parameters:

$$\begin{aligned}
 r_{11} : \forall y : \\
 \text{if } (a_m = \text{AskClarify} \wedge a_u = y) \text{ then} \\
 \begin{cases} P(a_{u-p} = y) = \theta_{r_{11}(1,1)} \\ P(\{\cdot\}) = \theta_{r_{11}(1,2)} \end{cases}
 \end{aligned}$$

Figure 6.2 illustrates how the distribution over the parameter values for  $\theta_{r_{11}(1,\cdot)}$  is automatically modified as part of the state update operation. The dialogue example remains unchanged:

USER : Now move forward

$$\tilde{a}_u = \langle (\text{Request(Forward)}, 0.6), (\text{Request(Backward)}), 0.4 \rangle$$

SYSTEM : Could you please repeat?

USER : Please move forward!

$$\tilde{a}_u = \langle (Request(Forward), 0.7), (Other, 0.2), (Request(Backward), 0.1) \rangle$$

To keep the procedure as simple as possible, the figure ignores the steps related to action selection and concentrates on the application of rule  $r_{11}$ . The prior distribution  $P(\theta_{r_{11}(1,\cdot)})$  is set in this example to  $\sim \text{Dirichlet}(2, 1)$ , as we can reasonably presuppose that the user is more likely than not to repeat her/his last utterance after an explicit request from the system.

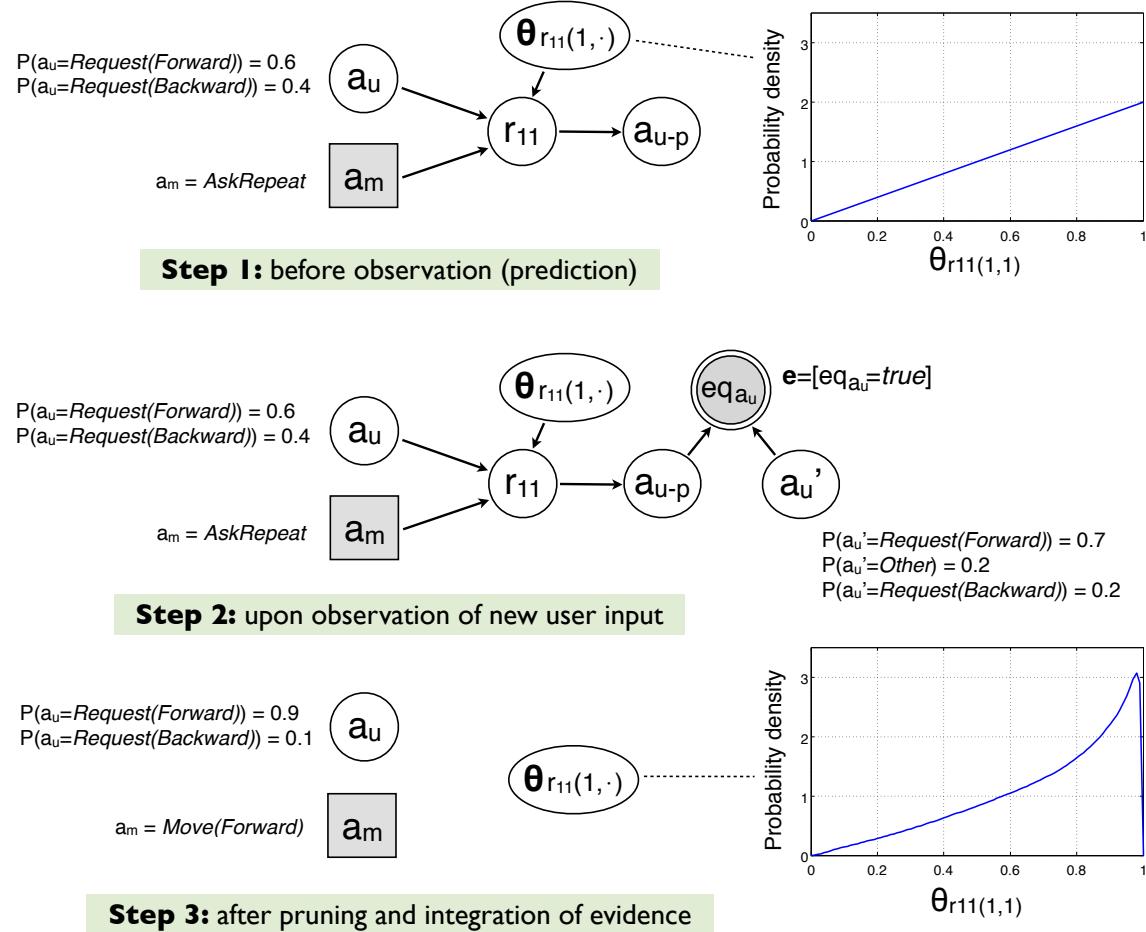


Figure 6.2: Parameter update for  $\theta_{r_{11}(1,\cdot)}$  after the reception of a new user input.

The first step illustrates the instantiation of the rule along with its parameter. Upon the reception of a new observation in the form of a user input  $a'_u$ , the dialogue state and parameters are updated (step 2). After pruning and integration of the evidence (step 3), we notice that the parameter distribution  $P(\theta_{r_{11}(1,\cdot)})$  has shifted part of its probability mass further to the right. In other words, the probability of the user repeating his last utterance becomes somewhat more likely. The posterior distribution  $P(\theta_{r_{11}(1,\cdot)})$  after the update is constructed using kernel density estimation.

### Action selection

As the  $Q$  values are not directly accessible in model-based approaches, action selection must resort to either dynamic programming or forward planning to calculate the expected future rewards of each

action. Action selection is the computational bottleneck in model-based Bayesian reinforcement learning, since the agent needs to reason not only over all the current and future states, but also over all possible models (parametrised by the  $\theta$  variables). The high dimensionality of the task often prevents the use of offline solution techniques. We apply in this work a simple forward planning algorithm coupled with importance sampling.

Algorithm 13 selects the action to execute through forward planning on a given horizon  $h$ . The selection procedure relies on the recursive function  $\text{CALCULATE-Q-VALUES}(\mathcal{B}, \mathbf{e}, h)$  to compute the Q-values of possible actions given a current state  $\mathcal{B}$ , evidence  $\mathbf{e}$  and planning horizon  $h$ .

---

**Algorithm 13 : PLANACTION ( $\mathcal{B}, \mathbf{e}, h$ )**


---

**Input:** Dialogue state  $\mathcal{B}$  as a decision network

**Input:** Evidence  $\mathbf{e}$

**Input:** Planning horizon  $h$

**Output:** Selected action  $\mathbf{a}^*$

- 1:  $Q_{\mathcal{B}} \leftarrow \text{CALCULATE-Q-VALUES}(\mathcal{B}, \mathbf{e}, h)$
  - 2: Find optimal value  $\mathbf{a}^* = \operatorname{argmax}_{\mathbf{a}} Q_{\mathcal{B}}(\mathbf{a})$
  - 3: Remove utility nodes from the state  $\mathcal{B}$
  - 4: **return**  $\mathbf{a}^*$
- 

---

**Algorithm 14 : CALCULATE-Q-VALUES ( $\mathcal{B}, \mathbf{e}, h$ )**


---

- 1: Let  $\mathbf{A}$  be the set of all decision variables in  $\mathcal{B}$
  - 2: **for all** possible action  $\mathbf{a} \in Val(\mathbf{A})$  **do**
  - 3:    $Q_{\mathcal{B}}(\mathbf{a}) \leftarrow U_{\mathcal{B}}(\mathbf{a}, \mathbf{e})$
  - 4:   **if**  $h > 1$  **then**
  - 5:      $\mathcal{B}' \leftarrow$  dialogue state updated from  $\mathcal{B}$  after action  $\mathbf{a}$
  - 6:     **for all** possible observation  $\mathbf{o}$  **do**
  - 7:        $\mathcal{B} \leftarrow$  dialogue state updated from  $\mathcal{B}'$  after observation  $\mathbf{o}$
  - 8:        $Q_{\mathcal{B}''} \leftarrow \text{CALCULATE-Q-VALUES}(\mathcal{B}'', \mathbf{e}, h - 1)$
  - 9:        $Q_{\mathcal{B}}(\mathbf{a}) \leftarrow Q_{\mathcal{B}}(\mathbf{a}) + \gamma P_{\mathcal{B}'}(\mathbf{o}) \max_{\mathbf{a}'} Q_{\mathcal{B}''}(\mathbf{a}')$
  - 10:      **end for**
  - 11:     **end if**
  - 12:   **end for**
  - 13: **return**  $Q_{\mathcal{B}}$
- 

The  $Q$  value of an action is the discounted addition of its immediate reward (line 3 in Algorithm 14) and the expected future reward following its execution (line 4-11). Line 6 loops on possible observations. For efficiency reasons, only a limited number of high-probability observations are selected. For each observation, the dialogue state is updated (line 7) and the  $Q$  values for the resulting state  $\mathcal{B}''$  are computed (line 8). The maximum  $Q$  value for this future state is then added to the  $Q$  value for the current state, weighted by the discount factor  $\gamma$  and the probability  $P_{\mathcal{B}'}(\mathbf{o})$  of the observation. The procedure stops when the planning horizon has been reached, or the algorithm has run out of time. The planner then simply selects the action with maximum expected cumulative utility.

Algorithm 14 contains two loops: one cycling over the set of possible actions, and one cycling over the set of possible observations following the system action. This process can be represented in

an AND-OR search tree anchored in the current state. The OR branches denote the system actions along with their respective rewards, while the AND branches denote the observations weighted by their likelihood. An example of such AND-OR search tree is provided in Figure 6.1, which is modified from ?).

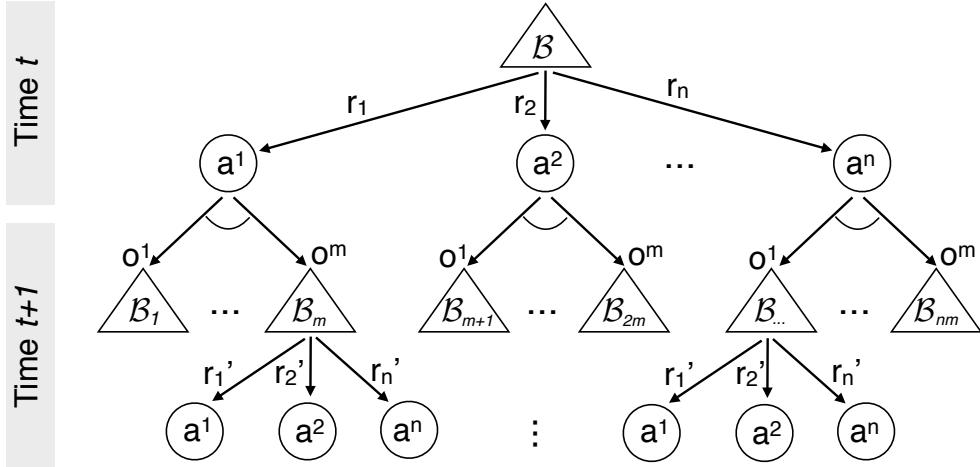


Figure 6.3: AND-OR search tree constructed through forward planning for a horizon of 2.

Our implementation of Algorithms 13 and 14 operates in anytime mode and expands the search tree by gradually adding new observations and actions in a breadth-first manner. At any point in time, the planning algorithm is thus able to deliver a solution. The quality of the solution will of course depend on the accuracy of the search tree, which itself depends on the number of sampled actions and observations – more trajectories leading to a more accurate plan, but at a higher computational cost. The anytime nature of the algorithm is important since the planner operates online and must thus satisfy real-time constraints.

As argued in ?), the reliance on utility rules to encode the reward model can help making the planning process more tractable. In addition to assigning utilities to system actions, utility rules also implicitly define the set of action values that are relevant at a given time.<sup>1</sup> In other words, actions that are not deemed relevant in a particular state are automatically filtered out from the planning process. Instead of searching through the whole space of possible actions, the planning algorithm is thus limited a subset of actions that are locally relevant. One interesting aspect of this approach is that the filtering of relevant actions is done solely on the basis of the provided reward model and does not require the integration of additional constraints or ad-hoc mechanisms. This stands in contrast with most existing work in dialogue management, where constraints on possible actions are typically enforced through external filtering techniques defined on the basis of e.g. information-state update rules (?), finite-state automata (?), or – in our own previous work on this problem – high-level constraints encoded in Markov Logic formulae (?).

### Learning cycle

The model-based learning cycle is detailed in Algorithm 15. The learning agent incrementally updates its model parameters  $\theta$  by running a number of interactions, either with a real user or in

<sup>1</sup>Recall that in Algorithm 5 from Section 4.3, the set of possible values of an action node is directly derived from the values listed in the utility nodes connected to it.

simulation. Starting with an initial dialogue state, the interaction alternates between the reception of new observations (in the form of e.g. user inputs or contextual changes in the environment) and the execution of system actions following these observations. The dialogue state is updated after each observation (line 5), using the procedure outlined in Section 4.4, with the action selection method replaced by Algorithm 13. The selected actions are then executed (line 7). If the reward model is unknown, the reward resulting from the system actions can be integrated in the set of observations and used to update the reward parameters accordingly.

---

**Algorithm 15 : MODEL-BASED-RL-LEARNING ( $\mathcal{M}, \mathcal{B}_0, \boldsymbol{\theta}, N$ )**


---

**Input:** Rule-structured models  $\mathcal{M}$  for the domain  
**Input:** Initial dialogue state  $\mathcal{B}_0$   
**Input:** Model parameters  $\boldsymbol{\theta}$  with prior distribution  $P(\boldsymbol{\theta})$   
**Input:** Number  $N$  of interactions to collect  
**Output:** Posterior distribution  $P(\boldsymbol{\theta})$  for the parameters

- 1: **for**  $i = 0 \rightarrow N$  **do**
- 2:     Start new interaction with initial state  $\mathcal{B} = \mathcal{B}_0 \cup \boldsymbol{\theta}$
- 3:     **while** interaction is active **do**
- 4:         Get new observations  $\mathbf{O}$
- 5:         UPDATESTATE( $\mathcal{B}, \mathbf{O}$ )
- 6:         **if** non-empty selected action  $a$  in  $\mathcal{B}$  **then**
- 7:             Execute action  $a$  and get resulting reward  $r$
- 8:         **end if**
- 9:     **end while**
- 10: **end for**
- 11: **return**  $P(\boldsymbol{\theta})$

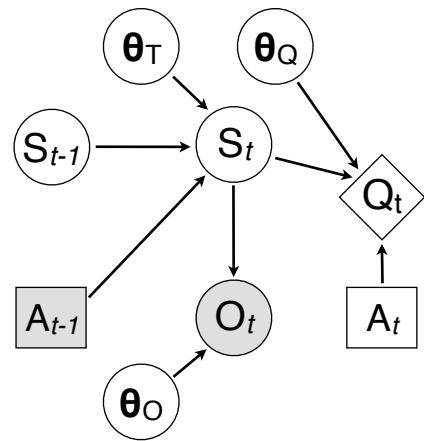
---

### 6.2.2 Model-free approach

In parallel to the model-based learning strategy described above, we also developed an alternative Bayesian model-free approach to the estimation of rule parameters. In this approach, the core model that is to be estimated is the *action-value model* which specifies the expected cumulative reward  $Q$  of the system actions depending on the current state.

In addition to this action-value model, the domain may also include transition and observation models. However, the transition and observation models are in model-free approaches only used for state update (if the dialogue state contains hidden state variables that are indirectly inferred from observations, such as the user intentions) and are not exploited in the action selection. This stands in contrast with model-based approaches to reinforcement learning where transition and observation models are directly employed to plan the next action.

Figure 6.4 illustrates how these distributions combine to form a dynamic decision network. As for the model-based approach, all domain models are specified with probabilistic rules (which are again abstracted away from the simplified diagram in the figure). The transition and observation models are encoded by probability rules and the action-value model by utility rules.



In the worst case, all models may include unknown parameters. The transition model is thus defined as a probability distribution  $P(s_t | s_{t-1}, a_{t-1}; \boldsymbol{\theta}_T)$ , the observation model by a distribution  $P(o_t | s_t; \boldsymbol{\theta}_O)$  and the  $Q$  value model by a distribution  $Q_t(s_t, a_t; \boldsymbol{\theta}_Q)$ .

### Parameter estimation

The transition and observation models can be estimated in the same manner as in the model-based approach – that is, by including the parameters in the dialogue state and refining their distributions as part of the state update process.

The estimation of the action-value model is however slightly more intricate, since the  $Q$  values are not directly accessible to the learning agent. The only feedback perceived by the agent are indeed the immediate rewards resulting from its actions and the subsequent observations, not the expected cumulative rewards  $Q$ . A solution to this estimation problem is to rely on Bellman's equation to incrementally improve the action-value estimates on the basis of the rewards resulting from the agent actions. Such methods are called temporal-difference methods and includes many popular reinforcement learning algorithms such as SARSA and Q-learning (?). Temporal-difference methods are also called “bootstrapping” methods as they approximate new  $Q$  value estimates based on previously learned estimates.

The particular model-free learning method applied in this work is based on the well-known SARSA algorithm.<sup>2</sup> The classical, MDP-based definition of SARSA proceeds as follows. Let  $s_t$  be a dialogue state at time  $t$ , followed by a system action  $a_t$ . The execution of the system action  $a_t$  results in a reward  $r_{t+1}$  and a new dialogue state  $s_{t+1}$  which is itself followed by a second system action  $a_{t+1}$ . The SARSA update of the  $Q$  value estimate for the first action  $a_t$  is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (6.1)$$

where  $\alpha$  represents the learning rate of the algorithm. The estimate  $Q(s_t, a_t)$  is thus modified in direction of the value  $[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})]$ , with a learning step expressed by  $\alpha$ .

The approach developed in this thesis rests on a simple Bayesian extension of SARSA. The posterior distribution over the parameters is here computed on the basis of the evidence provided by the reward  $r_{t+1}$  and next system action  $a_{t+1}$ . Given a sequence of state-action-rewards  $\langle \mathcal{B}_t, a_t, r_{t+1}, \mathcal{B}_{t+1}, a_{t+1} \rangle$ , we define the likelihood distribution  $P(r_{t+1}, a_{t+1}; \boldsymbol{\theta})$  as:

$$P(r_{t+1}, a_{t+1}; \boldsymbol{\theta}) = \phi \left( \frac{Q_{\mathcal{B}_t}(a_t; \boldsymbol{\theta}) - (r_{t+1} + \gamma Q_{\mathcal{B}_{t+1}}(a_{t+1}; \boldsymbol{\theta}))}{\sigma} \right) \quad (6.2)$$

where  $\phi(\cdot)$  is the density function for the standard normal distribution  $\mathcal{N}(0, 1)$ . The standard normal distribution has its peak around the value 0 and decreases exponentially with the distance to this mean. The likelihood distribution will therefore yield a high probability when the initial estimate  $Q_{\mathcal{B}_t}(a_t; \boldsymbol{\theta})$  available at time  $t$  is close to the updated estimate  $[r_{t+1} + \gamma Q_{\mathcal{B}_{t+1}}(a_{t+1}; \boldsymbol{\theta})]$  at time  $t + 1$ , and a low probability otherwise. The variance  $\sigma$  encodes the spread of the bell curve and thereby controls the learning rate.

---

<sup>2</sup>SARSA stands for “State-Action-Reward-State-Action, as a reference to the algorithm’s processing sequence.

Based on the likelihood distribution defined in Equation (6.2), the posterior distribution on the parameters is finally rewritten as:

$$P(\boldsymbol{\theta} | r_{t+1}, a_{t+1}) = \eta P(r_{t+1}, a_{t+1}; \boldsymbol{\theta}) P(\boldsymbol{\theta}) \quad (6.3)$$

### Action selection

The above section described how the parameter distributions were updated on the basis of the received rewards and executed actions, but did not explain how the system actions were selected at runtime. A simple strategy is to select the action yielding the maximum  $Q$  value for the current state. This greedy strategy can however result in poor control policies whenever the agent gets stuck in a suboptimal behaviour. Greedy strategies can be improved by allowing the agent to explore other actions once in a while. The relative frequency of these exploration actions compared to the “greedy” actions is expressed by the probability  $\epsilon$ , which is usually small. This method is called an  $\epsilon$ -greedy strategy and is illustrated in Algorithm 16.

---

#### Algorithm 16 : $\epsilon$ -GREEDY-POLICY ( $\mathcal{B}$ , $e$ )

---

**Input:** Dialogue state  $\mathcal{B}$  as a decision network

**Input:** Evidence  $e$

**Output:** Selected action  $a^*$

- 1: Select value  $a^* = \begin{cases} \text{argmax}_a Q(a, e) & \text{with probability } (1 - \epsilon) \\ \text{another action} & \text{with probability } \epsilon \end{cases}$
  - 2: Remove utility nodes from the state  $\mathcal{B}$
  - 3: **return**  $a^*$
- 

### Learning cycle

The model-free learning cycle is mostly similar to the one defined in the model-based setting. As shown in Algorithm 17, the agent estimates the values of its rule parameters by collecting a number of interactions. Each interaction starts from the initial dialogue state  $\mathcal{B}_0$  and unfolds as a sequence of observations and actions. The dialogue state contains both traditional state variables and parameter variables. After perceiving new observations, the dialogue state is correspondingly updated – including transition and observation parameters if present in the domain (line 5). The update process comprises the selection of new system actions, according to the  $\epsilon$ -greedy selection procedure shown in Algorithm 16. When a new action is selected, the posterior distributions over parameters are correspondingly updated through temporal-difference learning (line 7). The action is then executed and the resulting reward is retrieved (line 8). The process is repeated for each interaction.

## 6.3 Experiments

We performed an empirical evaluation of the two outlined approaches to Bayesian reinforcement learning based on a user simulator for a human-robot interaction domain. The evaluation is divided in two parts:

---

**Algorithm 17** MODEL-FREE-RL-LEARNING ( $\mathcal{M}, \mathcal{B}_0, \boldsymbol{\theta}, N$ )

---

**Input:** Rule-structured models  $\mathcal{M}$  for the domain

**Input:** Initial dialogue state  $\mathcal{B}_0$

**Input:** Model parameters  $\boldsymbol{\theta}$  with prior distribution  $P(\boldsymbol{\theta})$

**Input:** Number  $N$  of interactions to collect

**Output:** Posterior distribution  $P(\boldsymbol{\theta})$  for the parameters

```
1: for  $i = 0 \rightarrow N$  do
2:   Start new interaction with initial state  $\mathcal{B} = \mathcal{B}_0 \cup \boldsymbol{\theta}$ 
3:   while interaction is active do
4:     Get new observations  $\mathbf{O}$ 
5:     UPDATESTATE( $\mathcal{B}, \mathbf{O}$ )
6:     if non-empty selected action  $a$  in  $\mathcal{B}$  then
7:       Update posterior  $P(\boldsymbol{\theta} | r, a)$  based on Equation (6.3)
8:       Execute action  $a$  and get resulting reward  $r$ 
9:     end if
10:    end while
11:   end for
12:   return  $P(\boldsymbol{\theta})$ 
```

---

1. The goal of the first experiment was to determine whether the use of probabilistic rules could be shown to improve the performance of a reinforcement learning agent. More specifically, the experiment compared two alternative formalisations of a transition model for a human-robot interaction scenario: one encoded with traditional categorical distributions, and one encoded with probability rules.
2. The goal of the second experiment focused on the comparison between model-based and model-free approaches in Bayesian reinforcement learning. The evaluation compared a model-based learner with an unknown transition model (such as the one used in the first experiment) to a model-free learner with an unknown action-value model. Both strategies relied on probabilistic rules to capture their respective models and were evaluated on the basis of their average rewards when interacting with the user simulator.

We first describe in this section the dialogue domain and user simulator used in both experiments, and then detail the evaluation setups and empirical results for each experiment.

### 6.3.1 Dialogue domain

As in the previous chapter, the dialogue domain chosen for the experiments is a human-robot interaction scenario with a Nao robot. The task is however more complex than the one developed for the supervised learning case. The interactions collected for the experiments involved the Nao robot conversing with a human user in a shared visual scene including a few graspable visual objects, as illustrated in Figure 6.5. The users were instructed to command the robot to carry the objects from one place to another. The users were free to decide which object(s) to pick up, where to place them on the floor, and what kinds of navigation commands to provide to perform the task. In addition to following the human instructions, the robot could also answer factual questions from

the user regarding its own knowledge of the environment such as “*do you see a blue cylinder?*” or “*what do you see?*”.

Each (physical or information-gathering) sub-task is represented as a distinct user intention. As the human users could only instruct the robot to perform one sub-task at a time, the user intention is represented by a single variable denoting the current sub-task that the user wish to see fulfilled. The user intentions for the domain are listed in Table 6.1.

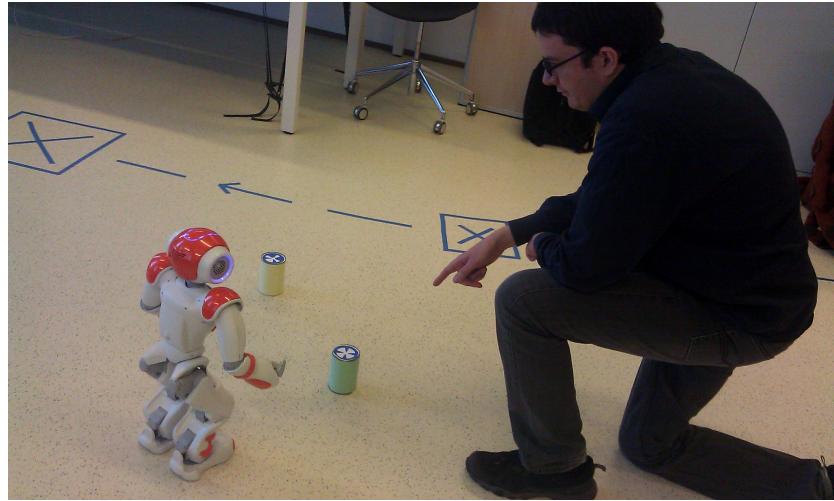


Figure 6.5: Human user interacting with the Nao robot in a shared visual scene with two objects.

The objects in the scene consisted of coloured metallic cylinders with a special marking on their top to facilitate the robot’s visual servoing during grasping tasks. As the Nao robot used in the experiments did not include actuated fingers, the grasping operation employed permanent magnets attached to the robot hands to grasp and carry the cylinders.

In addition to following the user commands related to spatial navigation and object manipulation, the robot could also perform grounding-related actions such clarification requests and acknowledgements. In total, the domain included 9 user dialogue templates. The robot has a repertoire of 8 possible action templates that can be executed. For a dialogue domain with two objects, the user actions  $a_u$  and system actions  $a_m$  will thus expand into respectively 15 and 37 actions. Tables 6.2 and 6.3 list the possible actions respectively available for the user and system.

Transcript 3 provides a detailed example of recorded interaction between a human user and the (wizard-controlled) robot.

The reward model was defined by hand, using conventional values: the execution of correct actions or the correct answer to user questions leads to large positive values (+6 in this particular case) while the execution of wrong or irrelevant actions leads to large negative values (-6) and the use of clarification or confirmation requests to small negative values (from -0.5 to -1.5, depending on the type of request). Table 6.4 presents the reward model defined for the domain.

· $\text{Move}(x)$ where $x = \{\text{Left}, \text{Right}, \text{Forward},$ $\text{Backward}\}$	· $\text{Release}(x)$ where $x$ is an object identifier
· $\text{PickUp}(x)$ where $x$ is an object identifier	· $\text{WhatDoYouSee}$ · $\text{DoYouSee}(x)$ where $x$ is an object identifier

Table 6.1: List of user intentions  $i_u$

· $\text{Ask}(\text{Move}(x))$ where $x = \{\text{Left}, \text{Right}, \text{Forward},$ $\text{Backward}\}$	· $\text{RepeatLastMove}$
· $\text{Ask}(\text{PickUp}(x))$ where $x$ is an object identifier	· $\text{Ask}(\text{WhatDoYouSee})$ · $\text{Ask}(\text{DoYouSee}(x))$ where $x$ is an object identifier
· $\text{Ask}(\text{Release}(x))$ where $x$ is an object identifier	· $\text{Confirm}$ · $\text{Disconfirm}$ · $\text{Other}$

Table 6.2: List of user actions  $a_u$

· $\text{Do}(x)$ where $x = \{\text{Move}(y), \text{PickUp}(z),$ $\text{Release}(z)\}$ and $y = \{\text{Left}, \text{Right}, \text{Forward},$ $\text{Backward}\}$ and $z =$ an object identifier	· $\text{Ground}(x)$ where $x = \{\text{Move}(y), \text{PickUp}(z),$ $\text{Release}(z)\}$ and $y = \{\text{Left}, \text{Right}, \text{Forward},$ $\text{Backward}\}$ and $z =$ an object identifier
· $\text{Excuse}(x)$ where $x = \{\text{DoNotSeeObject}$ $\text{DoNotCarryObject},$ $\text{AlreadyCarryObject}\}$	· $\text{AskClarify}$ · $\text{AskConfirm}(x)$ where $x = \{\text{Move}(y), \text{PickUp}(z),$ $\text{Release}(z), \text{DoYouSee}(z),$ $\text{WhatDoYouSee}\}$
· $\text{Describe}(x)$ where $x =$ a (possibly empty) list of object identifiers	and $y = \{\text{Left}, \text{Right}, \text{Forward},$ $\text{Backward}\}$ and $z =$ an object identifier
· $\text{ConfirmDetection}$	
· $\text{DisconfirmDetection}$	

Table 6.3: List of system actions  $a_m$

---

**Transcript 3** User interaction with wizard-controlled robot

---

ROBOT: Hello human

HUMAN: Hi, hello Lenny!

[ROBOT stands up]

HUMAN: Would you please move forward

ROBOT: OK, moving forward

[ROBOT steps forward]

HUMAN: Would you please move a bit more forward

ROBOT: Should I move backward

HUMAN: No, a bit more forward

ROBOT: OK, forward then

[ROBOT steps forward]

HUMAN: Do you see any object?

ROBOT: Sorry I do not understand

HUMAN: Do you see any object?

ROBOT: I see a blue cylinder

HUMAN: Can you pick it up please?

ROBOT: Should I pick up the object?

HUMAN: Yes please

ROBOT: OK, thanks

[ROBOT moves in direction of the blue cylinder]

[ROBOT picks up the object]

ROBOT: Done

HUMAN: Now turn right

[ROBOT turns right while holding the object]

HUMAN: Move forward

[ROBOT moves forward while holding the object]

HUMAN: And then move two steps backward

ROBOT: OK, moving forward

[ROBOT starts movement]

HUMAN: No please move backward

[ROBOT stops]

[...]

---

Action	Reward
Execution of correct physical action $a_m = \text{Do}(i_u)$	+6
Execution of wrong physical action $a_m = \text{Do}(x)$ with $x \neq i_u$	-6
Declare $a_m = \text{Excuse}(\text{DoNotSeeObject})$ when $i_u = \text{PickUp}(x)$ and $x$ is not perceived	+6
Declare $a_m = \text{Excuse}(\text{DoNotCarryObject})$ when $i_u = \text{Release}(x)$ and $x$ is not carried	+6
Declare $a_m = \text{Excuse}(\text{AlreadyCarryObject})$ when $i_u = \text{PickUp}(x)$ and $x$ is carried	+6
Declare $a_m = \text{Excuse}(*)$ in other circumstances	-6
Correct answer $a_m = \text{Describe}(x)$ when $i_u = \text{WhatDoYouSee}$ and $x$ are the perceived objects	+6
Wrong answer $a_m = \text{Describe}(*)$ when $i_u \neq \text{WhatDoYouSee}$	-6
Correct answer $a_m = \text{ConfirmDetection}$ when $i_u = \text{DoYouSee}(x)$ and $x$ is perceived	+6
Wrong answer $a_m = \text{ConfirmDetection}$ when $i_u \neq \text{DoYouSee}(x)$ or $x$ is not perceived	-6
Correct answer $a_m = \text{DisconfirmDetection}$ when $i_u = \text{DoYouSee}(x)$ and $x$ is not perceived	+6
Wrong answer $a_m = \text{DisconfirmDetection}$ when $i_u \neq \text{DoYouSee}(x)$ or $x$ is perceived	-6
Grounding of correct intention $a_m = \text{Ground}(i_u)$	+2
Grounding of wrong intention $a_m = \text{Ground}(x)$ with $x \neq i_u$	-6
Request to confirm correct intention $a_m = \text{AskConfirm}(i_u)$	-0.5
Request to confirm wrong intention $a_m = \text{AskConfirm}(x)$ with $x \neq i_u$	-1.5
Request to clarify $a_m = \text{AskClarify}$	-1
Ignore user act $a_m = \text{None}$ when $a_u \neq \text{None}$	-1.5

Table 6.4: Reward model for the domain.

### 6.3.2 Simulator

#### Generalities

In order to draw meaningful and reliable comparisons between reinforcement learning approaches, the user behaviours must be made fully consistent across interactions. This consistency must be enforced on both the conversational choices of the user and the average amount of noise and comprehension errors that characterise them. Needless to say, this criteria is hard to satisfy when working with human participants. The comparative evaluation of learning approaches was thus conducted with the help of a simulator.

The simulator is responsible for two tasks:

1. The first task is to emulate the typical dialogue behaviour of an human user, and generate relevant user responses to the system actions.
2. The second task is to maintain a virtual representation of the environment during the interaction (represented in our case by the physical objects) and update this representation as a function of the system actions

It should however be emphasised that the reliance on user simulators to conduct the comparative evaluation of our learning approaches does not in any way imply that simulators are a necessary component of the learning approach presented in this thesis.<sup>3</sup>

---

<sup>3</sup>In fact, probabilistic rules are particularly well suited for optimisation from live interactions, as they require drastically less training data than traditional learning approaches (as evidence by the results in this chapter).

## Wizard-of-Oz study

In order to build a user simulator that matches as closely as possible the behaviour of actual human subjects, we started by recording a set of Wizard-of-Oz interactions in the human-robot dialogue domain chosen for the experiments.

The technical setup employed for the Wizard-of-Oz data collection was mostly similar to the one described in the previous chapter. The purpose of the data collection is however not identical: while the Wizard-of-Oz interactions described in Section 5.3.2 served to determine the most appropriate *system* actions depending on the situation, the goal of the Wizard-of-Oz interactions is here to collect empirical data about the most likely *user* actions in their context. As the wizard behaviour was not the focus of the study, the wizard was allowed to directly listen to the user utterances, without using the speech recogniser as intermediary. The wizard controlled the verbal and physical actions via a remote screen coupled to the robotic platform. Various types of errors and misunderstandings were artificially introduced by the wizard in the course of the interaction in order to also gather data about the user responses to such comprehension errors.

A total of eight interactions were recorded, each with a different speaker (5 males and 3 females), totalling about 50 minutes divided in 486 turns. The interactions were performed in English. The users were again recruited amongst the local group of students and employees in the Department of Informatics at the University of Oslo and were (with one exception) non-native English speakers. The author of the present thesis served as the wizard.

After the recording, the dialogues were segmented and annotated by hand. The first layer of annotation encodes the user dialogue acts  $a_u$  and system actions  $a_m$  listed in Table 6.2 and 6.3. The user intentions  $i_u$  underlying the user commands are annotated on top of this sequence of turns.<sup>4</sup> The annotation also includes two contextual variables respectively expressing the lists of objects perceived and carried by the robot at a given time. Transcript 4 provides a concrete example of annotation for the first part of the interaction in Transcript 3.

## User and context modelling

After collecting and annotating the Wizard-of-Oz interactions, the next step in the development of the simulator is to design the (stochastic) transition model that determines how the user and the environment are to respond to the system actions. This statistical model is used at runtime to sample possible responses and feed them back to the dialogue system.

The transition model for our human-robot interaction dialogue domain is factored in four components:

1. a user goal model  $P(i'_u | i_u, a_m, \text{perceived}', \text{carried}')$  describing the probability of the next user intention  $i'_u$  as a function of the current intention  $i_u$ , the last system action  $a_m$  and the two contextual variables *perceived* and *carried*.<sup>5</sup>
2. a user action model  $P(a'_u | i'_u, a_m)$  describing the probability of the new user action  $a'_u$  as a function of the new user intention and last system action.

---

<sup>4</sup>Although the user intentions are in principle hidden “mentalistic” entities, they can in our domain be easily determined by a human annotator from the dialogue transcript.

<sup>5</sup>The user intention can depend from the contextual variables since the user is e.g. more likely to ask the robot to grasp an object if it sees it, and less likely to ask the same request if the robot already carries an object.

---

**Transcript 4** Annotated dialogue excerpt

---

HUMAN: Would you please move forward

**Annotation:**  $a_u = \text{Ask}(\text{Move(Forward)})$   
 $i_u = \text{Move(Forward)}, \text{carried} = [], \text{perceived} = []$

ROBOT: OK, moving forward

**Annotation:**  $a_m = \text{Ground}(\text{Move(Forward)}) + \text{Do}(\text{Move(Forward)})$

HUMAN: Would you please move a bit more forward

**Annotation:**  $a_u = \text{Ask}(\text{Move(Forward)})$   
 $i_u = \text{Move(Forward)}, \text{carried} = [], \text{perceived} = [\text{object}_1]$

ROBOT: Should I move backward

**Annotation:**  $a_m = \text{AskConfirm}(\text{Move(Backward)})$

HUMAN: No, a bit more forward

**Annotation:**  $a_u = \text{Disconfirm} + \text{Ask}(\text{Move(Forward)}),$   
 $i_u = \text{Move(Forward)}, \text{carried} = [], \text{perceived} = [\text{object}_1]$

ROBOT: OK, forward then

**Annotation:**  $a_m = \text{Ground}(\text{Move(Forward)}) + \text{Do}(\text{Move(Forward)})$

HUMAN: Do you see any object?

**Annotation:**  $a_u = \text{Ask}(\text{WhatDoYouSee}),$   
 $i_u = \text{WhatDoYouSee}, \text{carried} = [], \text{perceived} = [\text{object}_1]$

ROBOT: Sorry I do not understand

**Annotation:**  $a_m = \text{AskClarify}$

HUMAN: Do you see any object?

**Annotation:**  $a_u = \text{Ask}(\text{WhatDoYouSee}),$   
 $i_u = \text{WhatDoYouSee}, \text{carried} = [], \text{perceived} = [\text{object}_1]$

ROBOT: I see a blue cylinder

**Annotation:**  $a_m = \text{Describe}([\text{object}_1])$

---

3. a contextual model  $P(\text{perceived}' | \text{perceived}, a_m)$  describing how the set of objects perceived by the robot evolves as a function of the system action (since a movement may result in the detection of new objects or make other objects fall from view).
4. a contextual model  $P(\text{carried}' | \text{carried}, a_m)$  describing how the set of objects carried by the robot can change due to grasping and releasing actions.

The resulting probabilistic model that combines these four distributions is depicted in Figure 6.6. The reader may notice that the current state variables at time  $t$  are greyed out, signifying that their values are observed. It should be stressed that the knowledge of these values is limited to the simulator (since the user is aware of its own intentions and dialogue acts, and the environment “knows” its own state). The robot has however no access to the internal state of the simulator.

The transition model was practically designed with a set of probability rules expressing the four distributions based on a small number of structural assumptions about the user behaviour. The probabilities associated with the rule effects were then estimated by maximum likelihood on the basis of the annotated dialogues.

### Error modelling

In real interactions, user inputs are not directly observed by the dialogue manager but must first be processed by the speech recognition and understanding modules. These modules are prone to various failures and frequently distort or misinterpret the actual user utterance. The user simulator should account for this fact by explicitly modelling errors and uncertainties arising from speech recognition and natural language understanding.

In order to reproduce the imperfect nature of the communication channel, the simulator wraps every user input in a N-best list of the following form:

$$\tilde{a}_u = \begin{cases} P(\text{correct } a_u) = p_1 \\ P(\text{another randomly selected value for } a_u) = p_2 \\ P(\text{spurious recognition}) = p_3 \end{cases}$$

where  $\langle p_1, p_2, p_3 \rangle$  are probability values sampled at runtime from a three-dimensional Dirichlet distribution. The distribution  $P(p_1, p_2, p_3)$  is estimated in an empirical manner based on actual speech recognition results for the domain. We first applied the off-the-shelf speech recogniser embarked on the robot (Nuance Vocon) to the audio segments corresponding to the user utterances collected in the Wizard-of-Oz study. The recognition results were then processed in order to extract from each audio segment the three probabilities  $\langle p_1, p_2, p_3 \rangle$ , where  $p_1$  stands for the probability of the correct utterance (which can be zero if the utterance does not appear in the N-Best list),  $p_2$  to the total probability of incorrect utterances, and  $p_3$  to the probability of no recognition. We finally

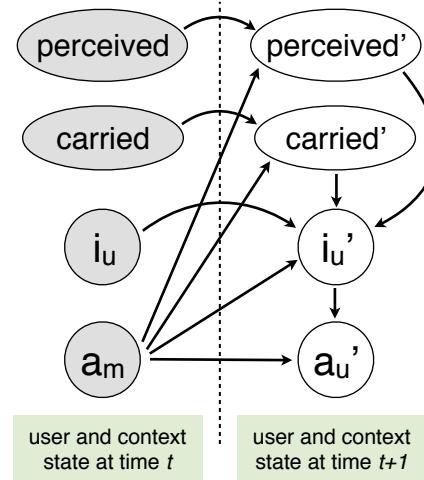


Figure 6.6: User and context models employed by the simulator.

derived a Dirichlet distribution based on these sample probabilities using the estimation method developed by T. Minka (?). The particular Dirichlet distribution resulting from the recognition results was  $\sim \text{Dirichlet}(5.4, 0.52, 1.6)$

At runtime, the probabilities for the N-best list elements are drawn from the Dirichlet distribution. Probability values falling below a minimum threshold are automatically pruned from the N-Best list. The method has been found to match reasonably well the actual recognition results produced by the speech recogniser, although one could naturally refine the approach by e.g. explicitly modelling confusion probabilities between individual inputs.

### Simulation procedure

The simulation procedure takes the form of an interaction loop between the simulator and the dialogue system, as shown in Figure 6.7. Two separate dialogue systems are active: the simulator on the one hand and the control system for the robot on the other hand. The two systems operate however differently, as the simulator has a fully observable state, while the system state is only partially observable. Another obvious difference is the fact that the domain models of the system contain unknown parameters to optimise, while the simulator models are known and fixed.

Contrary to the experiments presented in the previous chapter, the dialogue architecture used in this learning experiment is essentially reduced to the dialogue management module, since the simulator and the dialogue system directly exchange their dialogue actions without needing to express them in actual spoken utterances.

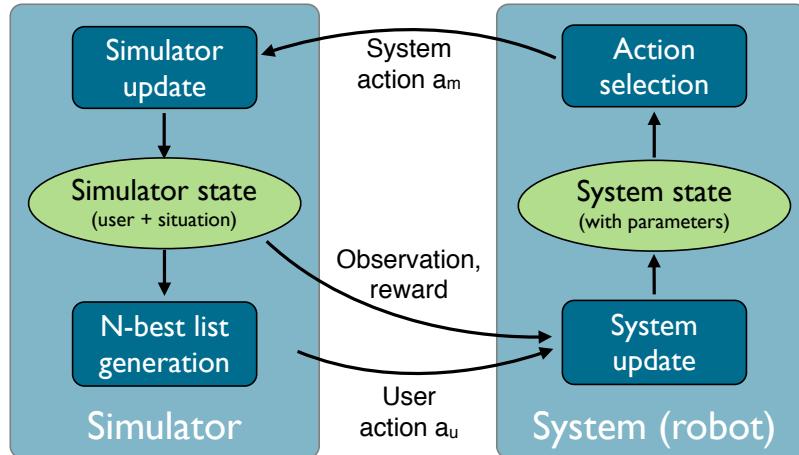


Figure 6.7: Processing workflow for the simulated interaction.

An example of interaction generated when coupling the simulator to the dialogue system is provided in Transcript 5. The excerpt is taken at the onset of the learning cycle, when the system's knowledge of the domain models is still rudimentary, which explains the prevalence of information-gathering actions.

---

**Transcript 5** Example of simulated interaction

---

[Initial context: perceived = [object<sub>1</sub>, object<sub>2</sub>], carried = []]

HUMAN:  $\begin{bmatrix} \text{Ask(PickUp(object<sub>2</sub>))} & 0.06 \\ \text{Ask(PickUp(object<sub>1</sub>))} & 0.42 \end{bmatrix}$

ROBOT: AskRepeat → Reward = -1

HUMAN:  $\begin{bmatrix} \text{Ask(DoYouSee(object<sub>2</sub>))} & 0.01 \\ \text{Ask(PickUp(object<sub>1</sub>))} & 0.77 \end{bmatrix}$

ROBOT: Ground(PickUp(object<sub>1</sub>)) → Reward = 2

ROBOT: Do(PickUp(object<sub>1</sub>)) → Reward = 6, carried = [object<sub>1</sub>]

HUMAN:  $\begin{bmatrix} \text{Ask(DoYouSee(object<sub>2</sub>))} & 0.05 \end{bmatrix}$

ROBOT: AskRepeat → Reward = -1

HUMAN:  $\begin{bmatrix} \text{Ask(DoYouSee(object<sub>2</sub>))} & 0.87 \end{bmatrix}$

ROBOT: AskConfirm(DoYouSee(object<sub>2</sub>)) → Reward = -0.5

HUMAN:  $\begin{bmatrix} \text{RepeatLast} & 0.05 \\ \text{Confirm} & 0.63 \end{bmatrix}$

ROBOT: ConfirmDetection → Reward = 6

HUMAN:  $\begin{bmatrix} \text{Ask(Move(Forward))} & 0.73 \\ \text{Ask(PickUp(object<sub>2</sub>))} & 0.03 \end{bmatrix}$

ROBOT: AskRepeat → Reward = -1

HUMAN:  $\begin{bmatrix} \text{Ask(Move(Forward))} & 0.92 \\ \text{Ask(Move(Left))} & 0.04 \end{bmatrix}$

ROBOT: Ground(Move(Forward)) → Reward = 2

ROBOT: Do(Move(Forward)) → Reward = 6

[...]

---

### 6.3.3 First experiment

The goal of the first experiment is to determine whether the use of probabilistic rules has a beneficial influence on the learning performance of the agent. The motivation is sensibly the same as the one put forward in the experiment of the previous chapter, except the estimation procedure is here based on model-based Bayesian reinforcement learning techniques instead of supervised learning.

The experiment focuses more precisely on the estimation of the transition model for the human-robot interaction domain described in Section 6.3.1. Based on the simulator presented in the previous pages, the experiment compares two alternative representations of the transition model: one baseline model encoded via standard categorical distributions, and one equivalent model encoded via probability rules. The relative performance of these two representations is measured by the average return – i.e. the sum of rewards – per interaction.

The reward model was held fixed and identical in both cases (cf. Table 6.4). Online planning was used for action selection and operated with a horizon of length 2.

#### Baseline model

The transition model  $P(s' | s, a_m)$  is represented in the baseline approach through traditional factored categorical distributions. The transition model is more precisely divided into a user goal model  $P(i'_u | i_u, a_m)$  and a user action model  $P(a'_u | i'_u, a_m)$ .<sup>6</sup> The user goal model is defined in the following manner:

$$P(i'_u | i_u, a_m) = \begin{cases} P(i'_u) & \text{if } a_m \text{ fulfills the intention } i_u \\ 1 & \text{if above condition does not hold and } i'_u = i_u \\ 0 & \text{otherwise} \end{cases}$$

where  $P(i'_u)$  is a categorical distribution that expresses the prior probability of a new user intention  $i'_u$ . The user action model  $P(a'_u | i'_u, a_m)$  is for its part constructed as a plain probability table where each possible assignment of values for the parent variables  $i'_u$  and  $a_m$  is assigned a distinct categorical distribution on the values of  $a'_u$ .

The resulting parameters for these categorical distributions are encoded with Dirichlet priors. The baseline model used for this experiment contains a total of 229 Dirichlet parameters, composed of one Dirichlet with 12 dimensions and 228 with 16 dimensions. Weakly informative priors are used to initialise the prior distributions.

A linear model has also been constructed for this experiment but is not shown in the results as its parameter estimation systematically diverged and fared much worse than the classical factored distributions, most likely due to the non-linearity of the underlying domain models.

#### Rule-structured model

The rule-structured model is encoded with parametrised probability rules. A total of six rules with 13 corresponding Dirichlet parameters (of varying dimensions) is used to define the transition model. As for the baseline model, the rule parameters are initially associated with weakly

---

<sup>6</sup>The two contextual variables perceived and carried are not included in the transition model since their values do not need to be predicted in advance.

informative Dirichlet priors. The rules designed for the experiment are listed in Appendix B.

### Empirical results

The performance was first measured in terms of average return per simulated interaction, shown in Figure 6.8. To analyse the accuracy of the transition model, we also derived the Kullback-Leibler divergence (?) between the next user act distribution  $P(a'_u)$  predicted by the learned model and the actual distribution followed by the simulator at a given time, as shown in Figure 6.9. Some residual discrepancy is to be expected between these two distributions, the latter being based on the actual user intention while the former must infer it from the current belief state. The results of both figures are averaged on 100 simulations.

### Analysis of results

The empirical results illustrate that both models are able to capture at least some of the interaction dynamics and achieve higher returns as the number of turns increases, but they do so at different learning rates. In our view, this difference is to be explained by the higher generalisation capacity of the probabilistic rules compared to the unstructured categorical distributions.

One can observe from the empirical results that the Dirichlet parameters associated with the probabilistic rules converge to their optimal value very rapidly, after a handful of episodes. This is a promising result, since it implies that the proposed approach could in principle optimise dialogue policies from live interactions, without resorting to a user simulator.

One caveat is nevertheless here in order. As described in the previous section, the simulation model used to sample the next intentions and actions of the user is built on a number of structural assumptions. The learning results must therefore be interpreted with caution, as one could object that the good learning performance of the rule-structured model is an artifact of the regular behaviour exhibit by the simulator, and that such regularities may not be found in real-world dialogues. This is a valid objection, although the posited assumptions on the user behaviour are relatively conservative and backed by a preliminary analysis of the actual user responses in the Wizard-of-Oz studies. We shall however see in Chapter 8 that the encouraging results presented here also carry over to genuine human-robot interactions conducted without simulator.

#### 6.3.4 Second experiment

The first experiment was confined to the analysis of model-based methods to reinforcement learning and did not compare the performance of model-based and model-free approaches to the estimation of rule parameters. The second experiment remedies this shortcoming. As in the previous experiment, the reward model is provided but the transition model is unknown. Both approaches are encoded in this experiment with probabilistic rules.

In the model-free case, the rule parameters encode the action-value model  $Q(s, a)$  over the return of state-action pairs, while the model-based case focuses on the transition model  $P(s' | a, s)$ . Figure 6.10 illustrates these two learning strategies.

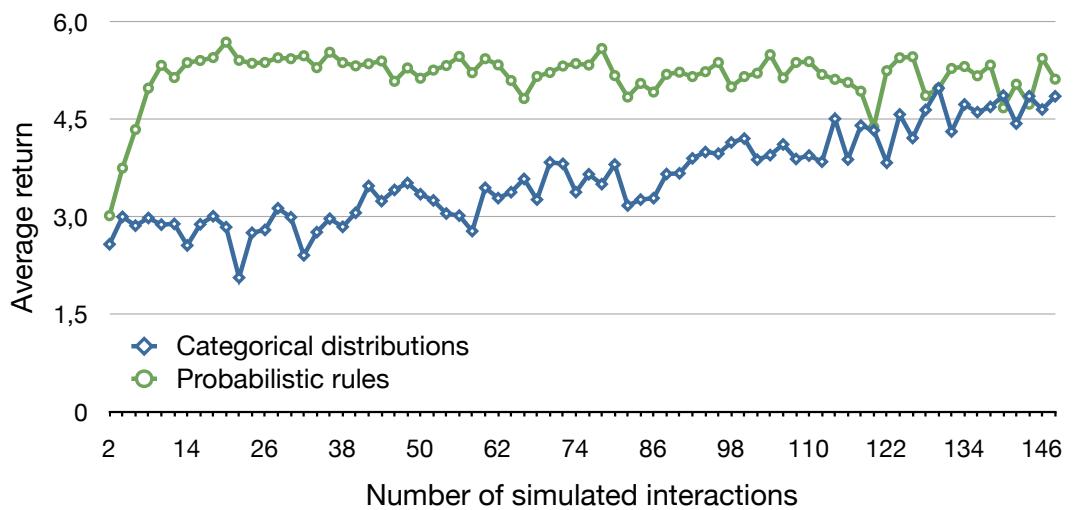


Figure 6.8: Average return as a function of the number of simulated interactions.

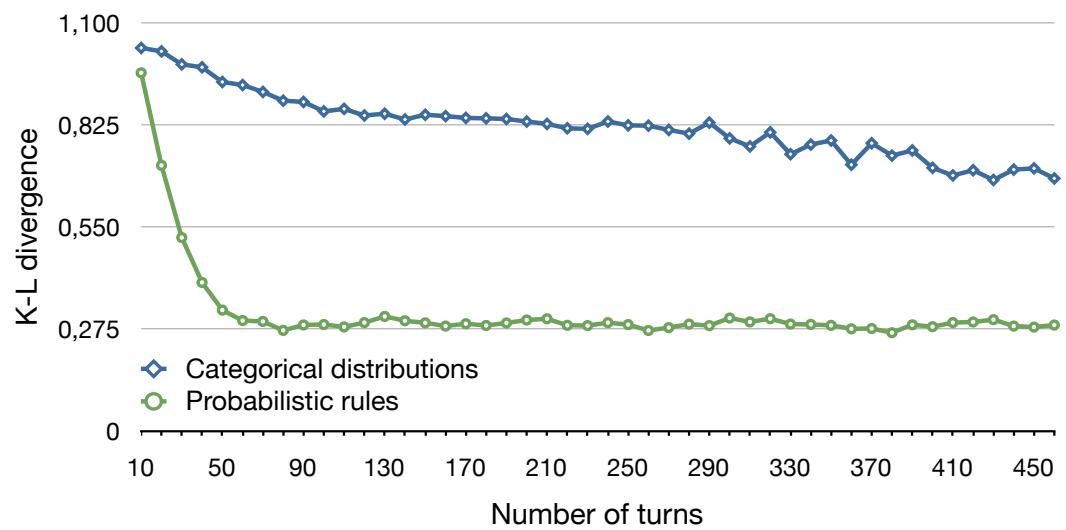


Figure 6.9: Kullback-Leibler divergence between the distribution  $P(a'_u)$  estimated from the current model and the actual distribution followed by the simulator.

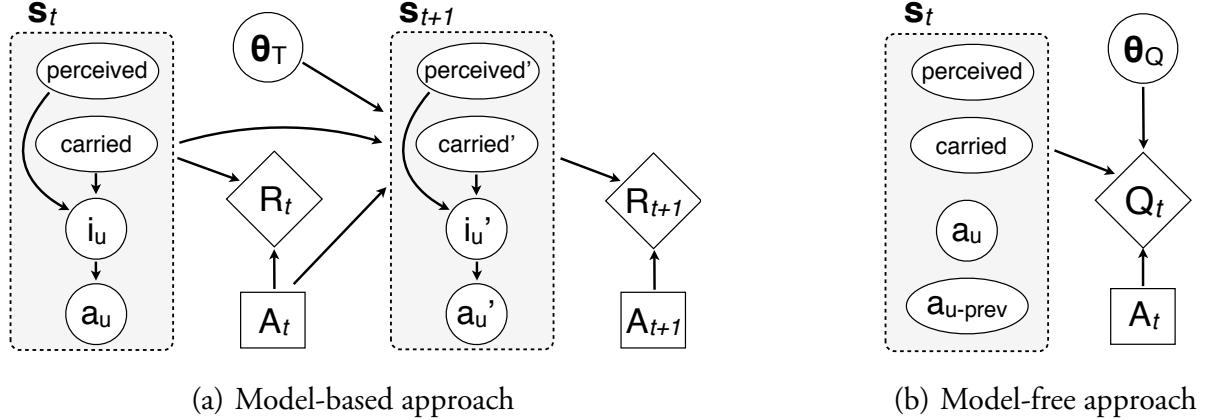


Figure 6.10: Model-based (left) and model-free (right) approaches compared in the experiment.

### Model-based approach

The model-based approach is identical to the one presented in the previous experiment. The transition model is thus structured with six rules associated with 13 corresponding Dirichlet parameters of varying dimensions. The Dirichlet parameters are again initialised with weakly informative Dirichlet priors. As for the first experiment, the action selection was performed with an online planner operating with a horizon of 2.

### Model-free approach

The model-free approach relied on a utility model structured with 12 probabilistic rules associated with 27 Gaussian parameters. As no transition model is here available to infer the underlying user intentions, the dialogue state is defined on the basis of the recent history of dialogue acts, namely the last user action  $a_u$ , last system action  $a_m$  and preceding action  $a_{u-prev}$ . The dialogue state also included the list of objects currently perceived and carried by the robot.

The resulting rule parameters were optimised using the SARSA-based method outlined in Section 6.2.2.

### Empirical results

The simulator was coupled to the dialogue system to compare the learning performance of the two methods. Figure 6.11 shows the average returns over 100 iterations.

### Analysis of results

We can notice on Figure 6.11 that the model-based approach converges to a high-quality policy in fewer episodes than its model-free counterpart. This comes however at the cost of higher computational demands brought about by the need to perform online planning, as evidenced in Figure 6.12 by the roughly similar times required for convergence (around 50 min.). We can also observe that the model-based approach yields slightly higher returns in this experiment, although this difference is harder to explain. One hypothesis is that the model-based approach can accumulate evidence about the underlying user intention in its belief state, while its model-free equivalent lacks an ex-

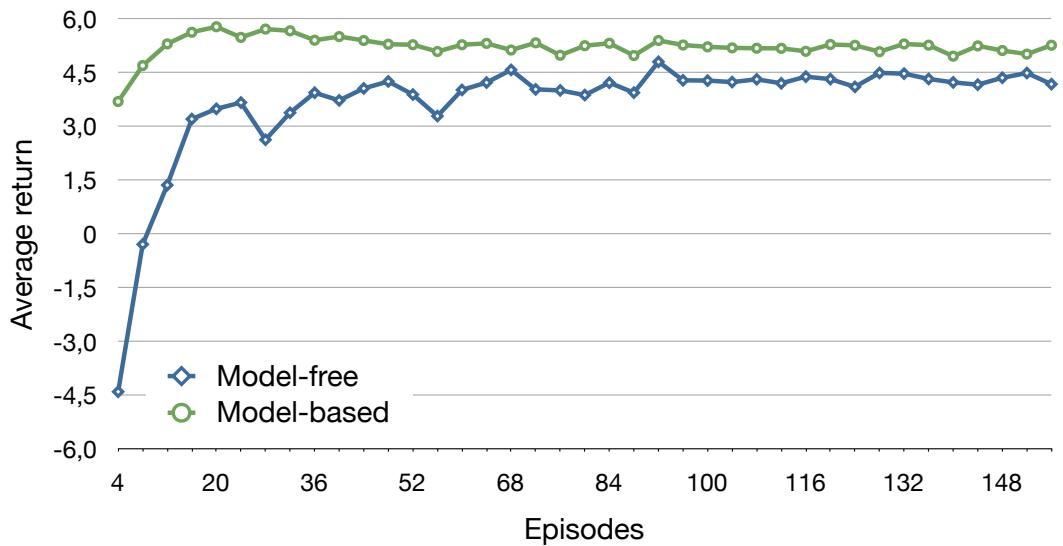


Figure 6.11: Average return as a function of the number of episodes

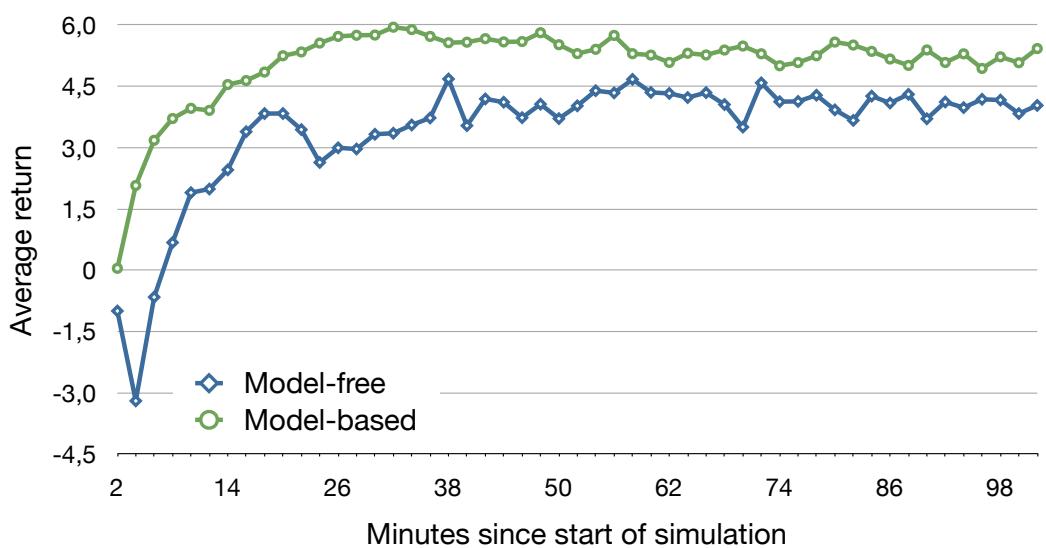


Figure 6.12: Average return as a function of the elapsed time

plicit transition model and can therefore only ground its decisions on the history of dialogue acts and objects perceived by the robot.

The results suggest that the model-based approach outperforms its competitor in this type of learning contexts. This conclusion is however subject to some caveats. First, the tractability of the model-based approach is directly dependent on the length of the planning horizon. Domains with longer horizons might be better addressed with a model-free strategy due to the exponential growth of the search tree. Second, the model-free learning algorithm presented here remains relatively simple – as it is based on a Bayesian variant of the well-known SARSA algorithm – and more advanced frameworks such as Gaussian Processes (?) might yield different model-free results. Finally, it should be noted that the model-based method could rely in this work on the availability of the reward model. This is not an unreasonable assumption for dialogue domains, as the reward model is often a reflection of the application objectives as specified by the system designer. It may nevertheless be difficult to specify all aspects of the reward model at design time. Whether the results carry out to cases where the reward model must also be (partially or fully) estimated remains an open question.

The model-free approach presented in this experiment only focused on the estimation of the action-value model and did not include a transition model. An interesting idea for future work would be to blend the two optimisation methods and simultaneously estimate a transition model together with the action-value model. Such hybrid approach would combine the advantages of model-free and model-based strategies, since it would allow the action-value model to be grounded in high-level variables such as the user intentions instead of remaining confined to observation variables. Action selection could be performed in such hybrid framework via a mixture of offline and online planning as in ?).

## 6.4 Conclusion

We exposed in this chapter two alternative Bayesian reinforcement learning techniques to the optimisation of parameters associated with probabilistic rules.

The first technique is a model-based approach that explicitly constructs statistical models of the domain in the form of transition, observation and reward models. The range of possible values for the domain parameters are represented as prior probability distributions that are incrementally updated based on the observations and rewards perceived during the interactions. Action selection is then achieved through forward planning on the basis of the current dialogue state and domain parameters. Probabilistic rules are employed to structure the domain models and thereby reduce the number of parameters to optimise.

The second technique is a model-free approach that skips the estimation of the domain models in order to directly construct an action-value model expressing the  $Q$  values of state-action pairs. As the  $Q$  values are never directly observed, the update of  $Q$  values is bootstrapped on the basis of Bellman's equation. We described how the action-value model could be structured with utility rules and optimised through a Bayesian variant of the SARSA learning algorithm combined with  $\epsilon$ -greedy action selection.

The last section presented two learning experiments conducted in a simulated environment. The simulator was common to both experiments and constructed on the basis of a preliminary Wizard-of-Oz study carried out in a human-robot interaction domain. The simulator included

three distinct models, all estimated from the collected Wizard-of-Oz data: a *user model* expressing how the user intentions and dialogue acts are likely to evolve as a function of the system actions, a *context model* expressing the objects in the visual field of the robot as well as the ones carried by the robot, and an *error model* introducing errors and inaccuracies into the generated user input in order to mimic the imperfect nature of speech recognition.

On the basis of this simulator, the first experiment compared the performance of model-based Bayesian reinforcement learning on two alternative formalisations of the transition model. The first formalisation relied on traditional factored distributions, while the second represented the transition model through probability rules. The empirical results showed that the rule-structured model could converge to a high-performing dialogue behaviour – as measured by the average return per interaction – much faster than the baseline.

The second experiment examined the relative performance of model-based and model-free approaches. The two compared models were in this experiment structured with probabilistic rules. The results showed that the model-based approach fared slightly better than its model-free counterpart in terms of average return per interaction. We however noted that this difference was contingent on particular aspects of the experimental setup such as the length of the planning horizon and the availability of a reward model.

The previous and present chapter demonstrated how probabilistic rules can facilitate the optimisation of dialogue policies both in the supervised and reinforcement learning case. The experiments have nevertheless so far concentrated on the *learning* performance of rule-structured approaches, and have not yet evaluated the effects of probabilistic rules on measures of interaction quality and user satisfaction. Chapter 8 will address this important question. But before doing so, we first discuss the technical implementation of the data structures and algorithms exposed in this thesis. This is the subject of the next chapter.

# **Chapter 7**

## **Implementation**

This chapter outlines the most important features of the openDial toolkit.<sup>1</sup> openDial is a Java-based software toolkit developed to construct and evaluate dialogue systems based on probabilistic rules. The toolkit aims to be fully generic and domain-independent, since all domain-specific knowledge is captured in the declarative specification of the dialogue domain.

The toolkit implements all the data structures and algorithms detailed in the present thesis and served as a experimental platform to carry out the empirical studies presented in Chapters 5, 6 and 8. Dedicated components have been developed to interface openDial with the Nao robotic platform and adapt the architecture to human-robot interaction settings.

The chapter is divided in two sections. The first section focuses on the design of the openDial toolkit and describes its general workflow, the specification of dialogue domains in XML, and the concrete implementation of the algorithms employed for approximate inference, sampling and forward planning. We also briefly compare the toolkit to other available architectures and discuss its current technical limitations. The second section goes on to explain how the toolkit is integrated and extended into a full-fledged dialogue system for human-robot interaction. The section describes the system architecture (built on top of openDial) as well as the individual components developed to control the verbal and non-verbal behaviour of the robot.

### **7.1 Toolkit design**

#### **7.1.1 Generalities**

The openDial toolkit relies on a blackboard, event-driven architecture. Blackboard architectures are widely used in spoken dialogue systems for their ability to handle flexible workflows where multiple modules “cooperate” to interpret the user inputs, maintain a representation of the dialogue state, and decide on the best actions to perform. In spoken dialogue systems, the blackboard corresponds to the system state, while the modules attached to this blackboard are in charge of specific processing tasks such as speech recognition and understanding, dialogue management, natural language generation and speech synthesis.

---

<sup>1</sup>The name of the toolkit was chosen because of the open design that characterises the framework, and more particularly its extensible, domain-independent architecture with a declarative domain specification.

## System workflow

The modules read and write to the dialogue state in an event-driven manner. After each change, the dialogue state sends an event message to all its attached modules to inform them of the state update. When appropriate, the modules can react to such event and further modify the dialogue state, thereby generating new updates. The process continues until the dialogue state is stabilised.<sup>2</sup> The openDial toolkit allows system modules to run in parallel via multi-threading. The possibility to execute modules in parallel is particularly useful in dialogue systems, as the agent must be able to react to new inputs and contextual changes occurring at any time – even while other modules are still busy processing a previous update. Many of the algorithms developed in the toolkit – such as planning and probabilistic inference – also operate in anytime mode, which implies that they can be gracefully interrupted and deliver their outputs at any time.

The implementation of openDial is currently optimised to run on a single platform.<sup>3</sup> Should such need arise, the framework could however be extended to run on multiple platforms, as the blackboard paradigm generally lends itself well to distributed architectures (?).

## Types of modules

The modules in the openDial toolkit are divided in two distinct types:

1. The first type of module is the rule-structured model already outlined in Section 4.4. A rule-structured model is simply defined as a collection of probabilistic rules together with a list of trigger variables. The update of (at least) one of these trigger variables in the dialogue state results in the instantiation of the probabilistic rules in the dialogue state.
2. The second type of module corresponds to external components such as the speech recogniser, text-to-speech engine and processes for robot perception and control. Given the blackboard architecture of the toolkit, modules can be easily plugged in and out of the system without affecting the rest of the processing pipeline. Similarly to rule-structured models, external components operate by monitoring the dialogue state and updating it when relevant changes are detected.

In the experiments carried out in this thesis, we found it useful to encode not only dialogue management models with the help of probabilistic rules, but also tasks related to natural language understanding and generation. As argued in ?), the expressive power of probabilistic rules allows them to capture the structure of many dialogue processing tasks, not necessarily confined to dialogue management. The dialogue domain designed for the experiments in Chapter 8 include for instance a total of six models: one dialogue act classification model, (triggered by the user utterance  $u_u$ ), one action utility model (triggered by the user dialogue act  $a_u$ ), three probability models to predict the effects of the system action on the context, the user intention and the next user action (all triggered by the system action  $a_m$ ), and a generation model (triggered by the system action  $a_m$ ). The parameters of the NLU and NLG models are however hand-crafted since they do not directly constitute the focus of our work.

---

<sup>2</sup>Recall that a model can only be applied once per update to avoid infinite cycles (cf. Algorithm 7).

<sup>3</sup>System modules can however remotely connect to external resources on the robotic platform to perform tasks related to robot perception and motor control, as explained in Section 7.2.

Compared to traditional architectures in which the components are developed separately and rely on ad hoc representation formats, the use of a shared description formalism (probabilistic rules) to encode multiple reasoning tasks offers several advantages:

- *Transparency*: The reliance on a common representation format provides a unified, transparent semantics for the dialogue state, since all state variables are described and related to one another through a principled framework grounded in probabilistic modelling. This makes it possible to derive a semantic interpretation for the dialogue state as a whole – in terms of e.g. of a joint probability distribution over the state variables.
- *Domain portability*: As most domain-specific knowledge is declaratively specified in the rules, the system architecture is essentially reduced to a generic platform for rule instantiation and probabilistic inference. This declarative design greatly enhances the system portability across domains, since adapting a system to a new domain only requires a rewrite or extension of the domain-specific rules, without having to reconfigure or re-develop other components. This stands in sharp contrast with “black-box” types of architectures where much of the task- and domain-specific knowledge is encoded in procedural form buried inside the implementation of the system components.
- *Flexible workflow*: Probability rules allow for very flexible processing pipelines where state variables are allowed to depend or influence each other in any order and direction. Models can be easily inserted or extended without requiring any change to the underlying platform. Furthermore, several models can be triggered concurrently on the same input/output variables.<sup>4</sup> This allows the system to take advantage of multiple, complementary processing strategies while ensuring that the dialogue state remains consistent.
- *Joint optimisation*: Finally, the use of a unified formalism allows domain models to be optimised jointly instead of being tuned in isolation from one another. Joint optimisation has recently gained much attention in the dialogue system community to overcome the fragmentation of current system architectures and attempt to directly optimise the end-to-end conversational behaviour of the system (see also ?).

Despite these merits, probabilistic rules cannot naturally model all dialogue reasoning tasks. Modules such as speech recognition or speech synthesis depend in particular on external resources and processes and must be integrated separately in the system. The openDial toolkit is designed to incorporate both rule-structured models and external components in its architecture.

### 7.1.2 Specification of dialogue domains

#### Dialogue domains

As already mentioned in Section 4.4, a dialogue domain is represented in openDial by a pair  $\langle \mathcal{B}_0, \mathcal{M} \rangle$  defined by an initial dialogue state  $\mathcal{B}_0$  and a set of models  $\mathcal{M}$ , each model being itself composed of a set of probability or utility rules. Dialogue domains are encoded in an XML format<sup>5</sup> with a syntax specifically designed for the toolkit.

---

<sup>4</sup>Output distributions can indeed handle effect specifications arising from multiple, sometimes conflicting sources, as we have seen in Section 4.3.1.

<sup>5</sup>Extensible Markup Language, cf. <http://w3.org/xml>.

In practice, the specification of a dialogue domain in XML takes the following form:

```

<domain>
    <initialstate>
        <!-- initial state variable values -->
    </initialstate>

    <model trigger="trigger variables for model 1">
        <!-- rules for model 1 -->
    </model>

    ...

    <model trigger="trigger variables for model n">
        <!-- rules for model n -->
    </model>

</domain>
```

The initial dialogue state is represented by a list of state variables, each being associated with a particular probability distribution. Most state variables are defined by a categorical distribution, according to the following skeleton:

```

<variable id="variable name">
    <value prob="prob for first value">first value</value>
    <value prob="prob for second value">second value</value>
    ...
    <value prob="prob for nth value">nth value</value>
</variable>
```

State variables with continuous ranges can also be encoded with uniform, Gaussian, or Dirichlet probability density functions (provided the required parameters are specified).

## Probabilistic rules

Each model specification encompasses a number of probability or utility rules also encoded in XML. Listing 7.1 illustrates an example of probability rule encoded in XML. The rule expresses the probability of the user intention  $Release(X)$ , which corresponds to the action of putting down a carried object  $X$  on the ground. The probability of this user intention is naturally dependent on (1) whether the object  $X$  is currently carried by the robot and (2) whether the previous task was completed. The values of the Dirichlet parameters  $\theta_2$  and  $\theta_3$  are unknown, but we may presuppose that the Dirichlet  $\theta_3$  is highly skewed towards near-zero values for the first dimension, since the probability of the user asking the robot to release an object which is currently not being carried is most likely very small.

Each rule is divided in a number of cases, each containing a (possibly empty) condition and a set of (possibly empty) effects. Conditions can include several sub-conditions combined by conjunction or disjunction operators (the default operator being a conjunction). Each basic condition is denoted by an `if` markup and is composed of a variable, a value, and a relation that must hold between the variable and the value. The default relation is equality, but relations may also correspond to inequalities ( $\neq$ ,  $<$  and  $>$ ), set inclusion ( $\in$  and  $\notin$ ) or string matching operations. Note

that underspecified variables are wrapped in curly brackets { } to distinguish them from normal values. Underspecified variables can occur both in the variables and values of the conditions. Effects are associated with probabilities that can either be fixed or correspond to parameters to learn such as the Dirichlet distributions  $\theta_2$  and  $\theta_3$  in the example. Effects can similarly contain underspecified variables denoted by curly brackets. Although the effects in the example only have one single output variable, rules can in the general case include multiple `set` markups to express complex effects ranging over more than one output variable.

Utility rules are defined in the same manner. An example of utility rule specified in XML is given in Listing 7.2. The rule defines the utility of asking the user to confirm whether the last system action *Demonstrate*( $X$ ) was correct after a certain amount of silence, divided here in three cases (between one and two seconds, between two and three seconds, and after three seconds).

## Parameters

When operating in learning mode, dialogue domains must be associated with parameter variables defined together with their prior probability distribution. This prior can take the form of a categorical, Dirichlet, Gaussian or uniform distributions. As an illustration, the prior distribution for the parameter variable  $\theta_2 \sim \text{Dirichlet}(1, 2)$  is specified as:

```
<variable id="theta_2">
  <distrib type="dirichlet">
    <alpha>1</alpha>
    <alpha>2</alpha>
  </distrib>
</variable>
```

Other types of prior distributions are defined in a similar manner.

### 7.1.3 Core algorithms

We survey below some technical questions related to the openDial implementation of the core algorithms presented through this thesis.

#### Inference

Probabilistic inference forms a key element in the state update process. Two basic types of inference are implemented in openDial :

1. The first is variable elimination, which is an exact inference method initially developed by (?). The algorithm operates by manipulating matrices through summation and pointwise products. The particular implementation in openDial follows the method presented in (?), p. 1101), which generalises classical variable elimination to also include decision networks.
2. The second inference algorithm is likelihood weighting, which is an approximate inference method relying on importance sampling. As explained in Section 3.1.2, likelihood weighting generates samples based on the topological ordering of the graphical model. Each sample is associated with a particular weight that represents its likelihood in the light of the provided evidence. The final estimates correspond to the weighted averages of the samples.

```

<rule>
  <quantifier id="O"/>
  <case>
    <condition>
      <if var="completed-task" value="true" />
      <if var="carried" value="{O}" relation="contains" />
    </condition>
    <effect prob="theta_2[0]">
      <set var="i_u" value="Release({O})" />
    </effect>
    <effect prob="theta_2[1]" />
  </case>
  <case>
    <condition>
      <if var="completed-task" value="true" />
    </condition>
    <effect prob="theta_3[0]">
      <set var="i_u" value="Release({O})" />
    </effect>
    <effect prob="theta_3[1]" />
  </case>
</rule>

```

Listing 7.1: Example of probability rule in XML format

```

<rule>
  <quantifier id="X"/>
  <case>
    <condition>
      <if var="silence" value="3" relation=">" />
      <if var="a_m" value="Demonstrate({X})" />
    </condition>
    <effect utility="theta_{confirmation3}">
      <set var="a_m" value="AskConfirmation" />
    </effect>
  </case>
  <case>
    <condition>
      <if var="silence" value="2" relation=">" />
      <if var="a_m" value="Demonstrate({X})" />
    </condition>
    <effect utility="theta_{confirmation2}">
      <set var="a_m" value="AskConfirmation" />
    </effect>
  </case>
  <case>
    <condition>
      <if var="silence" value="1" relation=">" />
      <if var="a_m" value="Demonstrate({X})" />
    </condition>
    <effect utility="theta_{confirmation1}">
      <set var="a_m" value="AskConfirmation" />
    </effect>
  </case>
</rule>

```

Listing 7.2: Example of utility rule in XML format

Each inference algorithm has its own strengths and weaknesses. Variable elimination is able to deliver exact results and is often the most efficient inference method for small graphical models. It however suffers from scalability problems when applied to models that are densely interconnected and/or include continuous variables. Likelihood weighting is easier to scale to larger networks and can be straightforwardly applied to hybrid models with both discrete and continuous variables. Large numbers of samples are unfortunately required to reach reasonably accurate estimates.

In order to bring together the best of both approaches, a switching mechanism has been integrated to openDial to select the inference method which is best suited to each probabilistic query. The mechanism proceeds as follows. For each probabilistic query, three elements are extracted: the maximum branching factor of the network, the number of continuous variables, and the number of variables specified in the query. These elements are then matched against predefined thresholds. If at least one threshold is exceeded, likelihood weighting is selected to perform the inference, while variable elimination is chosen in the remaining cases.

## Sampling techniques

The use of likelihood weighting necessitates the implementation of efficient sampling techniques for each possible probability distribution. In the interest of the statistically inclined reader, we describe below the sampling methods employed in openDial to efficiently draw values from both discrete and continuous distributions:

- *Categorical distributions:* Sampling a categorical distribution is done through inverse random sampling, following the method described in (?), p. 489). When constructing the distribution, the variable values are sorted according to some particular order (e.g. lexicographic), and a cumulative density function (CDF) calculated relative to this order. Sample values are then extracted at runtime by (1) generating a pseudo-random float number between 0 and 1 and then (2) locating the greatest number in the CDF that is less than or equal to the number just generated. The latter operation is done through binary search. The value corresponding to this number is then selected as the sample.
- *Uniform distributions:* Uniform distributions are directly sampled as  $g(b - a) + a$ , where  $g$  is a pseudo-random number between 0 and 1, and  $a, b$  are the distribution boundaries.
- *Normal distributions:* Normal distributions are sampled in openDial via the well-known Box-Muller method (?), which derives two sample values for a given normal distribution based on two pseudo-random numbers.
- *Dirichlet distributions:* Sampling Dirichlet distributions relies on a slightly more intricate procedure based on Gamma sampling. The first step is to derive  $K$  samples from  $\text{Gamma}(\alpha_i, 1)$  with  $K$  denoting the dimension of the Dirichlet and  $1 \leq i \leq K$ . This sampling procedure is implemented in openDial using the method presented in (?). The sampled Dirichlet value is then defined as  $[x_1, \dots, x_K]$  where  $x_i = \frac{y_i}{\sum_{j=1}^K y_j}$  and  $y_i$  is the sampled Gamma value for dimension  $i$ .
- *Kernel distributions:* We sample non-parametric distributions defined via kernel density estimation (KDE) in two steps. The first is to draw at random one point  $x_i$  from the set of

points  $x_1, \dots, x_n$  included in the KDE. A value is then drawn from the kernel associated with the point. In our case, this corresponds to drawing a sample from the normal distribution  $\mathcal{N}(x_i, h)$  centered at  $x_i$  and of variance  $h$  (the bandwidth).

## Forward planning

The implementation of the forward planning algorithm closely follows the procedure outlined in Section 6.2.1. The search tree is constructed in a breadth-first manner, until either the planning horizon has been reached or the planner has run out of time. The latter condition relies on the use of a time-out function to ensure that the planner does not exceed specific time limits.

In order to generate a set of possible observations in a given state  $\mathcal{B}$  (line 6 of Algorithm 13), the planner locates all predictive state variables (denoted with a superscript  $p$ ) currently present in the dialogue state and draws a sample value for each. In the experiment presented in Section 6.3, the predictive variable corresponds to the next user action, and the generated observations will therefore reflect possible values for this user action.

For tractability reasons, openDial limits the maximum number of actions and observations that are branched out at each point in the search tree. The actions are selected on the basis of their reward values in the current state – i.e. only the  $n$  actions with highest reward are selected, with  $n$  corresponding to an arbitrary threshold. The observations are filtered based on their likelihood of occurrence. The planner thus only expands the tree with the  $m$  most likely observations, where  $m$  is another threshold. The two thresholds are currently manually tuned, but could in principle correspond to additional parameters to optimise during learning.

### 7.1.4 Comparison with other architectures

The construction of generic, domain-independent architectures is a recurring theme in dialogue systems research, and there is a clear trend towards the development of platforms composed of more generic or reusable components. We present below the most important architectures currently used in the field and contrast their design with the one followed in the openDial toolkit. Finally, we discuss the current limitations of the presented framework.

#### Existing software frameworks

Information-state approaches are closely related to the framework presented in this chapter. The TrindiKit architecture presented by ?) relies on a shared information state accessed by various system modules and a rich repository of rules. A control module is in charge of the system scheduling for the whole architecture. The system modules can also connect to external resources such as databases and plan libraries. The TrindiKit is a platform for constructing and evaluating dialogue engines and is designed to be fully domain-independent. The related DIPPER architecture described in (?) is built on similar principles as the TrindiKit, but simplifies the architecture and the encoding format for the update rules. DIPPER employs the Open Agent Architecture as communication protocol.

The idea of domain independence is also taken up by plan-based approaches such as TRIPS (The Rochester Interactive Planning System, cf. ?). TRIPS uses an agent-oriented architecture comprised of multiple modules working together to recognise the intentions of the human user

and fulfilling the system goals. Most reasoning tasks are explicitly cast as planning problems, from high-level planning to response planning and surface realisation.

One of the most mature platform for prototyping dialogue systems is Olympus and its associated dialogue management framework Ravenclaw (??). The Olympus architecture is built on top of a centralised message-passing infrastructure in which modules can be plugged in and out to suit the needs of the application. Ravenclaw is a plan-based, task-independent dialogue engine which is fully integrated in Olympus. Ravenclaw supports mixed-initiative interaction and integrates dedicated functions for conversational skills such as error handling, timing and turn-taking. Action selection is based on a hierarchical decomposition of tasks whose execution is sequentially ordered via an agenda.

The agent-based Jaspis architecture (?) also has multiple points of contact with the openDial toolkit, as it similarly revolves around a shared representation of the system state. Jaspis components are themselves split into agents (in charge of decision-making), evaluators (in charge of selecting the most suitable agent in a particular situation) and managers (in charge of the general coordination of the components). Jaspis is designed to allow for distributed setups with dedicated mechanisms for the coordination and synchronisation of concurrent modules. The architecture also aims to facilitate system-level adaptivity. Compared to the openDial toolkit, Jaspis offers more advanced support for distributed and parallel setups, but at the cost of an increased system complexity.

Most MDP and POMDP-based dialogue architectures line up system components in a single processing sequence. The prototype systems developed for the TALK and CLASSIC projects (??) and the related BUDS POMDP system<sup>6</sup> are structured into pipelines where each component takes a probability distribution over input hypotheses and generates another distribution over possible outputs.

## Comparison and discussion

The openDial toolkit can be seen as an attempt to combine the flexibility of information-state architectures with the robustness and adaptivity of statistically optimised dialogue systems. In line with logic-based approaches (e.g. TrindiKit, DIPPER, TRIPS, Olympus/Ravenclaw, and Jaspis), the workflow of openDial is designed to allow for multiple processing paths and complex, context-sensitive reasoning strategies. And in line with MDP and POMDP-based approaches, the toolkit is also able to explicitly handle uncertain knowledge and stochastic relations between variables thanks to its probabilistic representation of the system state, as well as optimise domain models from interaction data via statistical inference.

The idea of combining statistical and symbolic approaches to dialogue processing is certainly not new in the literature on spoken dialogue systems. In many dialogue architectures, probabilistic reasoning techniques coexist with classical symbolic components limited to unique, deterministic inputs. Very often, this integration of heterogeneous statistical and symbolic components is achieved by reducing N-best lists to their single most likely hypothesis. An important drawback of such approach is however the substantial loss of information that result from this reduction.<sup>7</sup>

---

<sup>6</sup>Bayesian Update of Dialogue State, cf. <http://mi.eng.cam.ac.uk/~mh521/nipsdemo12/>

<sup>7</sup>For instance, speech recognition typically provides explicit measures of uncertainty in the form of e.g. confidence scores, but these confidence measures are often lost at higher reasoning stages such as syntactic parsing, semantic interpretation and dialogue management.

One can also observe some notable design differences in the representation of the action selection process. Many dialogue architectures indeed decompose dialogue management into two or more distinct behavioural layers. The Olympus framework incorporates for instance both an interaction manager in charge for the low-level control of the conversational floor, and a dialogue manager in charge of higher-level dialogue decisions. The TRIPS architecture similarly divides dialogue management in a cluster of modules encompassing discourse management, discourse context management, plan management, and a behavioural agent in charge of controlling the overall behaviour. The openDial toolkit leaves by comparison the system designer free to frame decision-making in one, two or more layers, depending on the particular needs of the domain.<sup>8</sup>

## Limitations

One important aspect of dialogue architectures that has not been really touched upon in our work is the question of incremental processing. As explained in the background chapter, dialogue processing should ideally be performed incrementally and output partial hypotheses as early as possible in the system workflow. The InproTK architecture presented in ??) is specifically designed to support across-the-board incremental processing, from speech recognition to dialogue management and speech synthesis. Although the current implementation of openDial cannot claim to be incremental, we do nevertheless hypothesise that the formalism of probabilistic rules could be extended to allow for incremental processing without major difficulties, since the chain of related hypotheses is already explicitly captured in the conditional dependencies generated by the instantiated rules. An update in the probability distribution over some observations (e.g. the user utterance) is therefore directly reflected in all hypotheses that depend on it (e.g. the corresponding user intention). Modifying the state update algorithm to run in a fully incremental manner without degrading the system performance is however a non-trivial engineering task, which could constitute an interesting topic for future work. Similarly, the turn-taking behaviour integrated in openDial remains relatively rudimentary and could certainly be improved to allow for e.g. barge-ins and fragmented utterances.

While the dialogue system used for our experiments does include basic modules for robot perception and control (cf. next section), the architecture does not yet scale to full-scale multi-modal processing, as such extensions would require dedicated mechanisms for information fusion and fission. Our previous work on situated human-robot interaction (?) illustrates how dialogue processing can be coupled to sensorimotor components in a bidirectional manner.

On a more practical note, it should finally be pointed out that openDial is at the time of writing at an early stage of development and does not benefit from the years of incremental refactoring and testing shown by more mature architectures such as TrindiKit or Olympus/Ravenclaw. Further development work is certainly needed to move openDial from being an advanced research prototype to a stable, robust platform capable of being deployed in any application domain. Of special importance is the integration of user-friendly authoring tools to allow human designers to write, organise and evaluate rule-structured dialogue domains without having to delve into the technical intricacies of XML syntax. openDial would also certain benefit from more advanced performance tuning to enhance the system reactivity and scalability to larger domains.

---

<sup>8</sup>This can be practically realised by creating distinct utility models for action selection. Hierarchical decision policies can notably be captured by lining up utility models in a top-down cascade of triggers.

## 7.2 System integration

The openDial toolkit was used as a foundation to construct a concrete, end-to-end dialogue system for human-robot interaction. We detail in the next pages the practical design of this dialogue system as well as the graphical user interface implemented to monitor and control the system behaviour (and its internal dialogue state) in real-time.

### 7.2.1 Architecture

In addition to the openDial core components – comprising the dialogue state and its update mechanism based on probabilistic rules – the dialogue system employed in our experiments also integrated specific modules for speech recognition and synthesis, robot perception and motor control. The generic architecture for the system is shown in Figure 7.1.

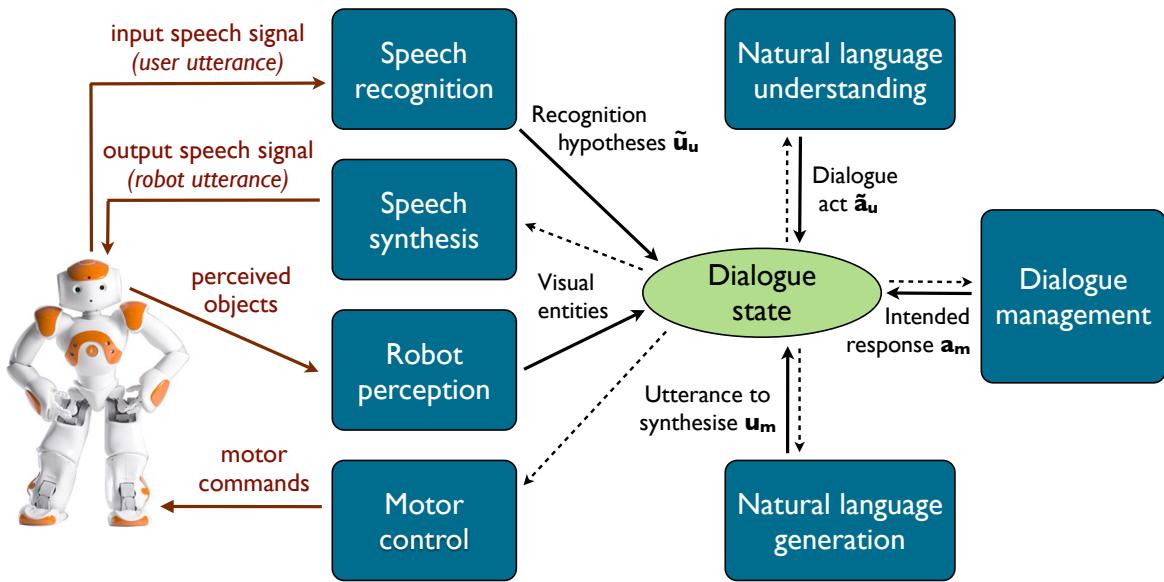


Figure 7.1: Generic system architecture employed in the experiments.

The system modules can operate in either synchronous and asynchronous mode:

- Synchronous modules continuously monitor the dialogue state for relevant changes. Their activation is thus in synchrony with the update events generated by the dialogue state.
- Asynchronous modules run independently of the dialogue state. They typically relate to visual or speech perception tasks. When new observations are made, they update the dialogue in accordance with the recognised user utterances or perceived contextual changes.

All modules have access to the complete dialogue state and can therefore exploit the full set of state variables (including generic contextual information) in their processing.

### 7.2.2 Individual modules

We describe below the modules shown in Figure 7.1 and explain their role and internal structure. It should be emphasised that the focus of the present thesis is on dialogue management. The other

system modules are therefore deliberately limited to simple, “shallow” processing methods in order to concentrate the implementation efforts on the dialogue manager. Many of these modules could however be extended to employ more sophisticated techniques, in particular in regard to speech recognition, natural language understanding and generation.

## Speech recognition

Speech recognition is performed on the robot platform, using a commercial, off-the-shelf speech recognition engine (Vocon 3200 from Nuance). Four microphones placed on the robot head are used for the sound capture. The placement of the microphones on the robot allows the user to interact with the robot in a natural manner, without needing to resort to head-mounted microphones. This advantage comes however at the price of a slightly degraded sound quality due to the larger (and varying) distance between the sound source and the microphones, combined with the presence of noisy mechanical motors placed at only a few centimetres from the microphones.

The acoustic model employed in all experiments is provided along with the speech recognition package and is optimised for American English. The language model takes the form of context-free recognition grammars in Bachus-Naur Form (BNF). Distinct grammars were used to cover the domain of discourse of each experiment. The grammars were designed by hand, based on the Wizard-of-Oz transcripts collected in our empirical studies (cf. previous chapters). Grammars can be dynamically attached or removed from the engine at runtime, thereby allowing the system to adapt the recognition to the current dialogue context. This functionality is however not yet exploited in the current architecture – but see e.g. [?](#)) for a description of our prior research work on this issue. As the recognition engine only generates hypotheses with raw, unnormalised confidence scores, a normalisation routine is used to convert them to proper probability distributions  $P(\tilde{u}_u)$ .<sup>9</sup>

## Natural language understanding

The goal of natural language understanding (NLU) is to map a collection of utterance hypotheses  $\tilde{u}_u$  to a corresponding set of dialogue act hypotheses  $\tilde{a}_u$  expressing the semantic and pragmatic content of the user input. This understanding step is decomposed in our implementation in two tasks, dialogue act recognition and visual reference resolution.

The goal of dialogue act recognition is to construct the logical form representing the pragmatic meaning of the utterance. One should note that user utterances may contain more than one dialogue act, as for instance in “yes and now pick the blue object”. A collection of domain-specific templates was designed by hand to convert surface forms into logical representations of dialogue acts. Although this approach does not allow for “deep” semantic extraction, it was shown to perform well in our dialogue domains. Future work may replace this template-based method with a data-driven semantic parser based on e.g. dependency parsing ([?](#)).

Reference resolution is used to map linguistic expressions referring to objects in the visual context to their corresponding object identifier. The properties stated in the linguistic expressions are first matched against the set of possible references. If the description remains ambiguous (i.e. more than one object matches the linguistic expression), the references can be further ranked according

---

<sup>9</sup>The conversion between confidence scores and probabilities is manually tuned in the current setup. Future work may however rely on more principled estimation techniques such as the ones outlined in [?](#)).

to their visual saliency, defined here in terms of their physical distance to the robot. The personal pronoun “it” is resolved by searching for the closest object reference in the dialogue history.

Natural language understanding is practically implemented in openDial via probabilistic rules. As seen in Section 4.5, the formalism of probabilistic rules already includes special-purpose operators for string manipulation and can thus readily encode the shallow templates used for dialogue act recognition. Rule  $r_{15}$  below is an example of such rule. The rule lists three regular expression patterns associated with the dialogue act MoveArm(Left, Down). If the value for the user utterance variable  $u_u$  matches at least one of the patterns, the dialogue act  $a_u$  is classified as MoveArm(Left, Down):

$$r_{15} : \begin{aligned} & \mathbf{if} (u_u \text{ matches } “(*) \text{ left arm down}”) \\ & \vee (u_u \text{ matches } “(*) \text{ lower (the | your) left arm}”) \\ & \vee (u_u \text{ matches } “(*) \text{ down (the | your) left arm}”) \quad \mathbf{then} \\ & \left\{ P(a_u = \text{MoveArm(Left, Down)}) = 1.0 \right. \end{aligned}$$

## Dialogue management

Dialogue management follows the procedure outlined in Section 4.4 and will thus not be repeated here. For each update of the dialogue state, the dialogue manager triggers the corresponding rule-structured models, and selects the next action to perform (if any).

## Natural language generation

If the system action selected by the dialogue manager is non-empty and denotes a verbal action, the natural language generation module is triggered. As for natural language understanding, the generation component of openDial is based on a manually designed collection of templates. The processing direction is however the reverse one of natural language understanding, as the templates are applied to convert a logical representation of the communicative goal into a surface form.

The generation templates are also encoded with probabilistic rules. The rules correspond here to utility rules, since natural language generation is at its core a decision-making task corresponding to the choice of a particular linguistic realisation for the intended communicative goal. As an example, rule  $r_{16}$  generates the system response  $u_m$  given the system act  $a_m = \text{Acknowledgement}$ . The rule specifies in this case three alternatives with equal utility:

$$r_{16} : \begin{aligned} & \mathbf{if} (a_m = \text{Acknowledgement}) \quad \mathbf{then} \\ & \left\{ \begin{array}{l} U(u'_m = “ok”) = 1 \\ U(u'_m = “great”) = 1 \\ U(u'_m = “thanks”) = 1 \end{array} \right. \end{aligned}$$

The specification of multiple realisations allows for some variation in the system behaviour, since the system will automatically select one realisation at random, due to the uniform utility assigned to the alternative realisations.

## **Speech synthesis**

Speech synthesis is performed on the robot, using an off-the-shelf speech synthesis engine developed by Acapela<sup>10</sup>. The synthesis engine is based on unit selection. The output speech signal is then sent to two speakers placed on the robot head. To avoid spurious recognition results, the speech recognition is automatically disabled when the robot is speaking.

## **Robot perception**

The robot can detect simple physical objects present in the visual scene. The object detection is done based on the vision libraries bundled with the robotic platform. Special markings are placed on top of the objects to facilitate the object recognition and the visual servoing.

## **Robot motion control**

Various types of physical movements were engineered for the purpose of our experiments, including:

- generic body movements: rotating the arms and the head in various directions,
- spatial navigation: moving forward and backward, turning left and right,
- object manipulation: grasping and releasing objects.

All the movements were programmed using the motion control libraries available on the robot. The object manipulation relies on the use of permanent magnets attached to the robot hands.

### **7.2.3 Graphical interface**

The graphical user interface developed for the openDial toolkit enables the system designer to monitor and control in real-time the state of the system. The interface is divided in two views, shown as distinct tabs in the application window: the chat window and the dialogue state monitor.

#### **Chat window**

The chat window presents the interaction history as a chat window, as illustrated by the screenshot in Figure 7.2.

The user inputs are shown as N-best lists together with their corresponding probabilities. In addition to monitoring the interaction, the chat window can also be used to test the dialogue system by typing new user and system inputs in the input field at the bottom of the window. The agent role can be switched in the drop-down field in the bottom right corner.

#### **Dialogue state monitor**

To allow the system designer to easily inspect the content of the dialogue state, a state visualisation tool has also been integrated into openDial . The monitor provides a visual representation of the dialogue state in the form of a directed graph with nodes corresponding to the state variables and directed edges corresponding to conditional dependencies.<sup>11</sup> An example of graph layout is shown

---

<sup>10</sup><http://www.acapela-group.com>

<sup>11</sup>The graphs are rendered with JUNG, an open source toolkit for drawing graphs: <http://jung.sourceforge.net>.

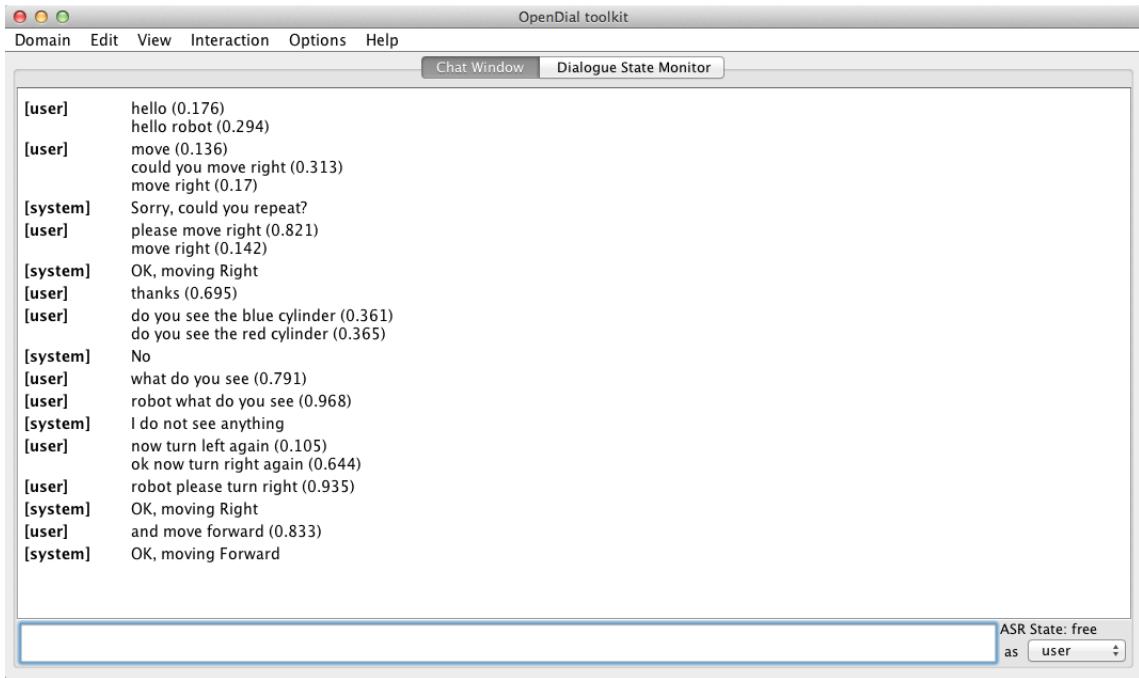


Figure 7.2: Graphical user interface showing the interaction history.

in Figure 7.3. The graph is dynamically refreshed after each update of the dialogue state, using standard graph drawing algorithms to optimise the layout of nodes on the screen.

In addition to depicting the current dialogue state, the dialogue state monitor can also record and store previous dialogue states. The dialogue state to visualise can be selected among the list on the left side of the window. This functionality is particularly useful to compare dialogue states with one another and analyse how the dialogue state is evolving over time.

The user can manipulate the graph in multiple ways in order to e.g. analyse the content of specific state variables, add or remove evidence, or request the calculation of marginal distributions on selected set of variables. The inference results are in such case shown in the text area at the bottom of the window. In addition, the system designer can also zoom in on selected probability distributions using the distribution viewer tool illustrated in Figure 7.4. Discrete probability distributions are shown as histograms, while continuous probability distributions are represented by their probability density functions.<sup>12</sup>

### 7.3 Conclusion

This chapter exposed the practical implementation of our structured modelling framework in the openDial toolkit and its instantiation in a spoken dialogue system for human-robot interaction. The first section presented the general architecture of the toolkit, the declarative specification of dialogue domains in a XML format, and the integration of efficient algorithms for inference, sampling and planning. We notably discussed the benefits of using a shared description formalism in terms of transparency, portability, flexibility and adaptivity over traditional “black-box” architectures. We also compared openDial to other existing dialogue system architectures, and pointed out a number

<sup>12</sup>The probability distributions are rendered with the open source toolkit JFreeChart: <http://jfreechart.sourceforge.net>.

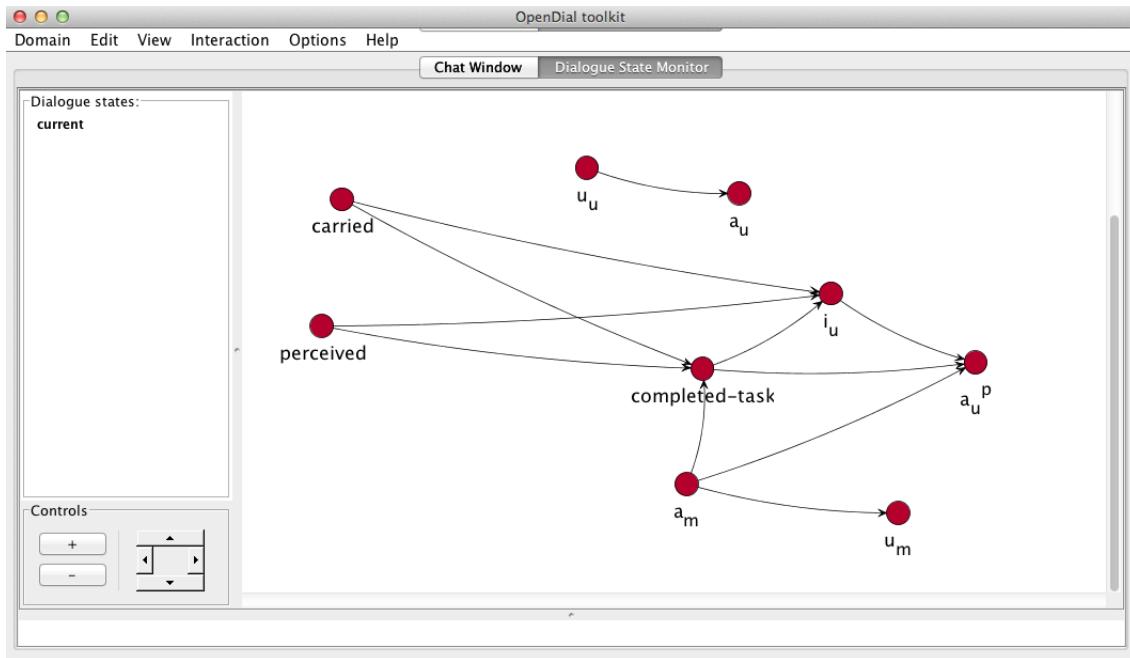


Figure 7.3: Visualisation of the current dialogue state.



Figure 7.4: Visualisation of a discrete probability distribution  $P(a_u^p)$  and a continuous probability distribution  $P(\theta_{15})$  in the dialogue state monitor.

of limitations in the current implementation of the toolkit, such as its lack of incremental processing and its limited turn-taking behaviour.

The openDial toolkit is inspired by both symbolic and statistical approaches to dialogue, and combines an information-state architecture with probabilistic reasoning based on rule-structured models. As stated in the introduction chapter, the long-term goal of the openDial framework is to bridge the gap between symbolic approaches to dialogue management, which usually concentrate on capturing rich interaction patterns, and probabilistic approaches, more focused on aspects related to noise, uncertainty, and adaptivity.

The hybrid design of openDial allows system designers to exploit powerful generalisations in the dialogue domain specification without sacrificing the probabilistic nature of the model. Another important side benefit of probabilistic rules is their improved readability for human experts, which are able to leverage their domain knowledge in the form of pragmatic rules, common sense assumptions, or task-specific constraints. Furthermore, the internal organisation of rules into models enables dialogue domains to be specified in a modular fashion, by clustering rules into distinct models. Some models may therefore reflect highly domain-specific knowledge while others encode generic interaction patterns that can be easily ported to other applications.

The last section described the integrated dialogue system used to carry out the practical experiments presented in Chapters 5, 6 and 8. The system comprised both synchronous and asynchronous components and included dedicated modules for speech recognition, speech synthesis, robot perception and motor control. Natural language understanding, dialogue management and natural language generation were encoded with models structured with probability rules.

The next chapter demonstrates the practical deployment of this system in a full-scale user evaluation experiment.



# **Chapter 8**

## **User evaluation**

### **NOT READY YET!!**

This chapter presents the last (and most extensive) experiment carried out in this thesis. The empirical studies exposed so far in Chapters 5 and 6 essentially focused on the empirical learning performance of rule-structured models compared to more classical representations. But albeit such measures can provide useful insights about the model adequacy for various dialogue domains, they are not by themselves sufficient to assess the suitability of a particular modelling approach and its practical effects on the quality of the produced interactions.

In order to corroborate our claims regarding the benefits of hybrid approaches to dialogue management and contrast it to traditional methods, we conducted an user evaluation experiment with a total of 37 users. The aim of the experiment was to compare three alternative approaches to dialogue management (two baselines and our own approach) in an human–robot interaction domain. The three approaches respectively correspond to a purely handcrafted approach, a purely statistical approach, and a hybrid approach based on probabilistic rules. The empirical results collected through this experiment show that the hybrid, rule-structured approach is able to outperform the two baselines on most of the metrics defined for the domain. The metrics include not only objective metrics extracted from the interaction logs but also subjective metrics of user satisfaction, based on a survey completed by the participants after each interaction.

The chapter is structured in three sections. The first section describes the human-robot interaction scenario within which the experiment is carried out, and details both the state variables, system architecture, and training data made available for the domain. The second section then exposes the three alternative dialogue management approaches developed for the experiment, with a particular emphasis on the design choices that characterise each strategy. Finally, the third section presents the user evaluation itself, and surveys the experimental setup, quality metrics, and of course the obtained empirical results and their analysis.

### **8.1 Interaction scenario**

The interaction scenario employed for the user evaluation is similar in most respects to the one presented in Section 6.3, modulo some minor adaptations to circumscribe more precisely the task that is to be performed.<sup>1</sup>

---

<sup>1</sup>This change was suggested by several participants in the previous study.

The interactions revolve around a human user and a Nao robot standing on a large table, as illustrated in Figure 8.1. The purpose of the interaction is to instruct the robot to (1) move to the other side of the table without bumping into the (imaginary) walls placed in the middle of the scene, (2) pick up the designated object, (3) bring it back to the left side of the table, and (4) release it on a yellow landmark. The robot is able to perceive the presence, colour and location of each physical object thanks to visual markings placed at their top. It can also grasp each object with the help of permanent magnets fixed on the hands of the robot.

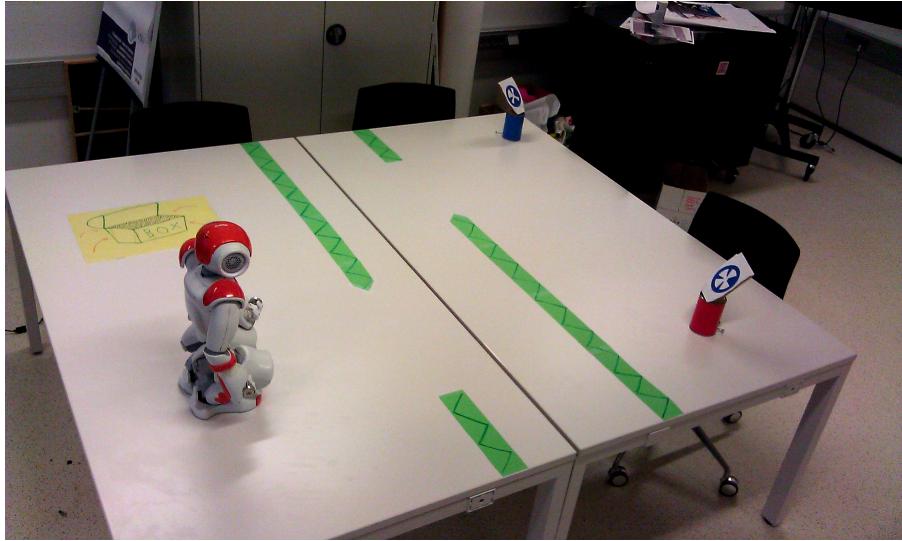


Figure 8.1: Interaction scenario for the experiment. On the right side of the table stand two objects (one red and one blue). Imaginary walls (in green) are also placed on the table, as well as a yellow landmark depicting the final destination of the robot.

### 8.1.1 Dialogue domain

#### User intentions and actions

As for the previous experiment, we represent each basic instruction (such as moving forward, turning left, picking up an object, etc.) as a distinct user intention  $i_u$ . After the fulfilment of each instruction, the user intention is reinitialised to represent the next instruction to perform. The possible user intentions for the domain are shown in Table 8.1.

A few changes can be noted compared to the Wizard-of-Oz study presented in Section 6.3. The basic movements are now completed with a second argument specifying the duration of the movement, e.g. short or long. The human user can therefore explicitly request the robot to perform a short or long movement. A new type of movement has also been added, Move(Turn) to make the robot turn around by 180 degrees. Finally, specific intentions are used to capture the opening (engagement) and closing (disengagement) phases of the dialogue.

The possible user actions  $a_u$ , listed in Table 8.2 correspond to the verbal realisations of these user intentions. The user can directly utter an instruction, ask the robot to repeat its last movement, or respond to a clarification or confirmation request. The user action Ask(Stop) can also be used

to command the robot to stop its current movement, while the grounding action Grounding represents explicit feedback provided by the user on the robot behaviour.

### System actions

As shown in Table 8.3, the system actions  $a_m$  include:

- Physical movements to move around in various directions and lengths, pick up or put down physical objects on the table, or stop the current move.
- Verbal responses to the user requests, in order to confirm or deny the detection of a particular object, describe the robot's current perception, or tell the user that it cannot perform a requested movement due to some impeding factor (e.g. not being able to pick up an object whose location is currently not known).
- Clarification and confirmation requests, such as asking the user to repeat her/his last utterance, or confirming a particular intention.

Instead of representing grounding actions in a separate category of actions (as in Table 6.3), the grounding acts are now directly coupled to the system actions. The selection of the system action  $\text{Do}(x)$  therefore triggers both a grounding action ("Ok, now doing  $x$ ") and the simultaneous execution of the corresponding physical movement.

### Dialogue state

The dialogue state  $\mathcal{B}$  chosen for the dialogue domain is factored in eleven distinct state variables, encoding both the user intentions and actions, the external context, the recent interaction history, and the physical status of the robot. Table 8.4 details the name, purpose, range of possible values, observability and dependencies of each state variable.

One can distinguish three distinct types of state variables in this factorisation:

1. State variables such as  $u_u$ , *perceived*, *carried* and *motion* represent observations arising from the robot sensors and speech recognition engine. Observations such as *carried* and *motion* are (near-)certain, while the last user utterance and the set of perceived objects are uncertain observations expressed through categorical distributions over alternative values.
2. State variables such as  $i_u$ ,  $a_u$ ,  $a_{u\text{-}prev}$  and *completed-task* correspond to hidden variables that must be indirectly inferred from the accumulated observations and actions.
3. Finally, variables such as  $a_m$ ,  $u_m$  and *lastMove* represent (present or past) decisions made by the system. These decisions are naturally known to the system and are thus fully observed.

Even when abstracting from the two variables  $u_u$  and  $u_m$  (which have a virtually infinite set of possible values, as they can accept any string) and assuming the presence of only two objects, the

- Move( $x, y$ )
  - where  $x = \{\text{Left, Right, Forward, Backward}\}$
  - and  $y = \{\text{Short, Long}\}$
- Move(Turn)
- PickUp( $x$ )
  - where  $x$  is an object identifier
- ReleaseObject
- WhatDoYouSee
- DoYouSee( $x$ )
  - where  $x$  is an object identifier
- Greeting
- Closing

Table 8.1: List of user intentions  $i_u$

- Ask(Move(x, y))
  - where  $x = \{\text{Left, Right, Forward, Backward}\}$
  - and  $y = \{\text{Short, Long}\}$
- Ask(Move(Turn))
- Ask(PickUp(x))
  - where  $x$  is an object identifier
  - or Other if the reference is not resolved
- Ask(ReleaseObject)
- RepeatLast
- Ask(Stop)
- Ask(WhatDoYouSee)
- Ask(DoYouSee(x))
  - where  $x$  is an object identifier
  - or Other if the reference is not resolved
- Confirm
- Disconfirm
- Grounding
- Greet
- Close
- Other

Table 8.2: List of user actions  $a_u$

- Do( $x$ )
  - where  $x = \{\text{Move}(y, z), \text{Move}(\text{Turn}), \text{PickUp}(a), \text{ReleaseObject}, \text{Stop}\}$
  - and  $y = \{\text{Left, Right, Forward, Backward}\}$
  - and  $z = \{\text{Short, Long}\}$
  - and  $a$  = an object identifier
- Excuse( $x$ )
  - where  $x = \{\text{UnseenObject, UnresolvedReference, NoCarriedObject, AlreadyCarryObject}\}$
- Greet
- Goodbye
- Describe( $x$ )
  - where  $x$  = a (possibly empty) list of object identifiers
- ConfirmDetection
- DisconfirmDetection
- AskClarify
- AskConfirm( $x$ )
  - where  $x = \{\text{Move}(y, z), \text{Move}(\text{Turn}), \text{PickUp}(a), \text{ReleaseObject}, \text{Stop}\}$
  - and  $y = \{\text{Left, Right, Forward, Backward}\}$
  - and  $z = \{\text{Short, Long}\}$
  - and  $a$  = an object identifier

Table 8.3: List of system actions  $a_m$

total size of the joint state space is non-trivial to tackle:

$$\begin{aligned}
 |\mathcal{S}| = & \quad 4 && [\text{size of } Val(\text{perceived})] \\
 & \times 3 && [\text{size of } Val(\text{carried})] \\
 & \times 18 && [\text{size of } Val(i_u)] \\
 & \times 25 && [\text{size of } Val(a_u)] \\
 & \times 25 && [\text{size of } Val(a_{u-\text{prev}})] \\
 & \times 41 && [\text{size of } Val(a_m)] \\
 & \times 2 && [\text{size of } Val(\text{motion})] \\
 & \times 14 && [\text{size of } Val(\text{lastMove})] \\
 & \times 2 && [\text{size of } Val(\text{completed-task})] \\
 = & \quad 30\,9960\,000
 \end{aligned}$$

### 8.1.2 System design

The dialogue system employed for the experiment follows the architecture described in the previous chapter (Section 7.2). We briefly describe below the configuration of each component. It should be stressed that the only component that varies across the three evaluated approaches is the dialogue manager. All other modules are fixed and remain identical for all interactions.

**Speech recognition:** The speech recognition engine relies on a hand-written recognition grammar of limited coverage. The grammar used for the experiment is shown in Table 8.5. The speech recognition engine runs directly on the robot platform based on the audio signals captured by four microphones on the robot head.

**Natural language understanding:** Natural language understanding is implemented in the dialogue system through probability rules. A total of 20 rules (of varying size) have been written for the domain, based on simple template-matching methods. One rule is responsible for the resolution of referring expressions based on the current visual context.

**Dialogue management:** Three distinct types of dialogue managers have been developed for this experiment: a finite-state automata, a purely statistical dialogue manager, and a dialogue manager based on probabilistic rules. The development of these three dialogue management approaches is presented in Section 8.2.

**Generation:** The natural language generation is realised through the application of one single rule (with 37 distinct cases) responsible for the mapping between the selected system action and its verbal realisation. The generated sentence is then converted to a speech signal with the text-to-speech library installed on the robot platform.

**Other modules:** As detailed in the previous chapter, the integrated dialogue system also comprises specific modules dedicated to object detection and motor control. The detection and localisation of the physical objects is facilitated by the use of visual markings on top of the objects. The robot uses permanent magnets to pick up and hold objects in its hand.

Variable label:	Description:	Range of values:	Observability:	Dependencies:
$perceived$	(Possibly empty) set of perceived objects	Object identifiers	Partial	$\emptyset$
$carried$	(Possibly empty) set of carried objects	Object identifiers	Full	$\emptyset$
$i_u$	Current user intention	cf. Table 8.1	Partial	<i>completed-task, perceived, carried</i>
$u_u$	Last user utterance	ASR hypotheses	Partial	$\emptyset$
$a_u$	Last dialogue act from the user	cf. Table 8.2	Partial	$u_u, i_u, a_m, lastMove$
$a_{u-prev}$	Next-to-last dialogue act from the user	cf. Table 8.2	Partial	$\emptyset$
$a_m$	Last system action	cf. Table 8.3	Full	$i_u, a_u, motion, perceived, carried$
$motion$	Activity status of the robot (denoting whether the robot is currently moving)	<i>true or false</i>	Full	$\emptyset$
$lastMove$	Last physical movement executed by robot	System actions of the form $Do(x)$	Full	$a_m$
$u_m$	Last system utterance	NLG candidates	Full	$a_m$
$completed-task$	Completion status of the user intention (denoting whether the current intention $i_u$ has been fulfilled by the last action)	<i>true or false</i>	Partial	$i_u, a_m$

Table 8.4: List of state variables defined for the domain.

$\langle TopRule \rangle ::= \langle Move-1 \rangle \mid \langle Move-2 \rangle \mid \langle PickUp \rangle \mid \langle Release \rangle \mid \langle Perception-1 \rangle \mid \langle Perception-2 \rangle \mid \langle Confirmation \rangle \mid \langle Grounding \rangle \mid \langle Repeat \rangle \mid \langle Opening \rangle \mid \langle Closing \rangle \mid \langle Stop \rangle$   
 $\langle Move-1 \rangle ::= \langle Front \rangle (\text{walk} \mid \text{go} \mid \text{move} \mid \text{continue})? \langle Translation \rangle \langle Back \rangle$   
 $\langle Move-2 \rangle ::= \langle Front \rangle (\text{go} \mid \text{move} \mid \text{turn} \mid \text{rotate})? \langle Rotation \rangle \langle Back \rangle$   
 $\langle PickUp \rangle ::= \langle Front \rangle ((\text{take} \mid \text{pick up} \mid \text{grasp}) \langle Object \rangle) \mid \text{pick it up} \mid \text{take it} \langle Back \rangle$   
 $\langle Release \rangle ::= \langle Front \rangle ((\text{release} \mid \text{put down} \mid \text{drop}) \langle Object \rangle) \mid (\text{put } \langle Object \rangle \text{ down}) \langle Back \rangle$   
 $\langle Perception-1 \rangle ::= \text{what } (\text{can} \mid \text{do}) \text{ you } (\text{see} \mid \text{perceive})$   
 $\langle Perception-2 \rangle ::= (\text{can} \mid \text{do}) \text{ you } (\text{see} \mid \text{perceive}) (\langle Object \rangle \mid \text{something} \mid \text{anything})$   
 $\langle Confirmation \rangle ::= \text{yes } (\text{please})? \mid \text{no} \mid \text{exactly} \mid \text{ok}$   
 $\langle Grounding \rangle ::= (\text{that is} \mid \text{this is})? (\text{correct} \mid \text{incorrect} \mid \text{wrong} \mid \text{right} \mid \text{perfect} \mid \text{great} \mid \text{good})$   
 $\langle Repeat \rangle ::= \langle Front \rangle (\text{do } (\text{it} \mid \text{that}) \mid (\text{again} \mid \text{once more} \mid \text{one more time})) \mid \text{repeat that}$   
 $\langle Opening \rangle ::= (\text{hi} \mid \text{hello}) \langle Name \rangle$   
 $\langle Closing \rangle ::= (\text{bye} \mid \text{goodbye}) \langle Name \rangle?$   
 $\langle Stop \rangle ::= \text{stop } (\text{it} \mid \text{that})?$   
 $\langle Front \rangle ::= (\text{and})? (\text{could you})? (\text{now})? (\text{please})?$   
 $\langle Back \rangle ::= (\text{please})? (\text{again})?$   
 $\langle Modifier \rangle ::= (\text{just})? \text{ a } (\text{little})? \text{ bit } (\text{more})?$   
 $\langle Translation \rangle ::= \langle Modifier \rangle? (\text{forward} \mid \text{backward} \mid \text{back} \mid \text{straight } (\text{forward} \mid \text{ahead})?) \langle Modifier \rangle?$   
 $\langle Rotation \rangle ::= (\langle Modifier \rangle? (\text{to the})? (\text{left} \mid \text{right}) \langle Modifier \rangle?) \mid \text{around} \mid 180 \text{ degrees}$   
 $\langle Object \rangle ::= (\text{the} \mid \text{a}) \langle Colour \rangle? (\text{object} \mid \text{cylinder} \mid \text{food can}) (\text{at your feet})?$   
 $\langle Colour \rangle ::= \text{red} \mid \text{blue} \mid \text{green} \mid \text{yellow} \mid \text{white} \mid \text{black}$   
 $\langle Name \rangle ::= \text{robot} \mid \text{nao} \mid \text{lenny}$

Table 8.5: Speech recognition grammar (in Bachus-Naur form) employed for the experiment.

### 8.1.3 Wizard-of-Oz data collection

In order to design accurate dialogue management models for the experiment, we started by collecting a set of Wizard-of-Oz interactions for the dialogue domain. We recorded a total of 10 interactions, each with a distinct participant. All interactions were performed in English. The participants to the Wizard-of-Oz study (5 males and 5 females) were selected amongst students and employees working at the Department of Informatics. All but one participant were non-native speakers of English. The author of the present thesis served as the wizard.

The interactions were once again encoded as a sequence of pairs  $\langle \mathcal{B}_i, a_i \rangle$ , where each system turn  $i$  is represented by the selected wizard action  $a_i$  and the dialogue state  $\mathcal{B}_i$  in effect at the time the selection was made. The interactions collected here are left without further annotation. The interactions had on average 84.2 system turns. 39 % of these turns resulted in a void action (i.e. no action at all), 40 % to a physical action (complemented by a grounding describing the action), and the remaining 21 % to a verbal response such as a factual answer or a clarification request.

Contrary to the Wizard-of-Oz study presented in Section 6.3, the interactions recorded in this data collection effort are not augmented with supplementary annotation.<sup>2</sup> The recorded state includes therefore only variables that are directly observed by the system: the last and next-to-last user actions  $a_u$  and  $a_{u\text{-}prev}$  (as provided by the ASR and NLU modules), the last system action  $a_m$ , the set of objects currently perceived and carried by the robot, the last physical movement of the robot, and the *motion* variable denoting whether the robot is currently in movement.

Here is an example of data point extracted from the collected data, corresponding to the user utterance “yes” after a confirmation request, followed by the system action Do(Move(Right)):

$$\mathcal{B} = \begin{cases} u_u = \langle ("yes", p=.95), (None, p=.05) \rangle \\ a_u = \langle (\text{Confirm}, p=.95), (None, p=.05) \rangle \\ a_{u\text{-}prev} = \langle (\text{Ask}(Move(\text{Right})), p=.63), (None, p=.37) \rangle \\ a_m = \langle (\text{AskConfirm}(Move(\text{Right})), p=1) \rangle \\ u_m = \langle ("should I move right?", p=1) \rangle \\ \text{perceived} = \langle ([RedObj], p=1) \rangle \\ \text{carried} = \langle ([], p=1) \rangle \\ \text{motion} = \langle (false, p=1) \rangle \end{cases} \Rightarrow a'_m = \text{Do}(Move(\text{Right}))$$

## 8.2 Dialogue management models

In order to evaluate how the choice of a dialogue management framework can influence the interaction quality, we developed three competing dialogue management approaches for the dialogue domain:

- The first approach is a purely handcrafted dialogue manager using a finite-state automaton to process the user inputs and determine the corresponding responses.

---

<sup>2</sup>As we shall see in the next section, the relations between the user actions and the underlying intentions they express are automatically estimated as part of the learning procedure.

- The second approach is a purely statistical dialogue manager that encodes the transition and utility model of the domain based on classical, factored probabilistic models.
- Finally, the third approach relies on a hybrid modelling strategy and uses the formalism of probabilistic rules to structure the domain models.

We detail below how the exact design of each approach.

### 8.2.1 Approach 1: handcrafted model

The first approach relies on a traditional finite state automaton to determine the current conversational situation and its corresponding system action. The finite state automaton is triggered upon each new user dialogue act  $a_u$ . Attached to each edge is a logical condition that determines when the edge can be traversed. The conditions are defined on the basis of the current dialogue act  $a_u$ , previous dialogue act  $a_{u-prev}$ , and contextual variables *perceived* and *carried*.<sup>3</sup>

#### Finite state automaton

The automaton designed for the dialogue domain operates on the basis of three specified threshold values  $T_1$ ,  $T_2$  and  $T_3$  (whose estimation is described shortly):

1. If the incoming dialogue act has an hypothesis whose probability is higher than a given threshold  $T_3$ , the system action associated to the user request is directly selected. For instance, if the new user act contains a hypothesis  $a_u = \text{Ask}(\text{Move}(Left))$  with probability  $p > T_3$ , the action  $a_m = \text{Do}(\text{Move}(Left))$  will be executed.
2. If the probability of the top hypothesis for  $a_u$  lies between  $T_2$  and  $T_3$  and corresponds to a request for a physical movement, the system will ask the user to confirm her/his intention. The system response to a user act  $a_u = \text{Ask}(\text{Move}(Left))$  with probability  $T_2 < p < T_3$  will hence be set to  $a_m = \text{AskConfirm}(\text{Move}(Left))$
3. If the probability of the top hypothesis lies between  $T_1$  and  $T_2$ , the system will ask the user to repeat her/his utterance by triggering the action  $a_m = \text{AskRepeat}$ .
4. Finally, if no hypothesis reaches the minimal threshold  $T_1$ , the user action is assumed to correspond to a spurious recognition result, and is simply ignored.

A (slightly simplified) graphical depiction of the corresponding finite state automaton is provided in Figure 8.2. The finite state contains 16 distinct states revolving around the key state  $s_1$ . Depending on the probability of the top hypothesis in the user dialogue act  $a_u$ , the automaton will either directly transit from the state  $s_1$  to the states  $s_4 \cdots s_{15}$  or be redirected to state  $s_2$  or  $s_3$  for clarification. After each user action in the states  $s_4 \cdots s_{15}$ , the current state is re-initialised to state  $s_1$ , through empty transitions.

---

<sup>3</sup>It should be noted that the finite state automaton directly operates at the level of user dialogue acts  $a_u$  without trying to evaluate the underlying user intention  $i_u$ , since the user intention is a hidden variable that is intrinsically difficult to account without resorting to probabilistic methods.

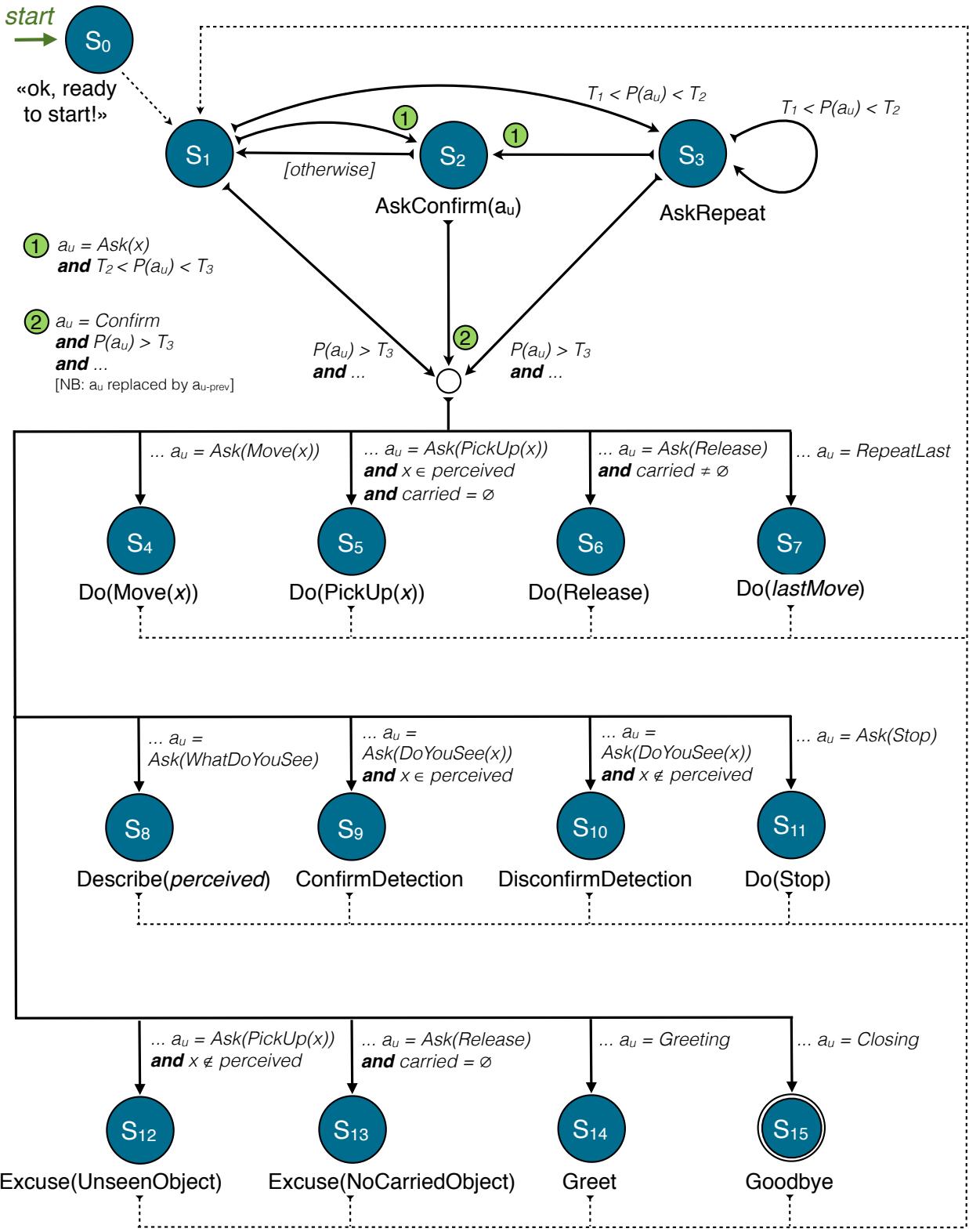


Figure 8.2: Finite state automata designed for the domain (slightly simplified for clarity's sake). The dashed arrows denote empty transitions. For presentation purposes, the conditions applied to the edges going from  $s_1, s_2, s_3$  to  $s_4 \dots s_{15}$  are decomposed in two parts: the first part from the incoming node to the small white circle, and the second part from the circle to the outgoing node.  $T_1, T_2$  and  $T_3$  are threshold values determined empirically.

## Probability thresholds

The threshold values  $T_1$ ,  $T_2$  and  $T_3$  are determined empirically on the basis of the actual probability values generated by the speech recogniser and NLU module during the Wizard-of-Oz study. This estimation was achieved in two steps. The first step consisted in extracting all probability values associated with the most likely hypothesis of the user dialogue act  $a_u$  in the recorded Wizard-of-Oz interactions and deriving a probability density from these values through kernel density estimation. The resulting density function is shown in Figure ???. We then divided the distribution over probability values for the top user act hypothesis into four non-overlapping regions:

- The first region corresponds to user actions that most likely arise from spurious recognition and should hence be ignored. Based on a detailed inspection of the wizard actions in such cases, we mapped this region to the lowest quintile of the distribution (that is, all values with a cumulative density between 0 and 0.2).
- The second region corresponds to user actions that are more likely to reflect real user inputs, but do not have sufficient confidence to be directly executed. This region was mapped to the second quintile of the distribution (values with a cumulative density between 0.2 and 0.4).
- The third region corresponds to user actions that are relatively confident, but should nevertheless be confirmed once before execution. This region was constrained to the third quintile of the distribution (values with a cumulative density between 0.4 and 0.6).
- Finally, the fourth and final region corresponds to user actions that are sufficiently confident to be directly executed without user confirmation. This region defines the top 40 % of the probability mass, with cumulative densities between 0.6 and 1.0.

The values that delineate these four regions are calculated using linear interpolation. In practice, the estimated thresholds for the density function shown in Figure 8.3 were derived to be  $T_1 = 0.49$ ,  $T_2 = 0.61$  and  $T_3 = 0.73$ . These values reflect the three thresholds employed in the finite state automaton.

### 8.2.2 Approach 2: factored statistical model

The second approach developed for this experiment consists of a traditional statistical model whose parameters are estimated from the Wizard-of-Oz data set collected for the experiment. We assume that both the transition model and utility model of the domain are initially unknown.

Both this approach and Approach 3 assume that the domain can be captured with a planning horizon limited to the present time step. In other words, the approach does not rely on online forward planning (which has proven to be practically difficult to reconcile with real-time constraints) and directly selects the action yielding the highest utility in the current dialogue state.<sup>4</sup>

## Domain modelling

Figure 8.4 shows the factored model employed to structure the dialogue domain of the experiment. As evidenced in the figure, the domain models are factored in four major distributions (the three first ones being part of the transition model, while the fourth one is a utility model):

---

<sup>4</sup>In other words, the utility model represents what is called a  $Q$ -value model in the reinforcement learning literature.

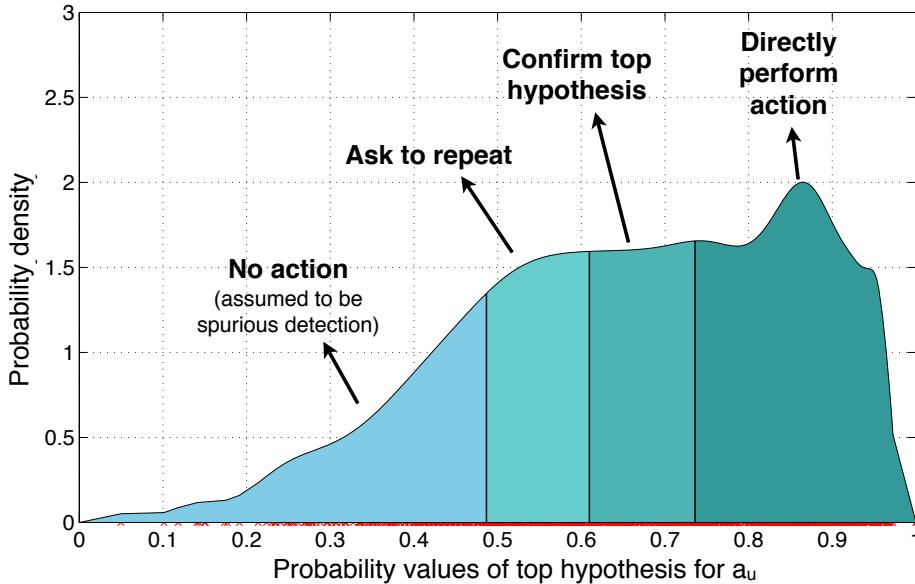


Figure 8.3: Probability density function for the probability values of the top hypothesis specified in the user dialogue act  $a_u$ , divided into four non-overlapping regions that respectively correspond to the first, second, third, and fourth+fifth quintiles of the distribution. Red crosses on the X axis represent the actual probability values used to construct the kernel density function.

- The task completion model  $P(\text{completed-task} | i_u, a_m)$  describes the probability that the current intention  $i_u$  is fulfilled by the last system action  $a_m$ . The task completion model  $P(\text{completed-task} | i_u, a_m)$  is essentially a deterministic distribution that marks the task as completed when the executed action  $a_m$  fulfils the user intention  $i_u$ , and as not completed in other cases.
- The user goal model  $P(i'_u | i_u, \text{completed-task}, \text{perceived}', \text{carried}')$  encodes the likelihood of a new user intention  $i'_u$  given the current intention  $i_u$ , task completion *completed-task*, as well as the two contextual variables *perceived* and *carried*.
- The user action model  $P(a'_u | i'_u, a_m)$  describes the probability of the next user dialogue act  $a'_u$  given the user intention  $i'_u$  and the last system action  $a_m$ .
- Finally, the utility model  $U(a'_m, i'_u, a'_u, \text{carried}', \text{perceived}', \text{motion}')$  describes the utility of a given system action  $a'_m$  given the user intention  $i'_u$ , last user dialogue act  $a'_u$ , robot motion status *motion'* and contextual variables *carried* and *perceived*.

### Dimensionality reduction

The four factored models remain however too large to be directly estimated from the limited amounts of data made available from the Wizard-of-Oz interactions. We therefore introduced a range of additional constraints and simplifying assumptions that aim to further reduce the number of parameters associated with the model.

The first simplifying assumption pertains to the user goal model  $P(i'_u | i_u, \text{completed-task}, \text{perceived}, \text{carried})$ , and rests on the idea that the user intention remains more or less constant

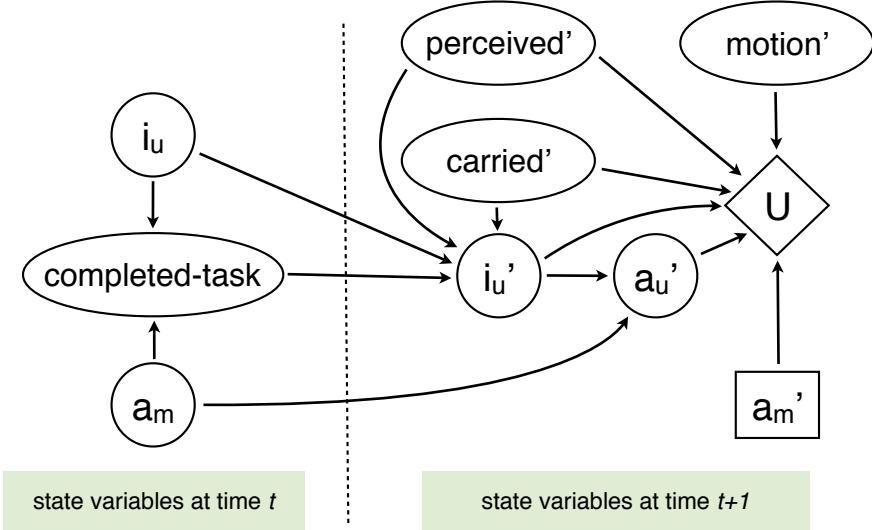


Figure 8.4: Factored transition and utility models for the dialogue domain.

(albeit with a small probability of change) until the task is marked as completed. The values of the two contextual variables *perceived* and *carried* variables are furthermore grouped into two partitions: empty set or non-empty set. These two simplifications reduce the number of parameters required for the model to four Dirichlet distributions, each with 18 dimensions (which is the number of possible user intentions).

We also compressed the size of the user action model  $P(a'_u | i'_u, a_m)$  by exploiting the fact that only a fraction of user dialogue acts  $a'_u$  are locally relevant for a given user intention  $i'_u$  and last system action  $a_m$ . A manually specified function  $\text{relevant} : \text{Val}(i_u) \times \text{Val}(a_m) \rightarrow 2^{\text{Val}(a_u)}$  is used to derive the set of relevant user actions given a particular pair of user intention and system action. For instance, the relevant dialogue acts for the user intention Move(Left, Short) are Ask(Move(Left, Short)), RepeatLast, Confirm, Disconfirm, Stop and Grounding. The resulting set of parameters associated with the user action model contains 133 Dirichlet distributions, each with 6 dimensions (the number of relevant actions allowed per intention).

The third and final heuristic is to reduce the number of relevant system actions in the utility distribution  $U(a'_m, i'_u, a'_u, \text{carried}, \text{perceived}, \text{motion}')$ . Similarly to the user action model, one can exploit the fact that only a fraction of system actions are relevant in a given state, and that the non-relevant ones have a constant negative utility. In order to express this constraint, we first define a function  $\text{best} : \text{Val}(i_u) \times \text{Val}(\text{perceived}) \times \text{carried} \rightarrow \text{Val}(a_m)$  that maps each user intention and context to the “best” action that can be executed at that state if we were to assume no uncertainty. This best action is identical to the action selected by the finite state automaton in Figure 8.2. In addition to this best action, we also allow the actions AskConfirm( $i'_u$ ), AskRepeat and *Do(Stop)* to be executed at any state. The utility distribution is then expressed as:

$$U(a'_m, i'_u, a'_u, \text{carried}, \text{perceived}, \text{motion}') =$$

$$\left\{ \begin{array}{ll} \theta_{\text{best}(i'_u, \text{carried}, \text{perceived})} & \text{if } a_m = \text{best}(i'_u, \text{carried}, \text{perceived}) \\ \theta_{\text{askConfirm}(i'_u, \text{carried}, \text{perceived})} & \text{if } a_m = \text{Confirm}(i'_u) \\ \theta_{\text{askRepeat}(a_u, \text{motion}')} & \text{if } a_m = \text{AskRepeat} \\ \theta_{\text{stop}(a_u, \text{motion}')} & \text{if } a_m = \text{Stop} \\ -10 & \text{otherwise} \end{array} \right. \quad (8.1)$$

The factorisation of the utility model in Equation (8.1) results in 296 parameters. In total, the numbers of parameters associated with all factored models in this approach (including both univariate and multivariate parameters) equals to 433.

### Parameter estimation

The parameter estimation follows the Bayesian learning procedure presented in Chapter 5. The learning algorithm operates by cycling through the collection of turns recorded in the Wizard-of-Oz experiments, and updating the parameter distributions after each turn using Bayesian inference. The wizard is assumed to act rationally in most cases and select the action that yields the highest utility given the current dialogue state.

It should be noted that, in contrast to the experiment presented in Section 5.3 (which concentrated on the estimation of a single model), the parameters to estimate in this setting include both the transition model and the utility model. The learning algorithm is thus an instance of a joint optimisation problem, as the system must simultaneously optimise both models in order to find the parameters that provide the best fit for the Wizard-of-Oz dataset as a whole.

The parameters of categorical distributions are all encoded by Dirichlet distributions initialised with weakly informative priors, while the utility parameters are encoded by uniform distributions with a support range of  $[-10, 30]$ . Section 8.2.4 presents the exact learning curve followed during the estimation process.

### 8.2.3 Approach 3: rule-structured model

The final dialogue management approach developed in this experiment is couched in the formalism of probabilistic rules presented in this thesis.

#### Domain modelling

The general structure of the domain remains similar to the one shown in Figure 8.4. As in the second approach, the models are decomposed into a task completion model, a user goal model, a user action model, and a utility model. Action selection is similarly formalised as a search for the highest-utility action in the current state (assuming a planning horizon limited to the current time step). However, in contrast to Approach 2, the internal structure of the domain models is here encoded through probabilistic rules instead of standard categorical distributions:

- The task completion model  $P(\text{completed-task} | i_u, a_m)$  is encoded by one single deterministic rule that define when the current intention  $i_u$  is fulfilled by the system action  $a_m$ .
- The user goal model  $P(i'_u | i_u, \text{completed-task}, \text{perceived}', \text{carried}')$  consists of a collection of seven rules that define the probability of the new user intention  $i'_u$  given the dialogue

context. One rule defines the prior probability of the user intentions  $\text{Move}(x)$ , four rules specify the prior probability of  $\text{PickUp}(x)$ ,  $\text{WhatDoYouSee}$  and  $\text{DoYouSee}(x)$  (depending on the value of the *perceived* variable), one rule specifies the prior probability of  $\text{Release}$  (depending on the value of the *carried* variable), and one final rule specifies the probability of the conventional Greeting and Closing intentions (depending on the last system move).

- The user action model  $P(a'_u | i'_u, a_m)$  is formalised with two rules that maps user intentions to their potential verbal realisations in terms of user dialogue acts. The first rule expresses the likelihood of various user dialogue acts depending on the user intention and last system action (in particular, whether the system uttered a clarification or confirmation request). The second rule expresses the likelihood of the Ask(Stop) and Disconfirm actions during the execution of a physical movement.
- Finally, the utility model  $U(a'_m, i'_u, a'_u, \text{carried}', \text{perceived}', \text{motion}')$  is encoded with a total of eleven utility rules. The model essentially contains one rule for each family of possible system actions. There is therefore one rule to define the relative utility of movement actions, one rule for grasping actions, one rule for the release action, two rules for the  $\text{Describe}(x)$ ,  $\text{ConfirmDetection}$  and  $\text{DisconfirmDetection}$  actions, one rule to handle the Greet and Goodbye actions, three rules for the clarification and confirmation requests, and one rule to stop the current movement. The last rule fixes an initial, negative utility for all actions to ensure that non-relevant actions have a utility lower than zero.

The probability and utility models described by these rules include a total of 28 parameters, amongst which 15 probability parameters and 13 utility parameters. The complete specification of the domain is provided in Appendix B (Section B.3).

### Parameter estimation

The rule parameters were optimised on the basis of the collected Wizard-of-Oz data following the same procedure as the one followed for the second approach. The parameters of probability rules are all encoded by Dirichlet distributions (of varying dimensions, from 2 to 10 depending on the rule) initialised with weakly informative priors, while the parameters of utility rules are encoded by uniform distributions on the support range  $[-10, 30]$ .

#### 8.2.4 Learning curves

Figure 8.5 presents the learning curves that characterise the three dialogue management approaches presented in the previous pages. The Wizard-of-Oz data set described in Section 8.1.3 was first divided into a training set of 9 interactions (summing up to 770 system turns) and a test set with one single interaction containing a total of 71 system turns. Based on this division, we measured the degree of agreement between the action selected by the system (on the basis of its model parameters) and the one selected by the wizard. This measurement is repeated at regular intervals during the parameter estimation process. It should be stressed once more that system actions that do not correspond to the ones selected by the wizard are not necessarily wrong or inappropriate, as they may reflect different but perfectly legitimate conversational strategies. The degree of agreement

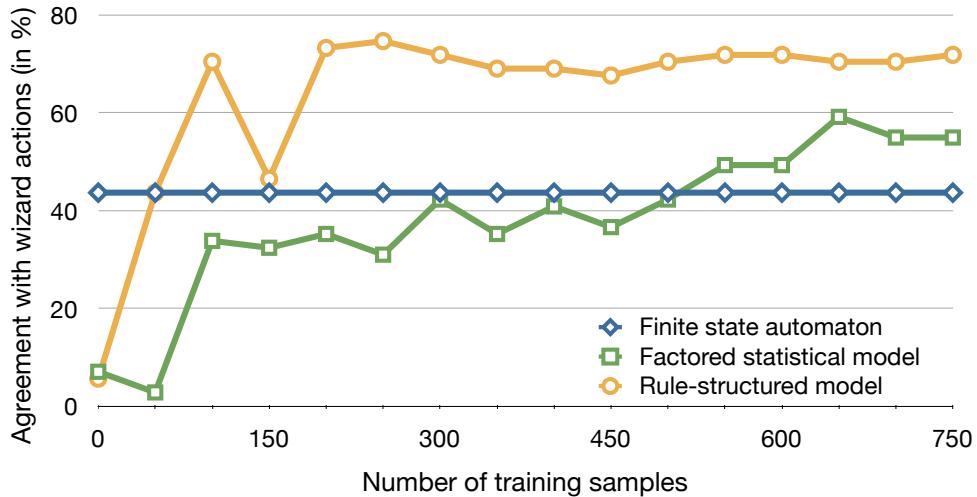


Figure 8.5: Learning curve for the three dialogue management approaches on the Wizard-of-Oz data set. The figure shows how the agreement between the best system action and the actual wizard action evolves as a function of the number of processed training samples. The agreement is calculated on a separate test set of 71 system turns.

between the model and the wizard examples is however a good indication of the model ability to capture the dynamics of the interaction and the trade-offs of the action selection process.

The learning curves in Figure 8.5 illustrate the evolution of the model accuracy as the function of the number of training samples processed by the learning algorithm. As the finite-state approach is entirely handcrafted and does not include any parameters to optimise, its accuracy remains constant at all time steps. We can notice that both statistical approaches (Approach 2 and 3) do improve their accuracy as they process more training samples, but do so at different learning speeds. While the rule-structured model is able to converge to a high-quality dialogue policy after observing only a fraction of the training data, the traditional, factored statistical model converges at a much slower rate due to its larger set of parameters and weaker generalisation capacity.

The final agreement results obtained after processing the complete training set are shown in Table 8.6. As one can observe, the rule-structured approach is the one that is best able to imitate the conversational behaviour of the wizard.

Type of model	Agreement (in %)
Finite state automaton	43.66
Factored statistical model	54.93
Rule-structured model	<b>71.83</b>

Table 8.6: Agreement results on the test test for the three approaches.

## 8.3 User trials

In order to assess the relative performance of the three dialogue management approaches presented in the previous section, we conducted a total of  $3 \times 36$  user trials.

### 8.3.1 Experimental setup

#### Participants

age, gender, provenance reward for participation

#### Procedure

shuffling the order of the approaches questionnaire right after the completion of each approach  
system responsible could intervene after systematic system failures

#### User expectations

show question and answers

### 8.3.2 Metrics

#### Objective metrics

#### Subjective metrics

various measures (number of turns, duration, number of disconfirm)

direct assessment of user satisfaction

no tasks completion since the notion of failure is not well defined

### 8.3.3 Results

to say: user turns based on ASR, duration cuts the system crashes, no repetitions requests in structured model

users feedback after experiment corroborate our claims

describe the limitations of each system (e.g. no accumulation of evidence or context-based reasoning in FSA, no good policy for states that have not been observed in the Woz interactions for unstructured model)

ANOVA, since we would have two baselines and our approach? (see e.g. Passonneau's article)

how ANOVA is calculated, and post-hoc tests

### 8.3.4 Analysis

also evaluate the difference between the first approach tested and the other ones?

## 8.4 Conclusion

Metrics	Approaches			<i>p</i> -values ANOVA	<i>p</i> -values 1 <sup>st</sup> vs. 2 <sup>nd</sup> ( <i>t</i> -test)
	Finite- state automata	Factored statistical model	Rule- structured model		
<b>Objective metrics:</b>					
1. Average number of repetition requests per dialogue	18.68	12.24	<b>0*</b>	$9 \times 10^{-19}$	$1 \times 10^{-16}$
2. Average number of confirmation requests per dialogue	9.16	10.32	<b>5.78*</b>	$1.7 \times 10^{-4}$	0.001
3. Average number of repeated instructions per dialogue	3.73	7.97	<b>2.78</b>	$3.8 \times 10^{-9}$	0.18
4. Average number of “Disconfirm” acts per dialogue	<b>2.16</b>	2.59	2.59	0.65	0.33
5. Average number of physical movements per dialogue	<b>26.68</b>	29.89	27.08	0.13	0.80
6. Average number of (user and system) turns between movements	3.63	3.1	<b>2.54*</b>	$4 \times 10^{-4}$	$2.2 \times 10^{-4}$
7. Average number of user turns per dialogue	78.95	77.3	<b>69.14</b>	0.17	0.11
8. Average number of system turns per dialogue	57.27	54.59	<b>35.11*</b>	$4.4 \times 10^{-9}$	$5.6 \times 10^{-8}$
9. Average duration of each dialogue (in minutes)	618	713	<b>5'24*</b>	$1.4 \times 10^{-4}$	0.02
<b>Subjective metrics:</b>					
“Did you feel that ...					
10. ... the robot correctly understood what you said?”	3.32	2.92	<b>3.68*</b>	$1.3 \times 10^{-4}$	0.03
11. ... the robot reacted appropriately to your instructions?”	3.70	3.32	<b>3.86</b>	$7.6 \times 10^{-3}$	0.23
12. ... the robot asked you to repeat or confirm your instructions ...”	2.16	2.19	<b>3.3*</b>	$1.7 \times 10^{-9}$	$4.7 \times 10^{-7}$
13. ... the robot sometimes ignored when you were speaking?”	3.24	2.76	<b>3.43</b>	$6.7 \times 10^{-3}$	0.21
14. ... the robot sometimes thought you were talking when you were not?”	3.43	3.14	<b>4.41*</b>	$3.4 \times 10^{-6}$	$4.7 \times 10^{-5}$
15. ... the interaction flowed in a pleasant and natural manner?”	2.97	2.46	<b>3.32*</b>	$8.6 \times 10^{-4}$	0.03

Table 8.7: Empirical results obtained for the user evaluation with a total of 37 participants, based on a set of 15 metrics (9 objective and 6 subjective). The \* symbol indicates results that outperform the two other approaches with a level of statistical significance  $\alpha = 0.05$ .

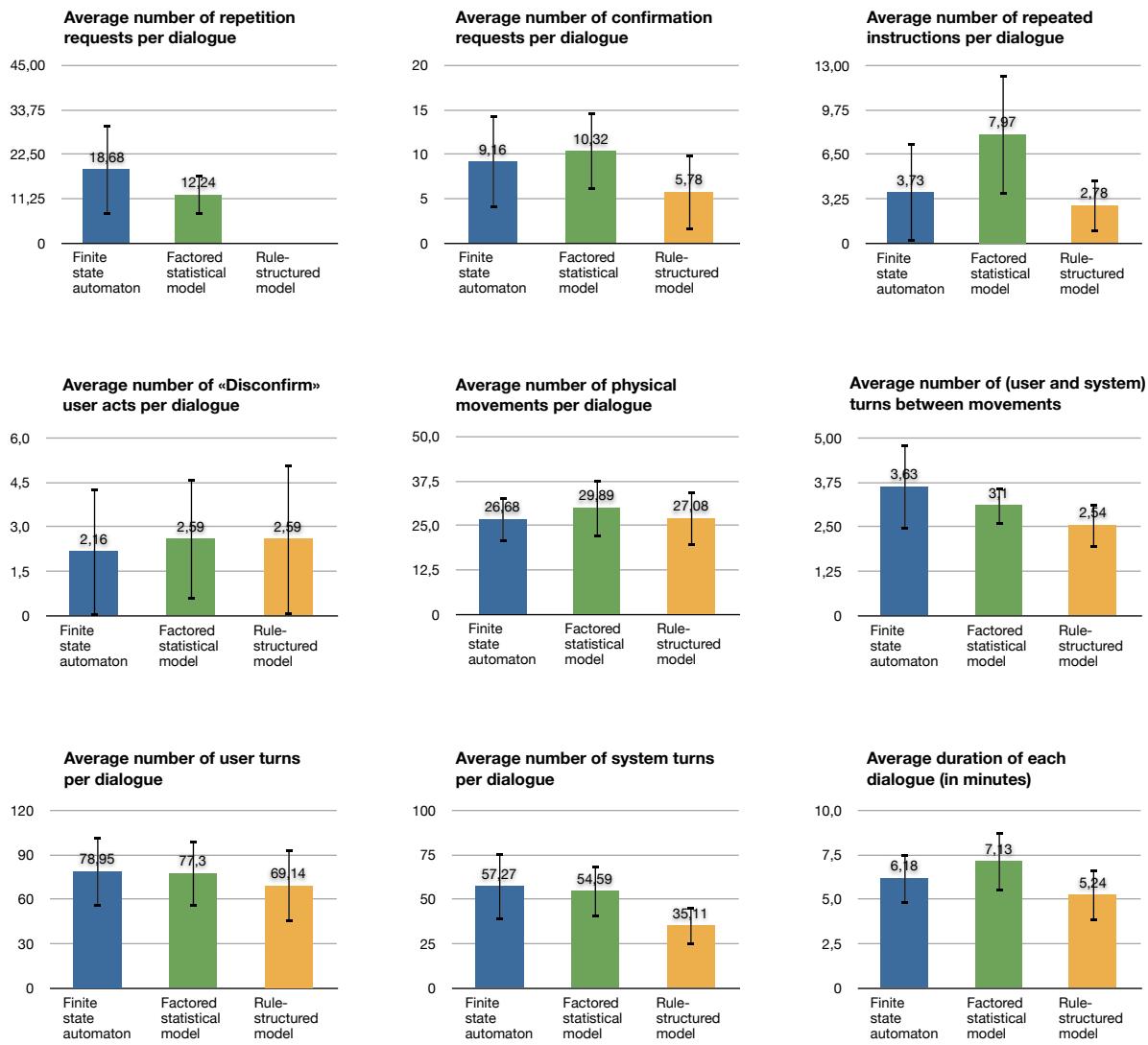


Figure 8.6: Objective metrics extracted from the collected dialogues.

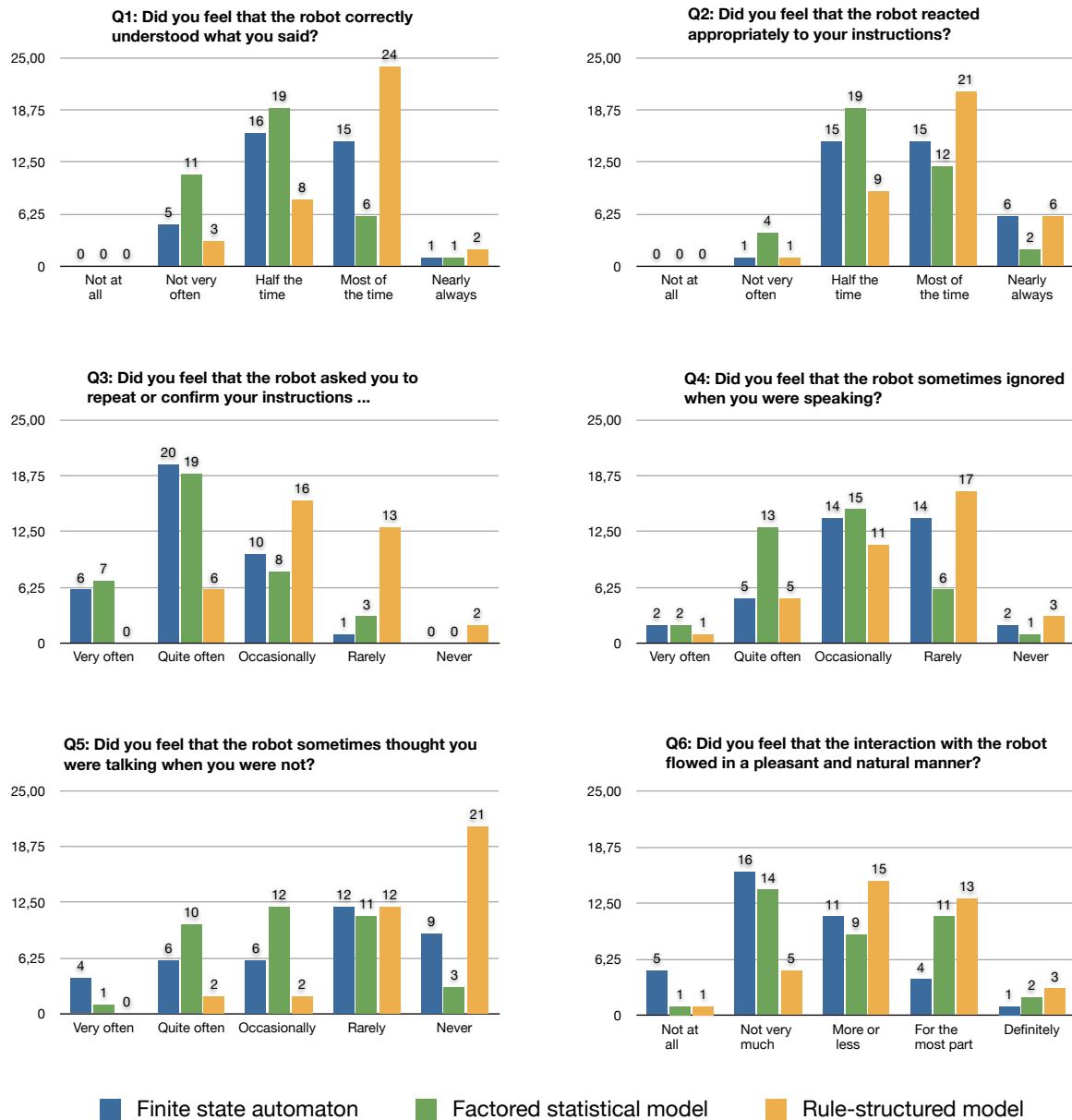


Figure 8.7: Survey responses obtained from the participants in the user trial.

# Chapter 9

## Concluding remarks

### 9.1 Summary of contributions

**A new, hybrid approach to dialogue management**

be able to capture richer conversational context, and better account for the cooperative nature of dialogue (e.g. Jokinen's argument).

**Estimation of rule parameters from Wizard-of-Oz data**

**Estimation of rule parameters via Bayesian reinforcement learning**

**Development of the openDial toolkit**

### 9.2 Future work

Formally characterise the expressivity of the rules and extend them to handle Ginzburg style update rules?

**Reinforcement learning with live interactions**

Try to learn a policy in a fully online fashion with real users, without simulator

**Combination of imitation and reinforcement learning**

do online reinforcement learning with real users and combine imitation+reinforcement learning. cognitive basis for this (toddlers). in line with Henderson's approach and others (Verena, Jason)

**Joint optimisation of dialogue models**

joint optimisation of models

**Incrementality and turn-taking**

**Scaling up to more complex domains**

more complex domains, with more variables and more complex dynamics

find ways to optimise performance of openDial



# Appendix A

## Relevant probability distributions

This appendix details the most important probability distributions employed in this thesis. For each distribution, we provide its parameters, probability density or mass function (depending on whether the distribution has a discrete or continuous domain), mean and variance.

### Continuous uniform distribution

**Parameters:** Continuous uniform distributions are described by two parameters  $a$  and  $b$  that define the range  $[a, b]$  upon which the distribution is defined.

**Density function:** The density remains constant within  $[a, b]$  and is 0 outside this range:

$$P(x ; a, b) = \begin{cases} \frac{1}{b-a} & \text{for } x \in [a, b] \\ 0 & \text{otherwise} \end{cases} \quad (\text{A.1})$$

**Mean:** The mean of a continuous uniform distribution is simply equal to  $\frac{a+b}{2}$

**Variance:** The distribution variance is  $\frac{(b-a)^2}{12}$

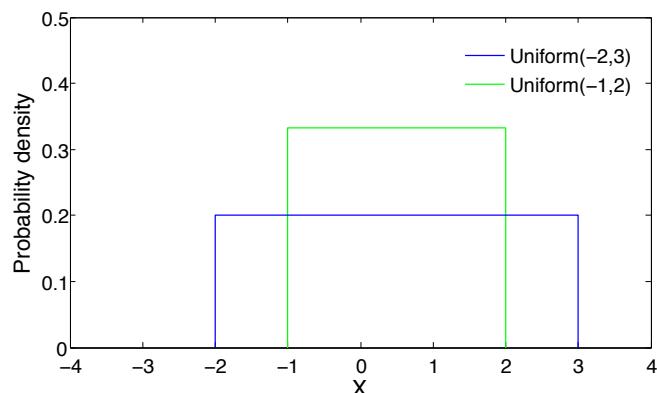


Figure A.1: Probability density functions for two continuous uniform distributions.

## Discrete uniform distribution

**Parameters:** Discrete uniform distributions are also described by two parameters  $a$  and  $b$  – but are here constrained to integer values. The number  $n$  of values in the distribution is  $b - a + 1$ .

**Probability mass function:** The probability mass function is defined as:

$$P(X = i; a, b) = \begin{cases} \frac{1}{n} & \text{for } i \in \mathcal{Z} \text{ and } i \in [a, b] \\ 0 & \text{otherwise} \end{cases} \quad (\text{A.2})$$

**Mean:** The mean of a discrete uniform distribution is also  $\frac{a + b}{2}$

**Variance:** The distribution variance is  $\frac{(n^2 - 1)}{12}$

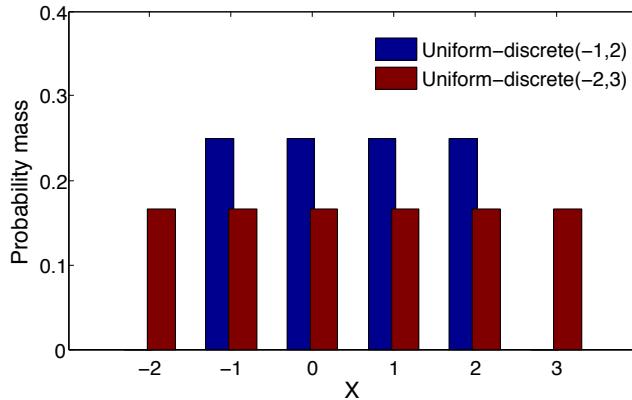


Figure A.2: Probability mass functions for two discrete uniform distributions.

## Categorical distribution

A categorical distribution describe the result of a random event that can take on one of  $K$  possible (and mutually exclusive) outcomes.

**Parameters:** A categorical distributions on  $K$  outcomes is defined by  $K$  parameters  $p_1, \dots, p_K$  representing the probabilities of each outcome.

**Probability mass function:** The probability mass function is simply defined as:

$$P(X = i; p_1, \dots, p_K) = p_i \quad (\text{A.3})$$

The mean and variance of categorical distributions are only defined when the outcomes have a numerical range. The mean is in such case equal to  $\mu = \sum_{i=1}^K i p_i$ , while the variance corresponds to  $\sigma^2 = \sum_{i=1}^K (p_i i^2) - \mu^2$ .

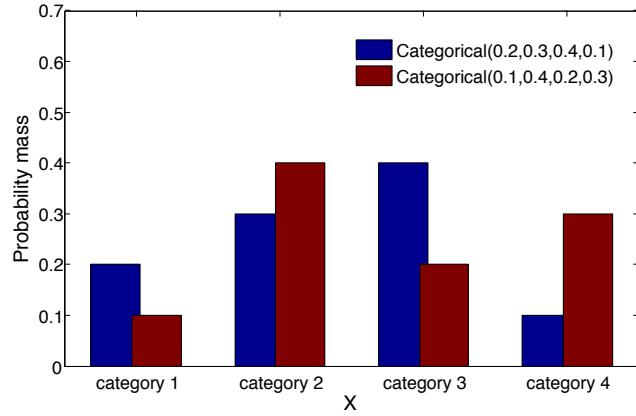


Figure A.3: Probability mass functions for two categorical distributions.

## Multinomial distribution

Given the repetition of  $n$  independent trials where each trial leads to one of  $K$  possible outcomes (described by a categorical distribution), the multinomial distribution gives the probability of a particular combination of numbers of outcomes for the various categories.

**Parameters:** Multinomial distributions are parametrised with  $K$  probability values  $p_1, \dots, p_K$  expressing the likelihood of each outcome, and an integer  $n$  denoting the number of trials.

**Probability mass function:** For  $n$  trials, the probability of having  $x_1$  outcomes for category 1,  $x_2$  outcomes for category 2, ... and  $x_K$  outcomes for category  $K$  is:

$$P(X_1 = x_1, \dots, X_K = x_K) = \begin{cases} \frac{n!}{x_1! \cdots x_K!} p_1^{x_1} \cdots p_K^{x_K}, & \text{when } \sum_{i=1}^K x_i = n \\ 0 & \text{otherwise,} \end{cases} \quad (\text{A.4})$$

**Mean:** The expected number of outcomes for a category  $i$  is  $np_i$ .

**Variance:** The variance on the number of outcomes for a category  $i$  is  $np_i(1 - p_i)$ .

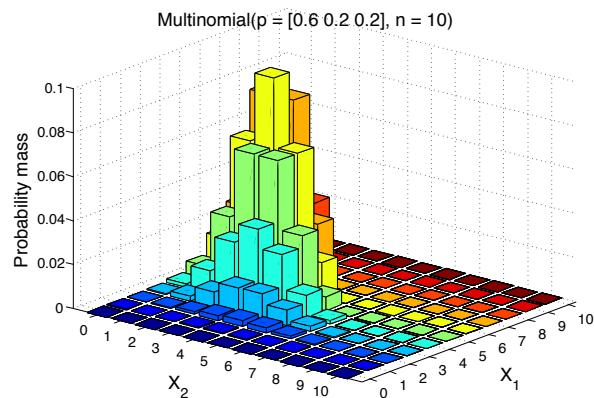


Figure A.4: Probability mass function for a multinomial distribution.

The binomial distribution is a special case of multinomial with only two categories. The binomial distribution expresses the number of successes associated with the repetition of  $n$  Bernoulli trials with a specific probability of success.

## Normal distribution

The normal distribution (also called the Gaussian distribution or the bell curve) is the most well known probability distribution in the continuous domain. The central limit theorem states that, under mild conditions, the mean of many random variables drawn independently from the same (arbitrary) distribution is distributed according to a normal distribution in the large sample limit.

**Parameters:** A normal distribution is encoded by two parameters: the mean  $\mu$  and variance  $\sigma^2$ .

**Probability density function:** The density function of a normal distribution is defined as:

$$P(x ; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left\{ -\frac{(x - \mu)^2}{2\sigma^2} \right\} \quad (\text{A.5})$$

**Mean:** The mean of the normal distribution corresponds to  $\mu$ .

**Variance:** The variance of the distribution is  $\sigma^2$ .

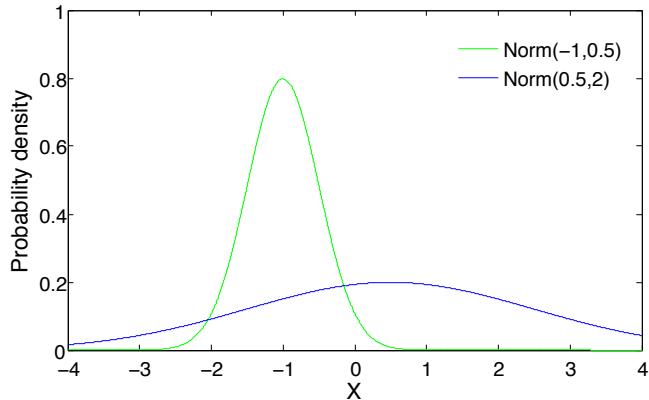


Figure A.5: Probability density functions for two normal distributions.

Multivariate normal distributions can be used to generalise one-dimensional normal distributions to higher dimensions. Multivariate normal distributions of dimensions  $K$  are parametrised by a  $K$ -dimensional mean vector  $\mu$  and a  $K \times K$  - dimensional covariance matrix  $\Sigma$ .

## Dirichlet distribution

A Dirichlet is a continuous, multivariate probability distribution that is often used to describe the parameters of categorical or multinomial distributions. The Dirichlet distribution is indeed the conjugate prior of these distributions.

**Parameters:** A Dirichlet distribution of dimension  $K$  is encoded by  $K$  parameters  $\boldsymbol{\alpha} = \alpha_1, \dots, \alpha_K$ .

**Probability density function:** The density function of a Dirichlet distribution is defined as:

$$P(x_1, \dots, x_K; \boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{i=1}^K x_i^{\alpha_i - 1} \quad \text{with } B(\boldsymbol{\alpha}) = \frac{\prod_{i=1}^K \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^K \alpha_i)} \quad (\text{A.6})$$

in which  $B(\boldsymbol{\alpha})$  is a normalisation factor and  $\Gamma$  is the gamma function  $\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt$ .

**Mean:** The mean of the Dirichlet distribution is  $\frac{\alpha_i}{\sum_k \alpha_k}$ .

**Variance:** The variance of the distribution is  $\frac{\alpha_i(\alpha_0 - \alpha_i)}{\alpha_0^2(\alpha_0 + 1)}$  with  $\alpha_0 = \sum_{i=1}^K \alpha_i$ .

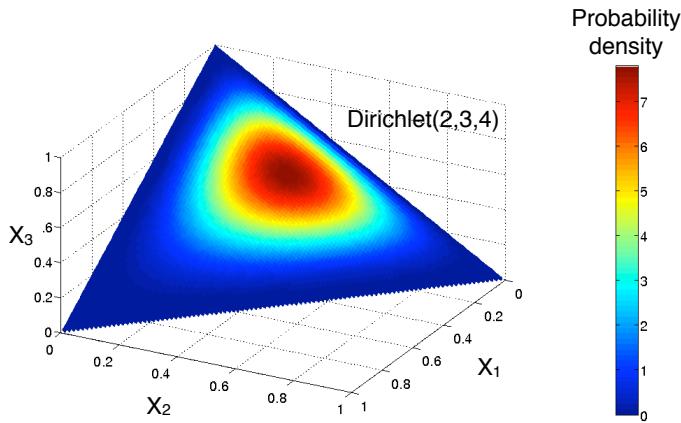


Figure A.6: Probability density function for a three-dimensional Dirichlet distribution.

## Kernel distribution

Distributions can also be defined in a non-parametric manner using kernel density estimation (KDE). Such distributions are constructed on the basis of a set of sample points  $x_1, \dots, x_n$ .

**Parameters:** A kernel distribution is fully defined by its sample points, the kernel function  $K(\cdot)$ , and the smoothing bandwidth  $h$ . Multiple kernels are possible but a common choice is the Gaussian distribution.

**Probability density function:** The kernel density estimator for the points  $x_1, \dots, x_n$  is:

$$P(x) = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x - x_i}{h}\right) \quad (\text{A.7})$$

**Mean:** The mean of kernel density estimate is the average of the sample points:  $\frac{\sum_{i=1}^n x_i}{n}$ .

**Variance:** The variance of the estimate is  $\sigma^2 + h^2 \kappa$ , where  $\sigma^2$  is the variance of the sample points and  $\kappa$  corresponds to the variance of the kernel.

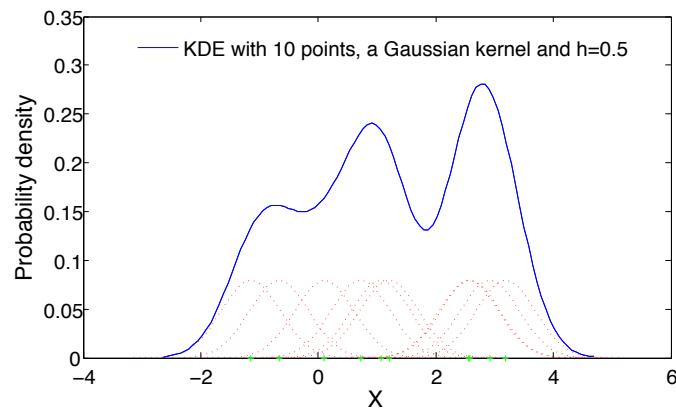


Figure A.7: Probability density function for a kernel density estimate with ten points. The Gaussian distributions making up the estimate are shown with red dashed lines.

## Geometric distribution

Add geometric distribution here

# Appendix B

## Domain specifications

to be formatted and proofread

### B.1 Experiments in Section 5.3

The dialogue management part of the domain is composed of a total of fifteen utility rules. The input variables of these rules are the user dialogue act  $a_u$ , the last system act  $a_m$ , the last physical movement  $lastMove$ , the recorded list of movements  $sequence$ , and a variable  $silence$  expressing the amount of time elapsed since the last (user or system) action.

The variable  $a_u$  is encoded in this experiment as a list of one or more dialogue act. This encoding facilitates the processing of utterances containing more than one dialogue act.

```
r1 :       $\forall x :$ 
            if ( $silence > 3 \wedge a_m = Demonstrate(x)$ ) then
                 $\begin{cases} U(a_m = AskConfirmation) = \theta_{confirmation3} \end{cases}$ 
            else if ( $silence > 2 \wedge a_m = Demonstrate(x)$ ) then
                 $\begin{cases} U(a_m = AskConfirmation) = \theta_{confirmation2} \end{cases}$ 
            else if ( $silence > 1 \wedge a_m = Demonstrate(x)$ ) then
                 $\begin{cases} U(a_m = AskConfirmation) = \theta_{confirmation1} \end{cases}$ 

r2 :      if ( $Confirm \in a_u \wedge lastMove \neq None$ ) then
             $\begin{cases} U(a_m = Register) = \theta_{registerExplicit} \end{cases}$ 
        else
             $\begin{cases} U(a_m = Register) = \theta_{registerExplicitNeg} \end{cases}$ 

r3 :      if ( $Disconfirm \in a_u \wedge None \neq lastMove$ ) then
             $\begin{cases} U(a_m = Undo) = \theta_{undo} \end{cases}$ 
        else
             $\begin{cases} U(a_m = Undo) = \theta_{undoNeg} \end{cases}$ 

r4 :      if ( $a_u \neq None \wedge lastMove \notin a_u \wedge Confirm \notin a_u$ 
             $\wedge Disconfirm \notin a_u \wedge lastMove \neq None$ ) then
             $\begin{cases} U(a_m = Register) = \theta_{registerImplicit} \end{cases}$ 
        else
             $\begin{cases} U(a_m = Register) = \theta_{registerImplicitNeg} \end{cases}$ 
```

$r_5 :$   $\forall x, y, z :$   
**if** ( $a_u = x \wedge lastMove \notin a_u \wedge (MoveArm(y, z) \in a_u$   
 $\vee MoveHead(y) \in a_u \vee Kneel \in a_u \vee MoveFoot(x, y) \in a_u$   
 $\vee StandUp \in a_u \vee SitDown \in a_u \vee Turn(y) \in a_u))$  **then**  
 $\quad \left\{ U(a_m = Demonstrate(x)) = \theta_{Demonstrate} \right.$   
**else**  
 $\quad \left\{ U(a_m = Demonstrate(x)) = \theta_{DemonstrateNeg} \right.$

$r_6 :$  **if** ( $a_m \neq AskRepeat$ ) **then**  
 $\quad \left\{ U(a_m = AskRepeat) = \theta_{repeatFirst} \right.$   
**else**  
 $\quad \left\{ U(a_m = AskRepeat) = \theta_{repeatSecond} \right.$

$r_7 :$  **if** ( $Confirm \in a_u \wedge |a_u| = 1$ ) **then**  
 $\quad \left\{ U(a_m = Ack) = \theta_{ack} \right.$   
**else**  
 $\quad \left\{ U(a_m = Ack) = \theta_{ackNeg} \right.$

$r_8 :$  **if** ( $Disconfirm \in a_u \wedge |a_u| = 1$ ) **then**  
 $\quad \left\{ U(a_m = AskIntention) = \theta_{askIntention} \right.$   
**else**  
 $\quad \left\{ U(a_m = AskIntention) = \theta_{askIntentionNeg} \right.$

$r_9 :$  **if** ( $RepeatAll \in a_u \wedge sequence \neq None$ ) **then**  
 $\quad \left\{ U(a_m = DemonstrateAll) = \theta_{demonstrateAll} \right.$

$r_{10} :$  **if** ( $ForgetAll \in a_u \wedge sequence \neq None$ ) **then**  
 $\quad \left\{ U(a_m = ForgetAll) = \theta_{forgetAll} \right.$

$r_{11} :$  **if** ( $Compliment \in a_u \wedge a_m = DemonstrateAll$ ) **then**  
 $\quad \left\{ U(a_m = SayThankYou) = \theta_{sayThankYou} \right.$

$r_{12} :$  **if** ( $FollowMe \in a_u$ ) **then**  
 $\quad \left\{ U(a_m = FollowMe) = \theta_{followMe} \right.$

$r_{13} :$  **if** ( $Stop \in a_u$ ) **then**  
 $\quad \left\{ U(a_m = Stop) = \theta_{stop} \right.$

$r_{14} :$  **if** ( $SayHello \in a_u$ ) **then**  
 $\quad \left\{ U(a_m = SayHello) = \theta_{sayHello} \right.$

$r_{15} :$  **if** ( $SayGoodbye \in a_u$ ) **then**  
 $\quad \left\{ U(a_m = SayGoodbye) = \theta_{sayGoodbye} \right.$

## B.2 Experiments in Section 6.3

### First experiment

The six rules below specify the transition model used in the first experiment of Section 6.3. The first rule expresses the probability of the current task being completed after the last system action. If the task is completed, the user intention is reinitialised with a prior distribution given by rules  $r_3$ ,  $r_4$  and  $r_5$ . The user action model based on this user intention is then represented by rule  $r_6$ .

$r_1:$        $\forall X :$   
**if** ( $a_m = Do(X) \wedge i_u = X$ ) **then**  
 $\quad \left\{ P(\text{completed-task} = \text{true}) = 1.0 \right.$   
**else if** ( $a_m = Excuse(\cdot)$ ) **then**  
 $\quad \left\{ P(\text{completed-task} = \text{true}) = 1.0 \right.$   
**else if** ( $a_m = ConfirmDetection \wedge i_u = Do YouSee(\cdot)$ ) **then**  
 $\quad \left\{ P(\text{completed-task} = \text{true}) = 1.0 \right.$   
**else if** ( $a_m = DisconfirmDetection \wedge i_u = Do YouSee(\cdot)$ ) **then**  
 $\quad \left\{ P(\text{completed-task} = \text{true}) = 1.0 \right.$   
**else if** ( $a_m = Describe(\cdot) \wedge i_u = WhatDo YouSee$ ) **then**  
 $\quad \left\{ P(\text{completed-task} = \text{true}) = 1.0 \right.$   
**else if** ( $a_m = Do(\cdot) \vee a_m = Excuse(\cdot) \vee a_m = Describe(\cdot)$   
 $\quad \vee a_m = DisconfirmDetection \vee a_m = ConfirmDetection$ ) **then**  
 $\quad \left\{ \begin{array}{l} P(\text{completed-task} = \text{true}) = \theta_{1[0]} \\ P(\text{completed-task} = \text{false}) = \theta_{1[1]} \end{array} \right.$   
**else if** ( $i_u = None$ ) **then**  
 $\quad \left\{ P(\text{completed-task} = \text{true}) = 1.0 \right.$   
**else**  
 $\quad \left\{ P(\text{completed-task} = \text{false}) = 1.0 \right.$

$r_2:$       **if** ( $\text{completed-task} = \text{false}$ ) **then**  
 $\quad \left\{ P(i_u = i_u) = 1 \right.$

$r_3:$       **if** ( $\text{completed-task} = \text{true}$ ) **then**  
 $\quad \left\{ \begin{array}{l} P(i_u = Move(Left)) = \theta_{1[0]} \\ P(i_u = Move(Forward)) = \theta_{2[1]} \\ P(i_u = Move(Backward)) = \theta_{2[2]} \\ P(i_u = Move(Right)) = \theta_{2[3]} \\ P(i_u = Do YouSee(object_1)) = \theta_{2[4]} \\ P(i_u = Do YouSee(object_2)) = \theta_{2[5]} \\ P(i_u = WhatDo YouSee) = \theta_{2[6]} \\ P(\cdot) = \theta_{2[7]} \end{array} \right.$

$r_4:$        $\forall O :$   
**if** ( $\text{completed-task} = \text{true} \wedge O \in carried$ ) **then**  
 $\quad \left\{ \begin{array}{l} P(i_u = Release(O)) = \theta_{3[0]} \\ P(\cdot) = \theta_{3[1]} \end{array} \right.$

**else if** (*completed-task* = *true*) **then**  

$$\begin{cases} P(i_u = Release(O)) = \theta_{4[0]} \\ P(\cdot) = \theta_{4[1]} \end{cases}$$

*r*<sub>5</sub>:  $\forall O :$   
**if** (*completed-task* = *true*  $\wedge$  *carried* =  $[]$ ) **then**  

$$\begin{cases} P(i_u = PickUp(O)) = \theta_{5[0]} \\ P(\cdot) = \theta_{5[1]} \end{cases}$$
**else if** (*completed-task* = *true*) **then**  

$$\begin{cases} P(i_u = PickUp(O)) = \theta_{6[0]} \\ P(\cdot) = \theta_{6[1]} \end{cases}$$

*r*<sub>6</sub>:  $\forall X :$   
**if** (*a<sub>m</sub>* = *Ground*(*X*)  $\wedge$  *i<sub>u</sub>* = *X*) **then**  

$$\begin{cases} P(a_u^p = Confirm) = \theta_{7[0]} \\ P(a_u^p = Nothing) = \theta_{7[1]} \\ P(\cdot) = \theta_{7[2]} \end{cases}$$
**else if** (*a<sub>m</sub>* = *Ground*(*X*)  $\wedge$  *i<sub>u</sub>*  $\neq$  *X*  $\wedge$  *i<sub>u</sub>*  $\neq$  *None*) **then**  

$$\begin{cases} P(a_u^p = Disconfirm) = \theta_{8[0]} \\ P(a_u^p = Ask(i_u)) = \theta_{8[1]} \\ P(a_u^p = Nothing) = \theta_{8[2]} \\ P(\cdot) = \theta_{8[3]} \end{cases}$$
**else if** (*a<sub>m</sub>* = *Confirm*(*X*)  $\wedge$  *i<sub>u</sub>* = *X*) **then**  

$$\begin{cases} P(a_u^p = Confirm) = \theta_{9[0]} \\ P(a_u^p = Ask(i_u)) = \theta_{9[1]} \\ P(a_u^p = Nothing) = \theta_{9[2]} \\ P(\cdot) = \theta_{9[3]} \end{cases}$$
**else if** (*a<sub>m</sub>* = *Confirm*(*X*)  $\wedge$  *i<sub>u</sub>*  $\neq$  *X*  $\wedge$  *i<sub>u</sub>*  $\neq$  *None*) **then**  

$$\begin{cases} P(a_u^p = Disconfirm) = \theta_{10[0]} \\ P(a_u^p = Nothing) = \theta_{10[1]} \\ P(\cdot) = \theta_{10[2]} \end{cases}$$
**else if** (*a<sub>m</sub>* = *Do*(*X*)  $\wedge$  *i<sub>u</sub>* = *X*) **then**  

$$\begin{cases} P(a_u^p = RepeatLast) = \theta_{11[0]} \\ P(a_u^p = Ask(i_u)) = \theta_{11[1]} \\ P(a_u^p = Nothing) = \theta_{11[2]} \\ P(\cdot) = \theta_{11[3]} \end{cases}$$
**else if** (*a<sub>m</sub>* = *AskRepeat*  $\vee$  *completed-task* = *true*) **then**  

$$\begin{cases} P(a_u^p = Ask(i_u)) = \theta_{12[0]} \\ P(a_u^p = Nothing) = \theta_{12[1]} \\ P(\cdot) = \theta_{12[2]} \end{cases}$$
**else if** (*i<sub>u</sub>* = *None*) **then**  

$$\begin{cases} P(a_u^p = Nothing) = 1.0 \end{cases}$$
**else**  

$$\begin{cases} P(a_u^p = Ask(i_u)) = \theta_{13[0]} \\ P(a_u^p = Nothing) = \theta_{13[1]} \\ P(\cdot) = \theta_{13[2]} \end{cases}$$

## Second experiment

Two rule-structured models are compared to one another in the second experiment: a transition model for the model-based approach, and an action-value model for the model-free approach. The transition model is identical to the one listed above and will hence not be repeated. The action-value model is specified with a set of eleven rules. The input variables of these rules are the last user dialogue act  $a_u$ , the user dialogue act preceding it  $a_{u\text{-}prev}$ , the last system act  $a_m$ , and the two contextual variables *perceived* and *carried*.

```

 $r_1:$        $\forall X :$ 
if ( $a_u = Move(\cdot)$ ) then
     $\begin{cases} U(a_m = Do(a_u)) = \theta_{movements1} \\ U(a_m = Ground(a_u)) = \theta_{movements2} \end{cases}$ 
else if ( $a_u = Confirm \wedge a_{u\text{-}prev} = Move(\cdot)$ ) then
     $\begin{cases} U(a_m = Do(a_{u\text{-}prev})) = \theta_{movements1} \\ U(a_m = Ground(a_{u\text{-}prev})) = \theta_{movements2} \end{cases}$ 
else if ( $a_m = Ground(Move(X)) \wedge a_u \neq Disconfirm$ ) then
     $\begin{cases} U(a_m = Do(Move(X))) = \theta_{movements1} \end{cases}$ 

 $r_2:$        $\forall X :$ 
if ( $a_u = PickUp(X) \wedge X \in perceived \wedge carried = []$ ) then
     $\begin{cases} U(a_m = Do(a_u)) = \theta_{pickup1} \\ U(a_m = Ground(a_u)) = \theta_{pickup2} \end{cases}$ 
else if ( $a_u = PickUp(X) \wedge X \in perceived \wedge carried \neq []$ ) then
     $\begin{cases} U(a_m = Excuse(AlreadyCarryObject)) = \theta_{pickup3} \end{cases}$ 
else if ( $a_u = PickUp(X) \wedge X \notin perceived$ ) then
     $\begin{cases} U(a_m = Excuse(DoNotSeeObject)) = \theta_{pickup4} \end{cases}$ 
else if ( $a_u = Confirm \wedge a_{u\text{-}prev} = PickUp(X) \wedge X \in perceived \wedge carried = []$ ) then
     $\begin{cases} U(a_m = Do(a_{u\text{-}prev})) = \theta_{pickup1} \\ U(a_m = Ground(a_{u\text{-}prev})) = \theta_{pickup2} \end{cases}$ 
else if ( $a_u = Confirm \wedge a_{u\text{-}prev} = PickUp(X) \wedge X \in perceived \wedge carried \neq []$ ) then
     $\begin{cases} U(a_m = Excuse(AlreadyCarryObject)) = \theta_{pickup3} \end{cases}$ 
else if ( $a_u = Confirm \wedge a_{u\text{-}prev} = PickUp(X) \wedge X \notin perceived$ ) then
     $\begin{cases} U(a_m = Excuse(DoNotSeeObject)) = \theta_{pickup4} \end{cases}$ 
else if ( $a_m = Ground(PickUpX) \wedge a_u \neq Disconfirm \wedge X \in perceived \wedge carried = []$ ) then
     $\begin{cases} U(a_m = Do(PickUp(X))) = \theta_{pickup1} \end{cases}$ 
else if ( $a_m = Ground(PickUp(X)) \wedge a_u \neq Disconfirm \wedge X \in perceived \wedge carried \neq []$ ) then
     $\begin{cases} U(a_m = Excuse(AlreadyCarryObject)) = \theta_{pickup3} \end{cases}$ 
else if ( $a_m = Ground(PickUp(X)) \wedge a_u \neq Disconfirm \wedge X \notin perceived$ ) then
     $\begin{cases} U(a_m = Excuse(DoNotSeeObject)) = \theta_{pickup4} \end{cases}$ 

 $r_3:$        $\forall X :$ 
if ( $a_u = Release(X) \wedge X \in carried$ ) then
     $\begin{cases} U(a_m = Do(a_u)) = \theta_{release1} \\ U(a_m = Ground(a_u)) = \theta_{release2} \end{cases}$ 
else if ( $a_u = Release(X) \wedge X \notin carried$ ) then

```

$\begin{cases} U(a_m = \text{Excuse}(DoNotCarryObject)) = \theta_{\text{release3}} \end{cases}$   
**else if** ( $a_u = \text{Confirm} \wedge a_{u-\text{prev}} = \text{Release}(X) \wedge X \in \text{carried}$ ) **then**  
 $\begin{cases} U(a_m = \text{Do}(a_{u-\text{prev}})) = \theta_{\text{release1}} \\ U(a_m = \text{Ground}(a_{u-\text{prev}})) = \theta_{\text{release2}} \end{cases}$   
**else if** ( $a_u = \text{Confirm} \wedge a_{u-\text{prev}} = \text{Release}(X) \wedge X \notin \text{carried}$ ) **then**  
 $\begin{cases} U(a_m = \text{Excuse}(DoNotCarryObject)) = \theta_{\text{release3}} \end{cases}$   
**else if** ( $a_m = \text{Ground}(\text{Release}(X)) \wedge a_u \neq \text{Disconfirm} \wedge X \in \text{carried}$ ) **then**  
 $\begin{cases} U(a_m = \text{Do}(a_u)) = \theta_{\text{release1}} \\ U(a_m = \text{Ground}(a_u)) = \theta_{\text{release2}} \end{cases}$   
**else if** ( $a_m = \text{Ground}(\text{Release}(X)) \wedge a_u \neq \text{Disconfirm} \wedge X \notin \text{carried}$ ) **then**  
 $\begin{cases} U(a_m = \text{Excuse}(DoNotCarryObject)) = \theta_{\text{release3}} \end{cases}$

$r_4:$    **if** ( $a_u \neq \text{None}$ ) **then**  
 $\begin{cases} U(a_m = \text{None}) = \theta_{\text{none}} \end{cases}$   
**else**  
 $\begin{cases} U(a_m = \text{None}) = \theta_{\text{none2}} \end{cases}$

$r_5:$    **if** ( $a_u \neq \text{Confirm} \wedge a_u \neq \text{RepeatLast} \wedge a_m \neq \text{Ground}(\cdot)$ ) **then**  
 $\begin{cases} U(a_m = \text{Do}(\cdot) \wedge a_m \neq \text{Do}(a_u)) = \theta_{\text{wrong1}} \\ U(a_m = \text{Ground}(\cdot) \wedge a_m \neq \text{Ground}(a_u)) = \theta_{\text{wrong2}} \\ U(a_m = \text{Excuse}(\cdot)) = \theta_{\text{wrong3}} \end{cases}$   
**else if** ( $a_u = \text{Confirm}$ ) **then**  
 $\begin{cases} U(a_m = \text{Do}(\cdot) \wedge a_m \neq \text{Do}(a_{u-\text{prev}})) = \theta_{\text{wrong1}} \\ U(a_m = \text{Ground}(\cdot) \wedge a_m \neq \text{Ground}(a_{u-\text{prev}})) = \theta_{\text{wrong2}} \\ U(a_m = \text{Excuse}(\cdot)) = \theta_{\text{wrong3}} \end{cases}$

$r_6:$    **if** ( $a_m \neq \text{AskRepeat}$ ) **then**  
 $\begin{cases} U(a_m = \text{AskRepeat}) = \theta_{\text{repeat}} \end{cases}$

$r_7:$    **if** ( $a_u = \text{Disconfirm}$ ) **then**  
 $\begin{cases} U(a_m = \text{AskRepeat}) = \theta_{\text{repeat2}} \end{cases}$

$r_8:$    **if** ( $\text{true}$ ) **then**  
 $\begin{cases} U(a_m = \text{Confirm}(a_u)) = \theta_{\text{confirm1}} \\ U(a_m = \text{Confirm}(\cdot) \wedge a_m \neq \text{Confirm}(a_u)) = \theta_{\text{confirm2}} \end{cases}$

$r_9:$     $\forall X :$   
**if** ( $a_u = \text{DoYouSee}(X) \wedge X \in \text{perceived}$ ) **then**  
 $\begin{cases} U(a_m = \text{Confirm}) = \theta_{\text{doyousee1}} \\ U(a_m = \text{Disconfirm}) = \theta_{\text{doyousee2}} \end{cases}$   
**else if** ( $a_u = \text{DoYouSee}(X) \wedge X \notin \text{perceived}$ ) **then**  
 $\begin{cases} U(a_m = \text{Disconfirm}) = \theta_{\text{doyousee3}} \\ U(a_m = \text{Confirm}) = \theta_{\text{doyousee4}} \end{cases}$   
**else if** ( $a_u = \text{Confirm} \wedge a_{u-\text{prev}} = \text{DoYouSee}(X) \wedge X \in \text{perceived}$ ) **then**  
 $\begin{cases} U(a_m = \text{Confirm}) = \theta_{\text{doyousee1}} \\ U(a_m = \text{Disconfirm}) = \theta_{\text{doyousee2}} \end{cases}$   
**else if** ( $a_u = \text{Confirm} \wedge a_{u-\text{prev}} = \text{DoYouSee}(X) \wedge X \notin \text{perceived}$ ) **then**

```


$$\begin{cases} U(a_m = Disconfirm) = \theta_{doyousee3} \\ U(a_m = Confirm) = \theta_{doyousee4} \end{cases}$$

else

$$\begin{cases} U(a_m = Confirm) = \theta_{doyousee5} \\ U(a_m = Disconfirm) = \theta_{doyousee6} \end{cases}$$

r10: if ( $a_u = WhatDoYouSee$ ) then

$$\begin{cases} U(a_m = Describe(perceived)) = \theta_{whatdoyousee1} \end{cases}$$

else if ( $a_u = Confirm \wedge a_{u-prev} = WhatDoYouSee$ ) then

$$\begin{cases} U(a_m = Describe(perceived)) = \theta_{whatdoyousee1} \end{cases}$$

else

$$\begin{cases} U(a_m = Describe(\cdot)) = \theta_{whatdoyousee2} \end{cases}$$

r11: if ( $a_u = RepeatLast$ ) then

$$\begin{cases} U(a_m = a_m) = \theta_{repeatLast} \end{cases}$$


```

## B.3 Experiments in Chapter 8

We list below the set of rules employed for the Approach 3 (rule-structured model) of the user evaluation experiment.

### Natural language understanding model

The natural language understanding model is composed of a set of simple, deterministic template-matching rules. The rules map the presence of particular words or phrases in the user utterance  $u_u$  to specific user dialogue acts  $a_u$ .

The model is divided in two phases. Rules  $r_1$  to  $r_{16}$  are responsible for defining the general structure of the dialogue acts. The last rule  $r_{17}$  is specifically devoted to the resolution of references to visual objects and the distinction between short and long movements.

```

r1: if ("forward"  $\in u_u \vee$  "forwards"  $\in u_u \vee$  "straight"  $\in u_u$ ) then

$$\begin{cases} P(a_u^{temp} = Move(Forward)) = 1 \end{cases}$$

r2: if ("backward"  $\in u_u \vee$  "backwards"  $\in u_u \vee$  "back"  $\in u_u \vee$  "turn"  $\notin u_u$ ) then

$$\begin{cases} P(a_u^{temp} = Move(Backward)) = 1 \end{cases}$$

r3: if ("left"  $\in u_u$ ) then

$$\begin{cases} P(a_u^{temp} = Move(Left)) = 1 \end{cases}$$

r4: if ("go * right"  $\in u_u \vee$  "turn * right"  $\in u_u \vee$  "move * right"  $\in u_u \vee$  "rotate * right"  $\in u_u$ ) then

$$\begin{cases} P(a_u^{temp} = Move(Right)) = 1 \end{cases}$$

r5: if ("turn back"  $\in u_u \vee$  "turn around"  $\in u_u \vee$  "one hundred eighty degrees"  $\in u_u \vee$  "u turn"  $\in u_u$ ) then

$$\begin{cases} P(a_u^{temp} = Move(Turn)) = 1 \end{cases}$$

r6:  $\forall X :$ 
if ("pick up X"  $\in u_u \vee$  "pick X up"  $\in u_u \vee$  "take X"  $\in u_u \vee$  "grasp X"  $\in u_u$ ) then

$$\begin{cases} P(a_u^{temp} = PickUp(X)) = 1 \end{cases}$$


```

$r_7:$    **if** ("release"  $\in u_u \vee$  "put \* down"  $\in u_u \vee$  "put \* on the floor"  $\in u_u \vee$  "put \* on the table"  $\in u_u$ ) **then**  
     $\quad \left\{ P(a_u^{temp} = Release) = 1 \right.$   
 $r_8:$    **if** ("what \* you see"  $\in u_u \vee$  "what \* you perceive"  $\in u_u$ ) **then**  
     $\quad \left\{ P(a_u^{temp} = WhatDoYouSee) = 1 \right.$   
 $r_9:$     $\forall X :$   
    **if** ("you see X"  $\in u_u \vee$  "you perceive X"  $\in u_u$ ) **then**  
     $\quad \left\{ P(a_u^{temp} = DoYouSee(X)) = 1 \right.$   
 $r_{10}:$    **if** ("yes"  $\in u_u \vee$  "exactly"  $\in u_u \vee$  "correct"  $\in u_u$ ) **then**  
     $\quad \left\{ P(a_u^{temp} = Confirm) = 1 \right.$   
 $r_{11}:$    **if** ("thanks"  $\in u_u \vee$  "thank"  $\in u_u \vee$  "correct"  $\in u_u \vee$  "perfect"  $\in u_u \vee$  "great"  $\in u_u \vee$  "good"  $\in u_u$   
     $\quad \vee$  ("right"  $\in u_u \wedge$  "turn"  $\notin u_u \wedge$  "rotate"  $\notin u_u \wedge$  "move"  $\notin u_u \wedge$  "go"  $\notin u_u$ )  $\vee$  "ok"  $\in u_u$ ) **then**  
     $\quad \left\{ P(a_u^{temp} = Grounding) = 1 \right.$   
 $r_{12}:$    **if** ("no"  $\in u_u \vee$  "wrong"  $\in u_u \vee$  "incorrect"  $\in u_u$ ) **then**  
     $\quad \left\{ P(a_u^{temp} = Disconfirm) = 1 \right.$   
 $r_{13}:$    **if** ("once more"  $\in u_u \vee$  "one more time"  $\in u_u \vee$  "repeat"  $\in u_u \vee$  "again"  $\in u_u$ ) **then**  
     $\quad \left\{ P(a_u^{temp} = RepeatLast) = 1 \right.$   
 $r_{14}:$    **if** ("hi"  $\in u_u \vee$  "hello"  $\in u_u \vee$  "good afternoon"  $\in u_u \vee$  "good morning"  $\in u_u$ ) **then**  
     $\quad \left\{ P(a_u^{temp} = Greeting) = 1 \right.$   
 $r_{15}:$    **if** ("bye"  $\in u_u \vee$  "goodbye"  $\in u_u$ ) **then**  
     $\quad \left\{ P(a_u^{temp} = Closing) = 1 \right.$   
 $r_{16}:$    **if** ("stop"  $\in u_u$ ) **then**  
     $\quad \left\{ P(a_u^{temp} = Stop) = 1 \right.$   
 $r_{17}:$     $\forall X :$   
    **if** ("bit"  $\in u_u \wedge a_u^{temp} = Move(X)$ ) **then**  
     $\quad \left\{ P(a_u = Move(X, Short)) = 1 \right.$   
    **else if** ( $a_u^{temp} = Move(X) \wedge (X = Left \vee X = Right) \wedge$  "thirty degrees"  $\in u_u$ ) **then**  
     $\quad \left\{ P(a_u = Move(X, Short)) = 1 \right.$   
    **else if** ( $a_u^{temp} = PickUp(*) \wedge$  "feet"  $\in u_u$ ) **then**  
     $\quad \left\{ P(a_u = PickUp(AtFeet)) = 1 \right.$   
    **else if** ( $a_u^{temp} = PickUp(X) \wedge$  "red"  $= X$ ) **then**  
     $\quad \left\{ P(a_u = PickUp(RedObj)) = 1 \right.$   
    **else if** ( $a_u^{temp} = PickUp(X) \wedge$  ("blue"  $= X \vee$  ("it"  $= X \wedge (BlueObj \in a_u \vee BlueObj \in a_m))$   
     $\quad \vee$  ("the"  $= X \wedge perceived = [BlueObj]))$ ) **then**  
     $\quad \left\{ P(a_u = PickUp(BlueObj)) = 1 \right.$   
    **else if** ( $a_u^{temp} = PickUp(X) \wedge$  ("red"  $= X \vee$  ("it"  $= X \wedge (RedObj \in a_u \vee RedObj \in a_m))$   
     $\quad \vee$  ("the"  $= X \wedge perceived = [RedObj]))$ ) **then**  
     $\quad \left\{ P(a_u = PickUp(RedObj)) = 1 \right.$   
    **else if** ( $a_u^{temp} = PickUp(*)$ ) **then**

```


$$\begin{cases} P(a_u = \text{PickUp}(Other)) = 1 \\ \text{else if } (a_u^{temp} = \text{Do YouSee}(X) \wedge \text{"red"} = X) \text{ then} \\ \quad \begin{cases} P(a_u = \text{Do YouSee}(RedObj)) = 1 \\ \text{else if } (a_u^{temp} = \text{Do YouSee}(X) \wedge \text{"blue"} = X) \text{ then} \\ \quad \begin{cases} P(a_u = \text{Do YouSee}(BlueObj)) = 1 \\ \text{else if } (a_u^{temp} = \text{Do YouSee}(X) \wedge \text{"it"} \notin X) \text{ then} \\ \quad \begin{cases} P(a_u = \text{WhatDo YouSee}) = 1 \\ \text{else} \\ \quad \begin{cases} P(a_u = a_u^{temp}) = 1 \end{cases} \end{cases} \end{cases} \end{cases}$$


```

### Task completion model

The task completion model considers a task to be completed whenever the system performs an action that is not a clarification request. The probability of the user changing his mind before the completion of the task is defined by the parameter  $\theta_{\text{changemind}}$ .

```


$$r_1: \quad \begin{cases} \text{if } (a_m \neq \text{Confirm}(\cdot) \wedge a_m \neq \text{AskRepeat} \wedge a_m \neq \text{None}) \text{ then} \\ \quad \begin{cases} P(\text{completed-task} = \text{true}) = 1 \\ \text{else if } (i_u = \text{None}) \text{ then} \\ \quad \begin{cases} P(\text{completed-task} = \text{true}) = 1 \\ \text{else} \\ \quad \begin{cases} P(\text{completed-task} = \text{false}) = \theta_{\text{changemind}[0]} \\ P(\text{completed-task} = \text{true}) = \theta_{\text{changemind}[1]} \end{cases} \end{cases} \end{cases} \end{cases}$$


```

$r_2:$        $\forall X :$

```


$$\begin{cases} \text{if } (a_m = \text{Do}(X)) \text{ then} \\ \quad \begin{cases} P(\text{lastMove} = X) = 1 \end{cases} \end{cases}$$


```

### User goal model

The user goal model encodes how the user intention  $i_u$  is expected to evolve over time, given the current dialogue context. The intention is only assumed to change when the current task is completed. Rule  $r_2$  defines the likelihood of instructions related to spatial movements, while  $r_3$  and  $r_4$  define the likelihood of user questions related to the robot perception,  $r_5$  the likelihood of greeting and closing actions,  $r_6$  the likelihood of releasing actions, and  $r_7$  and  $r_8$  the likelihood of grasping actions.

Rule  $r_3$  and  $r_8$  are used to increase the probability of grasping instructions and questions about visual objects whenever an object is actually perceived by the robot.

```


$$r_1: \quad \begin{cases} \text{if } (\text{completed-task} = \text{false}) \text{ then} \\ \quad \begin{cases} P(i_u = i_u) = 1 \end{cases} \end{cases}$$


```

$r_2: \quad \begin{cases} \text{if } (\text{completed-task} = \text{true}) \text{ then} \end{cases}$

$$\begin{cases} P(i_u = Move(Left)) = \theta_{moveprior[0]} \\ P(i_u = Move(Forward)) = \theta_{moveprior[1]} \\ P(i_u = Move(Backward)) = \theta_{moveprior[2]} \\ P(i_u = Move(Right)) = \theta_{moveprior[3]} \\ P(i_u = Move(Left, Short)) = \theta_{moveprior[4]} \\ P(i_u = Move(Forward, Short)) = \theta_{moveprior[5]} \\ P(i_u = Move(Backward, Short)) = \theta_{moveprior[6]} \\ P(i_u = Move(Right, Short)) = \theta_{moveprior[7]} \\ P(i_u = Move(Turn)) = \theta_{moveprior[8]} \\ P(\cdot) = \theta_{moveprior[9]} \end{cases}$$

$r_3:$

$$\forall X : \begin{cases} \text{if } (X \in perceived \wedge completed-task = true) \text{ then} \\ \quad \begin{cases} P(i_u = DoYouSee(X)) = \theta_{perceive_see_prior[0]} \\ P(i_u = WhatDoYouSee) = \theta_{perceive_see_prior[1]} \\ P(\cdot) = \theta_{perceive_see_prior[2]} \end{cases} \end{cases}$$

$r_4:$

$$\begin{cases} \text{if } (completed-task = true) \text{ then} \\ \quad \begin{cases} P(i_u = DoYouSee(Obj)) = \theta_{perceive_nosee_prior[0]} \\ P(i_u = DoYouSee(BlueObj)) = \theta_{perceive_nosee_prior[1]} \\ P(i_u = WhatDoYouSee) = \theta_{perceive_nosee_prior[2]} \\ P(\cdot) = \theta_{perceive_nosee_prior[3]} \end{cases} \end{cases}$$

$r_5:$

$$\begin{cases} \text{if } (lastMove = None \wedge completed-task = true) \text{ then} \\ \quad \begin{cases} P(i_u = Greeting) = \theta_{openingprior[0]} \\ P(\cdot) = \theta_{openingprior[1]} \end{cases} \\ \text{else if } (lastMove = Release) \text{ then} \\ \quad \begin{cases} P(i_u = Closing) = \theta_{closingprior[0]} \\ P(\cdot) = \theta_{closingprior[1]} \end{cases} \end{cases}$$

$r_6:$

$$\begin{cases} \text{if } (completed-task = true \wedge carried \neq [] \wedge carried \neq None) \text{ then} \\ \quad \begin{cases} P(i_u = Release) = \theta_{releaseprior[0]} \\ P(\cdot) = \theta_{releaseprior[1]} \end{cases} \end{cases}$$

$r_7:$

$$\forall X : \begin{cases} \text{if } (completed-task = true \wedge carried = [] \wedge X \in perceived) \text{ then} \\ \quad \begin{cases} P(i_u = PickUp(X)) = \theta_{pickup_see_prior[0]} \\ P(\cdot) = \theta_{pickup_see_prior[1]} \end{cases} \end{cases}$$

$r_8:$

$$\begin{cases} \text{if } (completed-task = true \wedge carried = []) \text{ then} \\ \quad \begin{cases} P(i_u = PickUp(Obj)) = \theta_{pickup_nosee_prior[0]} \\ P(i_u = PickUp(BlueObj)) = \theta_{pickup_nosee_prior[1]} \\ P(i_u = PickUp(AtFeet)) = \theta_{pickup_nosee_prior[2]} \\ P(\cdot) = \theta_{pickup_nosee_prior[3]} \end{cases} \end{cases}$$

## User action model

The user action model derives the possible user actions  $a_u$  as a function of the current user intention  $i_u$  and the last system action  $a_m$ .

$r_1:$       **if** ( $a_m = \text{Confirm}(X) \wedge i_u = X$ ) **then**  
 $\quad \quad \quad \left\{ \begin{array}{l} P(a_u = \text{Confirm}) = \theta_{\text{confirmpredict}[0]} \\ P(a_u = i_u) = \theta_{\text{confirmpredict}[1]} \\ P(\cdot) = \theta_{\text{confirmpredict}[2]} \end{array} \right.$   
**else if** ( $a_m = \text{Confirm}(X) \wedge i_u \neq X \wedge i_u \neq \text{None}$ ) **then**  
 $\quad \quad \quad \left\{ \begin{array}{l} P(a_u = \text{Disconfirm}) = \theta_{\text{disconfirmpredict}[0]} \\ P(a_u = i_u) = \theta_{\text{disconfirmpredict}[1]} \\ P(\cdot) = \theta_{\text{disconfirmpredict}[2]} \end{array} \right.$   
**else if** ( $\text{lastMove} = i_u$ ) **then**  
 $\quad \quad \quad \left\{ \begin{array}{l} P(a_u = \text{RepeatLast}) = \theta_{\text{samemove}[0]} \\ P(a_u = i_u) = \theta_{\text{samemove}[1]} \\ P(\cdot) = \theta_{\text{samemove}[2]} \end{array} \right.$   
**else if** ( $a_m = \text{AskRepeat} \vee \text{completed} - \text{task} = \text{true}$ ) **then**  
 $\quad \quad \quad \left\{ \begin{array}{l} P(a_u = i_u) = \theta_{\text{intent}[0]} \\ P(\cdot) = \theta_{\text{intent}[1]} \end{array} \right.$   
**else if** ( $i_u \neq \text{None}$ ) **then**  
 $\quad \quad \quad \left\{ \begin{array}{l} P(a_u = i_u) = \theta_{\text{otheccase}[0]} \\ P(\cdot) = \theta_{\text{otheccase}[1]} \end{array} \right.$   
  
 $r_2:$       **if** ( $a_m = \text{Do}(\cdot)$ ) **then**  
 $\quad \quad \quad \left\{ \begin{array}{l} P(a_u = \text{Stop}) = \theta_{\text{aftermove}[0]} \\ P(a_u = \text{Disconfirm}) = \theta_{\text{aftermove}[1]} \\ P(\cdot) = \theta_{\text{aftermove}[2]} \end{array} \right.$

## Utility model

The utility model captures the relative utility of various system actions depending on the current hypotheses about the user intention  $i_u$ , last user action  $a_u$ , and general dialogue context.

$r_1:$       **if** ( $i_u = \text{Move}(\cdot)$ ) **then**  
 $\quad \quad \quad \left\{ U(a_m = \text{Do}(i_u)) = \theta_{\text{move}} \right.$   
  
 $r_2:$       **if** ( $i_u = \text{PickUp}(X) \vee X \in \text{perceived} \vee \text{carried} = [] \wedge i_u = \text{PickUp}(\text{AtFeet}) \vee \text{carried} = []$ ) **then**  
 $\quad \quad \quad \left\{ U(a_m = \text{Do}(i_u)) = \theta_{\text{pickup}} \right.$   
**else if** ( $i_u = \text{PickUp}(Y) \wedge Y \in \text{perceived} \wedge \text{carried} \neq []$ ) **then**  
 $\quad \quad \quad \left\{ U(a_m = \text{Say}(\text{AlreadyCarryObject})) = \theta_{\text{excuse}} \right.$   
**else if** ( $i_u = \text{PickUp}(Z) \wedge Z = \text{Other}$ ) **then**  
 $\quad \quad \quad \left\{ U(a_m = \text{Say}(\text{CannotResolve})) = \theta_{\text{excuse}} \right.$   
**else if** ( $i_u = \text{PickUp}(Z) \wedge Z \notin \text{perceived}$ ) **then**  
 $\quad \quad \quad \left\{ U(a_m = \text{Say}(\text{DoNotSeeObject})) = \theta_{\text{excuse}} \right.$   
  
 $r_3:$       **if** ( $i_u = \text{Release} \wedge \text{carried} \neq \text{None} \wedge \text{carried} \neq []$ ) **then**  
 $\quad \quad \quad \left\{ U(a_m = \text{Do}(i_u)) = \theta_{\text{release}} \right.$   
**else if** ( $i_u = \text{Release} \wedge \text{carried} = []$ ) **then**  
 $\quad \quad \quad \left\{ U(a_m = \text{Say}(\text{DoNotCarryObject})) = \theta_{\text{excuse}} \right.$

```

r4:      if ( $i_u = Do\ YouSee(A) \wedge A \in perceived$ ) then
           $\begin{cases} U(a_m = Say(Confirm)) = \theta_{confirmpercept} \end{cases}$ 
      else if ( $i_u = Do\ YouSee(B) \wedge B \notin perceived$ ) then
           $\begin{cases} U(a_m = Say(Disconfirm)) = \theta_{confirmpercept} \end{cases}$ 
r5:      if ( $i_u = WhatDo\ YouSee$ ) then
           $\begin{cases} U(a_m = Describe(perceived)) = \theta_{describe} \end{cases}$ 
r6:      if ( $i_u = Greeting$ ) then
           $\begin{cases} U(a_m = Say(Greet)) = \theta_{chitchat} \end{cases}$ 
      else if ( $i_u = Closing$ ) then
           $\begin{cases} U(a_m = Say(Goodbye)) = \theta_{chitchat} \end{cases}$ 
r7:       $\begin{cases} U(a_m = Do(\cdot)) = -10 \\ U(a_m = Say(\cdot)) = -10 \\ U(a_m = Describe(\cdot)) = -10 \end{cases}$ 
r8:      if ( $a_m \neq AskRepeat \wedge a_u \neq None \wedge a_u \neq Confirm$ 
         $\wedge a_u \neq Disconfirm \wedge a_u \neq Grounding$ ) then
           $\begin{cases} U(a_m = AskRepeat) = \theta_{repeat1} \end{cases}$ 
      else
           $\begin{cases} U(a_m = AskRepeat) = \theta_{repeat2} \end{cases}$ 
r9:      if ( $i_u \neq Do\ YouSee(\cdot) \wedge i_u \neq WhatDo\ YouSee \wedge i_u \neq Greeting$ 
         $\wedge i_u \neq Closing \wedge i_u \neq None$ ) then
           $\begin{cases} U(a_m = Confirm(i_u)) = \theta_{confirm1} \end{cases}$ 
      else
           $\begin{cases} U(a_m = Confirm(\cdot)) = \theta_{confirm2} \end{cases}$ 
r10:     if ( $motion = true \wedge a_u = Stop$ ) then
           $\begin{cases} U(a_m = Do(Stop)) = \theta_{stop} \end{cases}$ 
      else if ( $motion = true \wedge a_u = Disconfirm$ ) then
           $\begin{cases} U(a_m = Do(Stop)) = \theta_{stop} \end{cases}$ 
      else if ( $motion = true$ ) then
           $\begin{cases} U(a_m \neq None) = \theta_{whilemove} \end{cases}$ 

```

## Generation model

The generation model maps each selected system action  $a_m$  to its corresponding verbal realisation  $u_m$ . The surface form is then picked up by the speech synthesis module.

```

r1:      if ( $a_m = Do(Move(Turn))$ ) then
           $\begin{cases} U(u_m = "OK, now turning back") = 1 \end{cases}$ 
      else if ( $a_m = Do(Move(X, Short))$ ) then
           $\begin{cases} U(u_m = "OK, moving X a little bit") = 1 \end{cases}$ 
      else if ( $a_m = Do(Move(X))$ ) then

```

```

 $\begin{cases} U(u_m = "OK, moving X") = 1 \end{cases}$ 
else if ( $a_m = Do(PickUp(RedObj))$ ) then
 $\begin{cases} U(u_m = "OK, picking up the red object") = 1 \end{cases}$ 
else if ( $a_m = Do(PickUp(BlueObj))$ ) then
 $\begin{cases} U(u_m = "OK, picking up the blue object") = 1 \end{cases}$ 
else if ( $a_m = Do(PickUp(AtFeet))$ ) then
 $\begin{cases} U(u_m = "OK, picking up the object at my feet") = 1 \end{cases}$ 
else if ( $a_m = Do(Release)$ ) then
 $\begin{cases} U(u_m = "OK, putting down the object") = 1 \end{cases}$ 
else if ( $a_m = Do(Stop)$ ) then
 $\begin{cases} U(u_m = "OK, stopping now") = 1 \end{cases}$ 
else if ( $a_m = AskRepeat$ ) then
 $\begin{cases} U(u_m = "Sorry, could you repeat?") = 1 \end{cases}$ 
else if ( $a_m = Say(AlreadyCarryObject)$ ) then
 $\begin{cases} U(u_m = "Sorry, I cannot do that, I am already carrying an object") = 1 \end{cases}$ 
else if ( $a_m = Say(DoNotCarryObject)$ ) then
 $\begin{cases} U(u_m = "Sorry, I cannot do that, I am not carrying the object") = 1 \end{cases}$ 
else if ( $a_m = Say(DoNotSeeObject)$ ) then
 $\begin{cases} U(u_m = "Sorry, I cannot do that, I do not see the object") = 1 \end{cases}$ 
else if ( $a_m = Say(CannotResolve)$ ) then
 $\begin{cases} U(u_m = "Sorry, I don't know which object you are talking about") = 1 \end{cases}$ 
else if ( $a_m = Confirm(Move(Left))$ ) then
 $\begin{cases} U(u_m = "Should I move left?") = 1 \end{cases}$ 
else if ( $a_m = Confirm(Move(Left, Short))$ ) then
 $\begin{cases} U(u_m = "Should I move a bit to the left?") = 1 \end{cases}$ 
else if ( $a_m = Confirm(Move(Right))$ ) then
 $\begin{cases} U(u_m = "Should I move right?") = 1 \end{cases}$ 
else if ( $a_m = Confirm(Move(Right, Short))$ ) then
 $\begin{cases} U(u_m = "Should I move a bit to the right?") = 1 \end{cases}$ 
else if ( $a_m = Confirm(Move(Forward))$ ) then
 $\begin{cases} U(u_m = "Should I move forward?") = 1 \end{cases}$ 
else if ( $a_m = Confirm(Move(Forward, Short))$ ) then
 $\begin{cases} U(u_m = "Should I move a bit forward?") = 1 \end{cases}$ 
else if ( $a_m = Confirm(Move(Backward))$ ) then
 $\begin{cases} U(u_m = "Should I move backward?") = 1 \end{cases}$ 
else if ( $a_m = Confirm(Move(Backward, Short))$ ) then
 $\begin{cases} U(u_m = "Should I step a bit backward?") = 1 \end{cases}$ 

```

```

else if ( $a_m = \text{Confirm}(\text{Move}(Turn))$ ) then
     $\{U(u_m = \text{"Should I turn 180 degrees?"}) = 1\}$ 

else if ( $a_m = \text{Confirm}(Stop)$ ) then
     $\{U(u_m = \text{"Should I stop?"}) = 1\}$ 

else if ( $a_m = \text{Confirm}(Release)$ ) then
     $\{U(u_m = \text{"Should I release the object?"}) = 1\}$ 

else if ( $a_m = \text{Confirm}(\text{PickUp}(RedObj))$ ) then
     $\{U(u_m = \text{"Should I pick up the red object?"}) = 1\}$ 

else if ( $a_m = \text{Confirm}(\text{PickUp}(BlueObj))$ ) then
     $\{U(u_m = \text{"Should I pick up the blue object?"}) = 1\}$ 

else if ( $a_m = \text{Confirm}(\text{PickUp}(AtFeet))$ ) then
     $\{U(u_m = \text{"Should I pick up the object at my feet?"}) = 1\}$ 

else if ( $a_m = \text{Say}(\text{Confirm})$ ) then
     $\{U(u_m = \text{"Yes"}) = 1\}$ 

else if ( $a_m = \text{Say}(\text{Disconfirm})$ ) then
     $\{U(u_m = \text{"No"}) = 1\}$ 

else if ( $a_m = \text{Describe}(X) \wedge X = []$ ) then
     $\{U(u_m = \text{"I do not see anything"}) = 1\}$ 

else if ( $a_m = \text{Describe}(X) \wedge RedObj \in X \wedge BlueObj \in X$ ) then
     $\{U(u_m = \text{"I see a red and a blue cylinder"}) = 1\}$ 

else if ( $a_m = \text{Describe}(X) \wedge RedObj \in X$ ) then
     $\{U(u_m = \text{"I see a red cylinder"}) = 1\}$ 

else if ( $a_m = \text{Describe}(X) \wedge BlueObj \in X$ ) then
     $\{U(u_m = \text{"I see a blue cylinder"}) = 1\}$ 

else if ( $a_m = \text{Say}(Greet)$ ) then
     $\{U(u_m = \text{"Hi there"}) = 1\}$ 

else if ( $a_m = \text{Say}(Goodbye)$ ) then
     $\{U(u_m = \text{"Bye, see you next time"}) = 1\}$ 

```