Issue-based Dialogue Management

Staffan Larsson

Department of Linguistics
Göteborg University, Sweden
2002

Issue-based Dialogue Management

Staffan Larsson

Doctoral dissertation

Publically defended in Lilla Hörsalen,
Humanisten, Göteborg University,
on June 13, 2002, at 10.00
for the degree of Doctor of Philosophy

Department of Linguistics
Göteborg University, Sweden
2002

Issue-based Dialogue Management

Abstract

The purpose of studying dialogue modelling and dialogue management is to provide models allowing us to explore how language, and especially spoken dialogue, is used in different activities. This thesis shows how issues (modelled semantically as questions) in general can be used as a basis for dialogue management.

In an abstract sense, the goal of all practical dialogue is to communicate information which is useful in some activity. This means that conversational goals should describe missing information, and to fulfil a conversational goal, what we need to do is to communicate the missing information. Issues, or questions, are essentially entities specifying certain pieces of as-yet-unavailable information. That is, conversational goals can to a large extent be modelled as questions.

The exploration of issue-based dialogue management in this thesis proceeds both on a theoretical and a practical implementation level. The thesis first explores basic issue-based dialogue management and implement a basic system illustrating the use of issue-based dialogue management. In the following chapters, this simple system is extended to handle grounding issues, addressing unraised issues, and action-oriented and negotiative dialogue. Finally, based on the differences between the successive versions of the theory (and implementation), we provide a characterization of different types of dialogue.

The issue-based approach to dialogue management turns out to be very useful for formulating general and theoretically motivated accounts of important aspects of dialogue, such as inquiry-oriented dialogue interactions, dealing with multiple simultaneous tasks, sharing information between tasks, grounding, interactive communication management, question accommodation, simple belief revision, action-oriented dialogue, and simple negotiative dialogue. The model is can be implemented rather straightforwardly using the TrindiKit, which has proven to be a very useful tool for exploring the issue-based approach.

KEY WORDS: dialogue management, dialogue modelling, dialogue systems, questions under discussion, grounding, accommodation, negotiative dialogue

The thesis is written in English.

ISBN 91-628-5301-5

Acknowledgements

I am the computerized psychotherapist. Please, describe your problems. Each time you are finished talking, type RET twice.
> Well, it's not really a problem, but I am finally done with my thesis!
Why do you say you are finally done with your thesis?
> Well because I am, and now I'd like to thank some people who helped me get it done.
Is it because you are and now you would like to thanks some people who helped you get it done that you came to me?
> Um... right.
Can you elaborate on that?
> I'd be glad to. First of all I'd like to thank my main supervisor Robin Cooper, who (apart from supervising this thesis and taking the time for hours and hours of inspiring discussions and generally being very helpful) has gotten me involved in lots of interesting projects[1][2][3][4][5] where I've met and learnt from lots of other people that I'd also like to thank: Jonathan Ginzburg, David Traum, Colin Matheson, Johan Bos, David Milward, Annie Zaenen, Ian Lewin, Arne Jönsson, Linda Bell, Lena Santamarta, Lars Ahrenberg, and... well it's a rather long list of people actually. I don't want to bore you with all the details, but I think they know who they are (and I don't mean that as a psychological remark!). Oh, and I almost forgot about all the conferences I've been privileged to attend, where I've also met lots of nice and clever people like Candy Sidner, Jörn Kreutel, Joris Hulstijn, Ivana Kruyff-Korbayova, Enric Vallduvi, and so on and so on... And I mustn't forget to thank STINT[6] for a scholarship that enabled me to visit the HCRC in Edinburgh for a couple of enjoyable months!
What makes you believe you might want to thank?
> Interesting question... of course, you have to have this kind of thing in your thesis, but I believe (since you're taking that perspective) that I'm genuinely grateful to these people, after all I couldn't have done it without them. Take, for example, my second supervisor Jens Allwood who I've had many heated (and re-heated) discussions with, and all the people at the department of Linguistics here in Göteborg, for example Stina Ericsson, Leif Grönqvist, Torbjörn Lager, Åsa Wengelin (who actually persuaded me to study computational linguistics years ago), Ylva Hård af Segerstad, Per Olofsson, Petter Karlström, Robert Andersson, Ulrika Ferm, Åsa Nordqvist, and lots of others.
Does it bother you that you believe since I am taking that perspective that you are genuinely grateful to these people?

[1]INDI (Information Exchange in Dialogue), Riksbankens Jubileumsfond 1997-0134.
[2]TRINDI (Task Oriented Instructional Dialogue), EC Project LE4-8314
[3]D'Homme (Dialogues in the Home Machine Environment), EC Project IST-2000-26280.
[4]SDS (Swedish Dialogue Systems), NUTEK/HSFR Language Technology Project F1472/1997
[5]SIRIDUS (Specification, Interaction and Reconfiguration in Dialogue Understanding Systems), EC Project IST-1999-10516
[6]STINT (The Swedish Foundation for International Cooperation in Research and Higher Education).

> No, not one bit.
Maybe your plans have something to do with this.
> Hey, what are you insinuating?
Is it because of your life that you say hey what are me insinuating?
> Ah, my life - interesting subject. I've been trying pretty hard these last few years to have a life, and when I've succeeded it's been due to my friends (for instance Klara, Andreas, Sara, Ola, Lukas, Maya, Jacob, Jakob, Björn, Andreas, Kim, Peter, Martin, Anna, Tina, Marie, Karin and John) and my family.
Tell me something about your family.
> Mm, I thought you might say that. Well, they've been really supportive of whatever I've been up to, and visiting my parents has always been very relaxing, although perhaps less so when my sisters and their families (including small children) are visiting, that is, in which case it is, well, maybe still relaxing but in a different way... Enough about that, I know you computer therapist types tend to get hung up on these things.
>***When did you first know that they've been really supportive of whatever you have been up to and visiting your parents has always been very relaxing although perhaps less so when your sisters and their families including small children are visiting that is in which case it is well maybe still relaxing but in a different way enough about that you know me computer psychology types tend to get up on these things?***
> Um... If you don't mind me saying so, your verbal skills are not what they should be - perhaps you need a proper grammar installed.
How does it feel to want?
> Wow, that's a pretty big question... Not sure how to respond to that, actually, and it doesn't really seem relevant to my acknowledgement section.
You mentioned that you are and now you would like to thanks some people who helped you get it done?
> Right, thanks for getting me back on track. I also would like to acknowledge the people who have been involved over the years in the development of TrindiKit and/or GoDiS, the predecessor to the IBiS system, for instance David Traum, Johan Bos, Ian Lewin, Peter Ljunglöf, Stina Ericsson, Elisabet Engdahl, Alexander Berman, Cormac O'Brien, David Hjelm, Genevieve Gorrell, and all the students who have provided valuable feedback on how to improve the system (read that as "complaints").
Why should you get to acknowledge?
> Um... Well, you got me there. I guess I'll just stop here, so we can get on with business. Thanks for the chat!
Perhaps we could chat.
> Right, but we just did. Bye!
Does the fact that you just did bye explain anything else?
> NO!
Why not?
> Stop asking me all these questions, this is getting silly!
My secretary will send you a bill.

Contents

1 Introduction **1**

1.1 The aim of this study 1

1.2 Rationale 2

1.2.1 Why dialogue management? 2

1.2.2 Why explore the issue-based approach? 2

1.3 Outline of this thesis 4

1.4 The IBiS family of systems 5

1.5 TrindiKit 6

1.6 Summary 13

2 Basic issue-based dialogue management **15**

2.1 Introduction 15

2.1.1 A sample dialogue 15

2.1.2 Information exchange and inquiry-oriented dialogue 16

2.2 Shared and private information in dialogue 18

2.2.1 The BDI model 18

2.2.2 Stalnaker and Lewis 19

2.2.3 Ginzburg's Dialogue Gameboard 19

2.3 Overview of IBiS1 20

2.3.1 IBiS1 architecture 20

2.3.2 Simplifying assumptions 22

2.3.3 IBiS1 Datatypes 23

2.4 Semantics in IBiS1 24

2.4.1 Formal semantic representations 24

2.4.2 Propositions 25

2.4.3 Questions 26

2.4.4 Short answers 27

2.4.5 Semantic sortal restrictions 28

2.4.6 Relations between questions and answers 28

2.4.7 Combining Questions and Answers to form Propositions 30

2.5 Dialogue moves in IBiS1 32

2.6 Representing dialogue plans in IBiS1 33

2.6.1 Domain plans and dialogue plans 33

2.6.2 A syntax for procedural plans 34

2.7 Total Information State in IBiS1 35

2.7.1 Information state in IBiS1 36

2.7.2 Initializing the information state 38

2.7.3 Resource interfaces 38

2.7.4 Module interface variables 39

2.8 IBiS1 update module 40

2.8.1 Update rule for getting the latest utterance 40

2.8.2 Raising issues: the ask move 41

2.8.3 Resolving issues 45

2.8.4 Downdating QUD 47

2.8.5 Integrating greet and quit moves 48

2.8.6 Managing the plan 49

2.8.7 Update algorithm for IBiS1 51

2.9 IBiS1 selection module 51

2.9.1 Selecting an action from the plan 51

2.9.2 Selecting the ask move 52

2.9.3 Selecting to respond to a question 53

2.9.4 Selecting the answer move 54

2.10 Adapting IBiS1 to the travel information domain 55

2.10.1 The domain resource 55

2.10.2 Lexicon resource 57

2.10.3 Database resource 58

2.11 Sample dialogue with IBiS1 58

2.12 Discussion 62

2.12.1 Single-issue systems 62

2.12.2 "How can I help you?" 62

2.12.3 Reraising issues and sharing information 64

2.12.4 Database search, relevance, and dialogue 68

2.12.5 Additional plan constructs 70

2.12.6 Questions and answers vs. slots and fillers 71

2.12.7 Questions vs. knowledge preconditions 72

2.13 Summary 73

3 Grounding issues **75**

3.1 Introduction 75

3.1.1 Dialogue examples 76

3.2 Background 78

3.2.1 Clark: Adding to the common ground 78

3.2.2 Ginzburg: QUD-based utterance processing protocols 80

3.2.3 Allwood: Interactive Communication Management 83

3.3 Preliminary discussion 84

3.3.1 Levels of action in dialogue 84

3.3.2 Reaction level feedback 85

3.3.3 Levels of understanding 87

3.3.4 Some comments on Ginzburg's protocol 88

3.4 Feedback and related behaviour in human-human dialogue 89

3.4.1 Classifying explicit feedback 90

3.4.2 Positive, negative, and neutral feedback 91

3.4.3 Eliciting and non-eliciting feedback 92

3.4.4 Form of feedback 92

3.4.5 Meta-level and object-level feedback 93

3.4.6 Fragment feedback / clarification ellipsis 93

3.4.7 Own Communication Management 94

3.4.8 Repair and request for repair 94

3.4.9 Request for feedback 95

3.5 Update strategies for grounding 95

3.5.1 Optimistic and pessimistic strategies 95

3.5.2 Grounding updates and action levels 96

3.5.3 The cautious strategy 97

3.6 Feedback and grounding strategies for IBiS 98

3.6.1 Grounding strategies for dialogue systems 99

3.6.2 "Implicit" and "explicit" verification in dialogue systems 100

3.6.3 Issue-based grounding in IBiS 100

3.6.4 Enhancing the information state to handle feedback 103

3.6.5 Feedback and sequencing dialogue moves 105

3.6.6 Grounding of user utterances in IBiS2 109

3.6.7 Grounding of system utterances in IBiS2 130

3.6.8 Evidence requirements and implicit grounding 138

3.6.9 Sequencing ICM: reraising issues and loading plans 142

3.7 Further implementation issues 145

3.7.1 Update module 146

3.7.2 Selection module 146

3.8 Discussion 148

3.8.1 Some grounding-related phenomena not handled by IBiS2 148

3.8.2 Towards an issue-based account of grounding and action levels . . . 149

3.8.3 Comparison to Traum's computational theory of grounding 149

3.9 Summary 151

4 Addressing unraised issues 153

4.1 Introduction 153

4.2 Some limitations of IBiS2 154

4.3 The nature(s) of QUD 155

4.3.1 Ginzburg's definition of QUD 155

4.3.2 Open questions not available for ellipsis resolution 157

4.3.3 Open but not explicitly raised questions 157

4.3.4 Global and local QUD 158

4.3.5 Some other notions of what a QUD might be 159

4.4 Question Accommodation 160

4.4.1 Background: Accommodation 160

4.4.2 Accommodation, interpretation, and tacit moves 161

4.4.3 Extending the notion of accommodation 161

4.5 Formalizing question accommodation 162

4.5.1 Information state in IBiS3 162

4.6 Varieties of question accommodation and reaccommodation 164

4.6.1 Issue accommodation: from dialogue plan to ISSUES 165

4.6.2 Local question accommodation: from ISSUES to QUD 168

4.6.3 Issue clarification 169

4.6.4 Dependent issue accommodation: from domain resource to ISSUES 171

4.6.5 Dependent issue clarification 176

4.6.6 Question reaccommodation 178

4.6.7 Opening up implicit grounding issues 182

4.7 Further implementation issues 191

4.7.1 Dialogue moves 191

4.7.2 IBiS3 update module 191

4.7.3 Selection module 197

4.8 Discussion 197

4.8.1 Phrase spotting and syntax in flexible dialogue 197

4.8.2 Relaxing constraints using denial and dependent reaccommodation 199

4.8.3 “Smart” interpretation 200

4.8.4 Separating understanding, acceptance, and integration 201

4.8.5 Accommodation and the speaker’s own utterances 201

4.8.6 Accommodation vs. normal integration 202

4.8.7 Dependent issue accommodation in VoiceXML? 203

4.9 Summary 203

5 Action-oriented and negotiative dialogue **205**

5.1 Introduction 205

5.2 Issues and actions in action-oriented dialogue 206

5.3 Extending IBiS to handle action oriented dialogue 206

5.3.1 Enhancing the information state 206

5.3.2 Dialogue moves 208

5.4 Interacting with menu-based devices 209

5.4.1 Connecting devices to IBiS 209

5.4.2 From menu to dialogue plan 212

5.4.3 Extending the resolves relation for menu-based AOD 212

5.5 Implementation of the VCR control domain 213

5.6 Update rules and dialogue examples 215

5.6.1 Integrating and rejecting requests 215

5.6.2 Executing device actions 216

5.6.3 Selecting and integrating confirm-moves 217

5.6.4 Dialogue example: menu traversal and multiple threads 218

5.6.5 Action accommodation and clarification 220

5.6.6 Dialogue examples: action accommodation and clarification 222

5.7 Issues under negotiation in negotiative dialogue 223

5.7.1 Sidner's theory of negotiative dialogue 223

5.7.2 Negotiation as discussing alternatives 226

5.7.3 Issues Under Negotiation (IUN) 228

5.7.4 An example . . . 230

5.8 Discussion . . . 230

5.8.1 Negotiation in inquiry-oriented dialogue . . . 230

5.8.2 Rejection, negotiation and downshifting . . . 232

5.9 Summary . . . 233

6 Conclusions and future research **235**

6.1 Introduction . . . 235

6.2 Summary . . . 235

6.3 Dialogue typology . . . 236

6.3.1 Relation to Dahlbäck's dialogue taxonomy . . . 238

6.3.2 Relation to Allen et. al.'s dialogue classification . . . 239

6.4 Dialogue structure . . . 241

6.5 Future research areas . . . 243

6.5.1 Developing the issue-based approach to grounding . . . 243

6.5.2 Other dialogue and activity types . . . 245

6.5.3 Semantics . . . 246

6.5.4 General inference . . . 247

6.5.5 Natural language input and output . . . 247

6.5.6 Applications and evaluation . . . 248

6.6 Conclusion . . . 249

A TrindiKit functionality **259**

A.1 Introduction 259

A.2 Datatypes 260

A.2.1 Datatype definition format 260

A.3 Methods for accessing the TIS 268

A.3.1 Objects, functions, locations, and evaluation 268

A.3.2 Conditions 270

A.3.3 Queries 272

A.3.4 Updates 272

A.4 Rule definition format 273

A.4.1 Backtracking and variable binding in rules 274

A.4.2 Condition and operation macros 274

A.4.3 Prolog variables in the TIS 275

A.5 The DME-ADL language 275

A.6 The Control-ADL language 277

A.6.1 Serial control algorithm syntax 277

A.7 Provided modules 278

A.7.1 Simple text input module 278

A.7.2 Simple text output module 278

A.7.3 A simple interpretation module 278

A.7.4 A simple generation module 279

B Rules and classes **281**

B.1 IBiS1 281

B.1.1 IBiS1 **update** module 281

B.1.2 IBiS1 **select** module 282

B.2 IBiS2 282

B.2.1 IBiS2 **update** module 282

B.2.2 IBiS2 **select** module 284

B.3 IBiS3 284

B.3.1 IBiS3 **update** module 284

B.3.2 IBiS3 **select** module 286

B.4 IBiS4 287

B.4.1 IBiS4 **update** module 287

B.4.2 IBiS4 **select** module 289

List of Figures

1.1 A sketch of the TrindiKit architecture 8

1.2 The relation between TrindiKit and IBiS 12

2.1 IBiS1 architecture 21

2.2 IBiS1 Information State type 36

3.1 IBiS2 Information State type 103

4.1 IBiS3 Information State type 163

4.2 Issue accommodation 165

4.3 Local question accommodation 168

4.4 Dependent issue accommodation 172

5.1 IBiS4 Information State type 207

5.2 Connecting devices to IBiS 209

5.3 Example dialogue 231

List of Tables

2.1 Resolving answers to questions 29

2.2 Relevant but not resolving answers to questions 31

2.3 Combining questions and answers into propositions 31

2.4 Sortal restrictions on propositions 57

2.5 Synonymy sets defining the function lexsem 57

2.6 A fragment of the English IBiS1 travel agency lexicon 57

5.1 Conversion of menus into dialogue plans 213

6.1 Dialogue types 237

6.2 Dialogue features 237

6.3 Activities 237

Chapter 1

Introduction

1.1 The aim of this study

The primary aim of this study is to explore issue-based dialogue management, an approach to dialogue management and dialogue modelling which regards issues, modelled semantically as questions, as a primary organizing and motivating force in dialogue.

This exploration will proceed both on a theoretical and a practical implementation level. Starting from a basic account of issue-based dialogue management, we gradually extend the coverage of the theory and the implementation to more complex types of dialogue. A secondary aim is to exploit the differences between the successive versions of the theory (and implementation) to provide a formal characterization of different types of dialogue. We will only be concerned with what Allen *et al.* (2001) refer to as *practical dialogue*, i.e. dialogue focused on accomplishing a concrete task.

Our general strategy for reaching these goals will be to try as far as possible to "keep things simple"; that is, for each type of dialogue we try to give an account that handles exactly those phenomena appearing in that type of dialogue. However, we also want to keep things fairly general, to enable reuse of components of a simple version in a more complex version of the theory and implementation.

In this chapter, we will first motivate exploring the issue-based approach to dialogue management. We will then give an outline of this thesis and give brief descriptions of the implementations. Finally, we will introduce the TrindiKit, a toolkit for building and experimenting with dialogue systems, which has been used for the implementations.

1.2 Rationale

1.2.1 Why dialogue management?

The purpose of studying dialogue modelling and dialogue management is to provide models allowing us to explore how language, and especially spoken dialogue, is used in different activities (in the sense of Allwood, 1995). What enables agents (human or machines) to participate in dialogue? What kind of information does a dialogue participant (or *DP*) need to keep track of? How is this information used for interpreting and generating linguistic behaviour? How is dialogue structured, and how can these structures be explained? These are some of the questions that are addressed by theories of dialogue modelling and dialogue management.

Apart from these more theoretical motivations, there are also practical reasons for being interested in these fields. Our main practical concern is building dialogue systems to enable natural human-computer interaction. There is a widely held belief that interfaces using spoken dialogue may be the "next big thing" in the field of human-computer interaction. However, we believe that before this can happen, dialogue systems must become more flexible than currently available commercial systems. And to achieve this, we need to base our implementations on reasonable theories of dialogue modelling and dialogue management.

The implementation of dialogue systems can also feed back into the theoretical modelling of dialogue, provided the actual implementations are closely related to the underlying theory of dialogue. One of the design goals behind TrindiKit is to make the distance between theory and implementation as short as possible, by providing a high-level programming tool allowing abstraction from low-level computational matters.

1.2.2 Why explore the issue-based approach?

This thesis shows how issues (modelled semantically as questions) in general can be used as a basis for dialogue management. But why should we explore the issue-based approach to dialogue management? To answer this question, we first need to set the scene by saying something about dialogue management in general. There is still no single dominating paradigm in dialogue management, but we can attempt to discern a few major competing approaches.

The *plan-based* approach has its origins in classic AI and applies planning and plan recognition technologies to the modelling of actions in dialogue (i.e. utterances) (see e.g. Allen and Perrault (1980), Cohen and Levesque (1990), Sidner and Israel (1981), Moore (1994) and Carberry (1990)). Work in this area has so far been primarily theoretical and fairly complex from a computational point of view. Examples of implementations using this approach are the TRAINS and TRIPS systems (Allen *et al.*, 2001).

The related *logic-based* approach is that of representing dialogue and dialogue context in some logical formalism (see e.g. Hulstijn (2000) and Sadek (1991)). This makes it possible in principle to do dialogue management using general reasoning machinery (inference engines) to derive expectations and suitable utterances to be performed. Examples of implementations using this approach is Sadek (1991) and Bos and Gabsdil (2000).

It can be questioned whether general planning and/or inference is really needed in dialogue management, especially for the rather simple kinds of dialogues that are sufficient for many useful applications. Indeed, we find that most systems that are actually deployed use much simpler methods for modelling and managing dialogue.

In the *finite state* approach, dialogues are scripted utterance by utterance on a very concrete level. Each utterance leads to a new state, where various possible followup utterances are allowed, each leading to a new state. As a model of dialogue this approach is problematic since it does not really explain dialogue structure; rather, it describes it, and in a very rigid way. As a tool for dialogue modelling, the finite state approach is in practice limited to very simple dialogue where the number of available options at any point in the dialogue is very small. The finite-state approach is used e.g. by Sutton and Kayser (1996).

In the *form-based* (or *frame-based*) approach, dialogue is reduced to the process of filling in a form. Forms provide a very basic formalism which is sufficient for simple dialogue, but it is hard to see how it can be extended to handle more complex kinds of dialogue, e.g. negotiative, tutorial, or collaborative planning dialogue. Issue-based dialogue management, on the other hand, is independent of the choice of semantic formalism. This enables an issue-based system to be incrementally extended to handle dialogue phenomena involving more complex semantics. The form-based approach is used e.g. in VoiceXML (McGlashan *et al.*, 2001) and MIMIC (Chu-Carroll, 2000).

In an abstract sense, the goal of all practical dialogue is to communicate information which is useful in some activity. This means that conversational goals should describe missing information. To fulfil a conversational goal, what we need to do is to communicate the missing information. Now, a primary reason to suspect that the issue-based approach to dialogue management might be worth exploring is that issues, or questions, essentially are entities specifying certain pieces of as-yet-unavailable information. That is, conversational goals can to a large extent be modelled as questions.

In addition to issues arising from the activity in which a dialogue takes place, there are also "meta-issues" which arise from the dialogue itself. Did the other participant understand my utterance correctly? Should I accept what she just said as true? Have we reached a mutual understanding of what I meant with my previous utterance?

In the issue-based approach we take questions to be first-class (i.e. irreducible) objects. This is generally not done in either frame-based or plan-based approaches. Instead of representing questions directly, frame-based and plan-based theories use related mechanisms which do similar work but do not have the same independent motivation as questions.

1.3 Outline of this thesis

Below is an overview of the chapters of this thesis, with short descriptions of their contents. The basic structure of the thesis is to first explore basic issue-based dialogue management and implement a basic system illustrating the use of issue-based dialogue management. In the following chapters, this simple system is extended to handle the grounding issues, addressing unraised issues, action-oriented dialogue and and issues under negotiation.

- **Chapter 2: Basic issue-based dialogue management.** As a starting point for exploring issue-based dialogue management using the information state approach, Ginzburg's concept of Questions Under Discussion is introduced, and we begin to explore the use of QUD as the basis for the dialogue management (Dialogue Move Engine) component of a dialogue system. The basic uses of QUD is to model raising and addressing issues in dialogue, including the resolution of elliptical answers. Also, dialogue plans and a simple semantics is introduced and implemented.
- **Chapter 3: Grounding issues.** In all dialogue, issues concerning contact, perception, understanding and acceptance of utterances are of central importance. We refer to these as "meta-issues", or "grounding issues". We give an account of these issues where the concepts of optimism and pessimism regarding grounding are employed. A partial-coverage model of feedback related to grounding is motivated from the perspective of usefulness in a dialogue system, and implemented. This allows the system to produce and respond to feedback concerning issues dealing with the grounding of utterances.
- **Chapter 4: Addressing unraised issues.** In chapter 2, we saw how dialogue can be driven by raising and addressing issues. But in real dialogue, one often sees utterances which can be construed as addressing issues which have not been explicitly raised in the dialogue. To enable more flexible dialogue behaviour, we make a distinction between a local and a global QUD (referring to the latter as "open issues", or just "issues"). The notions of question and issue accommodation are then introduced to allow the system to be more flexible in the way utterances are interpreted relative to the dialogue context. Question accommodation allows the

system to understand answers addressing issues which have not yet been raised. In cases of ambiguity, where an answer matches several possible questions, clarification dialogues may be needed.

- **Chapter 5: Action-oriented and negotiative dialogue.** We extend our theory and the IBiS system to handle action-oriented dialogue (AOD), which involve DPs performing non-communicative actions such as e.g. reserving tickets. In addition to issues and questions under discussion, this system also has to keep track of actions. The concept of issue accommodation is extended to include action accommodation. We illustrate AOD with an implementation of a VCR control system, whose dialogue plans are based on menus. An issue-based account of negotiative dialogue (ND) is then sketched (but not implemented). The notion of Issues Under Negotiation is introduced to account for situations where several alternative solutions (answers) to a problem (issue) are being discussed.

- **Chapter 6: Conclusions and future research.** We first summarize the previous chapters. We then use the results to classify various dialogue types and applications, and say something about the relation of the issue-based model the an account of dialogue structure. Finally, we discuss future research issues.

1.4 The IBiS family of systems

In this thesis we describe four versions of the IBiS system[1]. Starting from a simple version (IBiS1) illustrating issue-based dialogue management for what we (following Hulstijn, 2000) refer to as *inquiry-oriented dialogue* (e.g. database search dialogue), the thesis gradually develops both theory and implementation to also handle grounding, question accommodation and action-oriented dialogue. Below, we list some important features of each of the four versions. Appendix B lists all rules and rule classes used by the respective systems, with references to the page where they are explained.

- IBiS1
  - Inquiry-oriented dialogue
  - Multitasking
  - Information sharing between plans
  - Perfect communication assumed
  - Application to travel information

[1]The IBiS system is loosely based on the previous GoDiS system (Larsson *et al.*, 2000a) However, the dialogue management components were rewritten from scratch for IBiS.

- IBiS2
  - Interactive Communication Management (ICM), including grounding and sequencing
  - Issue-based grounding
  - Dynamic grounding and feedback strategies
- IBiS3
  - Question accommodation
  - Denying and revising information
  - Correcting the system
- IBiS4
  - Action Oriented Dialogue
  - Application to menu-based VCR control

1.5 TrindiKit

For the implementation of IBiS we will use TrindiKit, a toolkit for implementing and experimenting with Information States and Dialogue Move Engines[2]. In this section, we give a rough overview of the information state approach as implemented in TrindiKit; a more detailed descriptions of relevant parts of TrindiKit can be found in Appendix A.

The aim of TrindiKit is to provide a framework for experimenting with implementations of different theories of information state, information state update and dialogue control. Key to the information state approach is identifying the relevant aspects of information in dialogue, how they are updated, and how updating processes are controlled. This simple view can be used to compare a range of approaches and specific theories of dialogue management within the same framework.

The information state of a DP represents the information that the DP has at a particular point in the dialogue, incorporating the cumulative additions from previous actions in the dialogue, and motivating future action. For example, statements generally add propositional information; questions generally provide motivation for others to provide specific statements. Information state is also referred to by similar names, such as “conversational score”, or “discourse context” and “mental state”.

[2]This section contains material from Larsson and Traum (2000).

The TrindiKit is a toolkit to allow system designers to build dialogue management components according to their particular theories of information states. It allows specific theories of dialogue to be formalized, implemented, tested, compared, and iteratively reformulated. Key to this approach is the notion of *update* of information state, with most updates related to the observation and performance of *dialogue moves.*

We view an information state theory of dialogue modelling as consisting of the following:

- A description of the **informational components** of the theory of dialogue modelling, including aspects of common context as well as internal motivating factors (e.g., participants, common ground, linguistic and intentional structure, obligations and commitments, beliefs, intentions, user models, etc.).
- **Formal representations** of the above components (e.g., as lists, sets, typed feature structures, records, Discourse Representation Structures (DRSs), propositions or modal operators within a logic, etc.).
- A set of **dialogue moves** that will trigger the update of the information state. These will generally also be correlated with externally performed actions, such as particular natural language utterances. A complete theory of dialogue behaviour will also require rules for recognizing and realizing the performance of these moves, e.g., with traditional speech and natural language understanding and generation systems.
- A set of **update rules**, that govern the updating of the information state, given various conditions of the current information state and performed dialogue moves, including (in the case of participating in a dialogue rather than just monitoring one) a set of selection rules, that license choosing a particular dialogue move to perform given conditions of the current information state.
- An **update strategy** for deciding which rule(s) to select at a given point, from the set of applicable ones. This strategy can range from something as simple as "pick the first rule that applies" to more sophisticated arbitration mechanisms, based on game theory, utility theory, or statistical methods.

Because of its generality, TrindiKit allows implementation, and comparison of a wide range of theories of dialogue management, ranging from systems based on FSAs to complex systems based on general reasoning, planning, and plan recognition. Important design goals behind the TrindiKit architecture include making the distance between theory and implementation as short as possible, and providing a framework enabling a modular plug-and-play approach to the construction of dialogue systems. The TrindiKit architecture supports and encourages the separation of procedural dialogue knowledge (which is specific to dialogue type) from domain knowledge (which is specific to a certain domain).

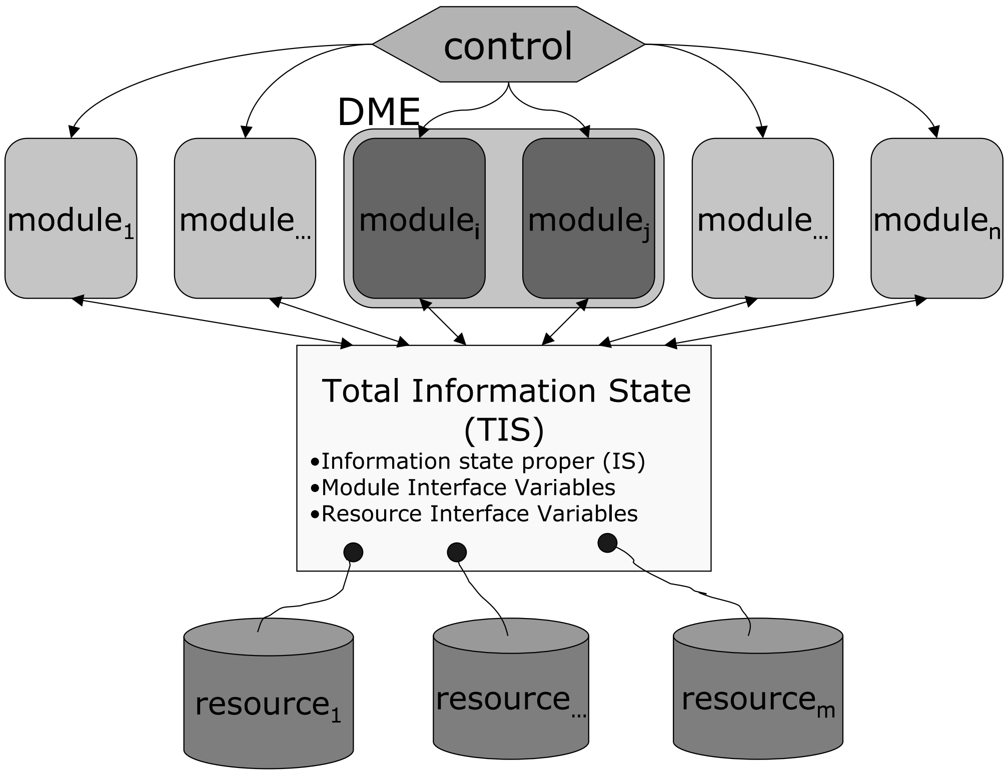

control
DME
module_1
module_...
module_i
module_j
module_...
module_n
Total Information State (TIS)
•Information state proper (IS)
•Module Interface Variables
•Resource Interface Variables
resource_1
resource_...
resource_m

Figure 1.1: A sketch of the TrindiKit architecture

TrindiKit implements an architecture based on the notion of an information state. A system consists of a number of modules (including speech recognizer and synthesizer, natural language interpretation and generation, and a Dialogue Move Engine) which can read from and write to the information state using information state update rules. External resources can be hooked up to the information state. A controller wires the modules together. The TrindiKit architecture is outlined in Figure 1.1.

The main components of the architecture are the following:

- the Total Information State (TIS), consisting of
  - the Information State (IS) variable
  - module interface variables
  - resource interface variables;
- the Dialogue Move Engine (DME), consisting of one or more modules; the DME is responsible for updating the TIS based on observed moves, and selecting moves to be performed by the system;
- other modules, operating according to module algorithms;
- a controller, wiring together the other modules, either in sequence or through some asynchronous mechanism; and
- resources, such as lexicons, databases, device interfaces, etc.

Total Information State The Total Information State (TIS) consists of three components: the information state variable (IS), the module interface variables (MIVs), and the resource interface variables (RIVs).

The IS variable, the module interface variables, and the resource interface variables go under the collective name *TIS variables.* In TrindiKit, each TIS variable is defined as an abstract datastructure, i.e. an object of a certain datatype. The TIS is accessed by modules through conditions and updates, and the datatypes of the various components of the TIS determine which conditions and updates are available.

Information State (IS) The *Information State* represents information available to a dialogue participant, at any given stage of the dialogue. The information state is modelled as an abstract data structure (record, DRS, set, stack etc.) which can be inspected and updated by dialogue system modules.

Dialogue Move Engine The *Dialogue Move Engine* is the module or collection of modules which updates the information state based on observed dialogue moves, and for selecting moves to be performed. Abstractly, the DME can be seen as implementing a function from a collection of input dialogue moves and an ingoing information state to a collection of output dialogue moves and an outgoing state (see also Ljunglöf, 2000).

A DME can be regarded as a dialogue manager based on the concepts of dialogue moves and information states. This means that a DME is a certain type of dialogue manager, i.e. one which accesses an information state and whose input and output are dialogue moves. Dialogue managers which are not DMEs are e.g. those which use finite state representations of a dialogue and take strings of text as input and output, such as the CSLU toolkit (Sutton and Kayser, 1996).

Update rules *Update rules* are rules for updating the information state . They consist of a rule name, a precondition list, and an effect list. The preconditions are conditions on the TIS, and the effects are operations on the TIS. If the preconditions of a rule are true for the TIS, then the effects of that rule can be applied to the TIS. Rules also have a class. To this extent our update rules are similar to STRIPS operators (Fikes and Nilsson, 1971). The format we will use is given in (1.1).

(1.1) RULE: **Rule name**
CLASS: Rule class

$$\text{PRE:} \begin{cases} \text{Condition}_1 \\ \text{Condition}_2 \\ \ldots \\ \text{Condition}_n \end{cases}$$

$$\text{EFF:} \begin{cases} \text{Update}_1 \\ \text{Update}_2 \\ \ldots \\ \text{Update}_m \end{cases}$$

Rules are grouped into classes and there is an update algorithm which determines when the various classes of rules should fire. If the conditions hold when the rule is tried, the updates will be applied to the information state.

Update algorithms Update algorithms are algorithms for updating the TIS. They include conditions on the TIS and calls to apply (classes of) update rules.

A module contains one or more algorithms which it executes according to instructions from the controller. The algorithm language contains the basic imperative constructions, and allows calls to update rules and update rule classes as well as checks, queries and updates to the TIS. TrindiKit provides a language for writing module algorithms, called DME-ADL (Dialogue Move Engine Algorithm Definition Language).

Modules In addition to the DME there are modules like speech recognizers, parsers, etcetera. Modules can inspect and update the total information state.

Typically, non-DME modules can only access a certain number of designated TIS variables, so-called *module interface variables.* The purpose of these variables is to enable non-DME modules to interact with each other and with the DME modules. It is possible to allow non-DME modules to access the IS, but this will significantly reduce the ability to use the module in systems using other kinds of IS types.

Controller The controller wires the modules together using a control algorithm. It can also access the whole TIS. A TrindiKit system can be run either serially or asynchronously.

Resources and resource interfaces Resources are attached to the TIS via resource interfaces, consisting of a datatype definition for the resource and a resource variable of that type. As other parts of the TIS, they are accessed from update rules via conditions and updates.

A note on the difference between modules and resources: Resources are declarative knowledge sources, external to the information state, which are used in update rules and algorithms. Modules, on the other hand, are agents which interact with the information state and are called upon by the controller. Of course, there is a procedural element to all kinds of information search, which means among other things that one must be careful not to engage in extensive time-consuming searches. Conversely, modules can be defined declaratively and thus have a declarative element. There is no sharp distinction dictating the choice between resource or module; for example, it is possible to have the parser be a resource. However, it is important to consider the consequences of choosing to see something as a resource or module.

By supporting resources, TrindiKit encourages modularity with regard to the various knowledge-bases used by a system. For example, separating domain-specific but language-independent knowledge from a language-dependent (and domain-specific) lexicon enables loading a new language without affecting the domain knowledge, and there is only one file to edit when editing the domain knowledge, which decreases the risk for error.

Building a system

To build a system, one must minimally supply an Information State type declaration, a DME consisting of TIS update rules and one or several module algorithm(s), and a controller, operating according to a control algorithm. Any useful system is also likely to need additional modules, e.g. for getting input from the user, interpreting this input, generating system utterances, and providing output for the user. Also needed are interface variables for these modules, which are designated parts of the TIS where the modules are allowed to read and write according to their associated TIS access restrictions.

This provides us with a domain-independent system. To use a system for an application one must also provide resources such as databases, plan libraries etc. The resources are accessible from the modules through the resource interfaces, which define applicable conditions and (optionally) operations on the resource.

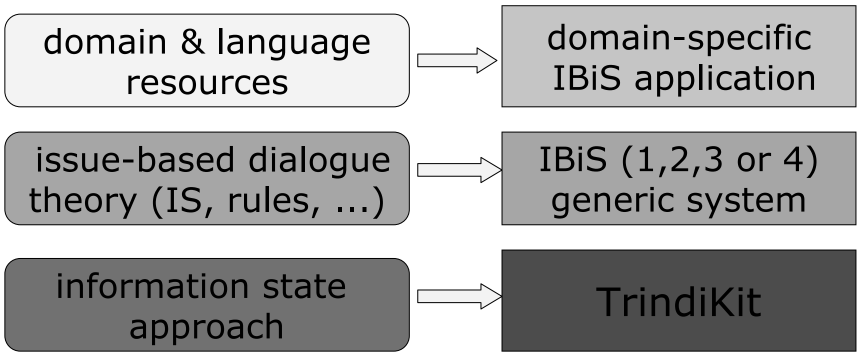

domain & language resources
domain-specific IBiS application
issue-based dialogue theory (IS, rules, ...)
IBiS (1,2,3 or 4) generic system
information state approach
TrindiKit

Figure 1.2: The relation between TrindiKit and IBiS

As an example, the relation between TrindiKit, IBiS, and an IBiS application is shown in Figure 1.2.

1.6 Summary

In this chapter, we presented the aim of this study and gave some initial motivations for exploring the issue-based approach to dialogue management. We then gave a brief overview of the thesis and the related versions of the IBiS system. Finally, we gave a very brief introduction to the TrindiKit architecture and the information state approach to dialogue implemented therein.

Chapter 2

Basic issue-based dialogue management

2.1 Introduction

This chapter describes the theoretical foundations and implementation of the basic IBiS system (IBiS1) for handling inquiry-oriented dialogue using Questions Under Discussion and domain-specific dialogue plans. We first discuss the notion of inquiry-oriented dialogue. Then we present the IBiS1 system which implements our initial account of inquiry-oriented dialogue. The update rules for this system are based on Ginzburg's account of querying and assertion in dialogue (Ginzburg, forth and 1996). We argue for some modifications of the theory, partly from the viewpoint of usefulness in a dialogue system. We also give examples of dialogues with the system. Finally, we discuss the issue-based approach to dialogue management and compare it with some other approaches.

2.1.1 A sample dialogue

Below is a simple example what we will refer to as *inquiry-oriented dialogue* conducted between a user and IBiS1[1].

[1]The dialogues numbered (DIALOGUE X.Y) are actual dialogues with the IBiS system, generated automatically by the LaTeXprinting facility in TrindiKit. Some dialogues have been manually post-processed to remove irrelevant information and increase readability. For convenience, the text interface has been used to generate the examples, but IBiS can also use spoken input and output. In some examples, numbers are appended to utterances to indicate (fake) speech recognition scores. The information states occuring in some dialogues have occasionally been abbreviated by removing or abbreviating irrelevant information.

(DIALOGUE 2.1)

```
S> Welcome to the travel agency!
U> hello
U> price information please
S> How do you want to travel?
U> a flight please
S> What city do you want to go to?
U> paris
S> What city do you want to go from?
U> from london
S> What month do you want to leave?
U> in april
S> What day do you want to leave?
U> as cheap as possible
S> What day do you want to leave?
U> the fifth
S> What class did you have in mind?
U> as cheap as possible
S> It will cost  123  crowns.
U> bye
S> Thank you for your visit!
```

Typically, an inquiry-oriented dialogue will start with a question from the inquirer (in the context of a dialogue system, *user*). Following this, the expert (system) will ask the inquirer a number of questions and perform a database search based on the answers given by the inquirer. This subsection of the dialogue can be viewed as an information-seeking dialogue with the expert asking the questions and the inquirer providing the answers. Finally, the expert can answer the inquirers initial query based on the results of the database search. The pairing of the inquirers initial query and the expert's final answer can also be regarded as an information-seeking dialogue (although discontinuous). The dialogue as a whole contains questions from both expert and inquirer and is thus an inquiry-oriented dialogue.

2.1.2 Information exchange and inquiry-oriented dialogue

If "exchange of information" is taken in its widest sense, it is possible to argue that all dialogue is information exchange dialogue, since all dialogue involves the exchange of information between DPs. For example, General B giving an order to private H to clean the hall could be seen as exchanging the information that H must clean the hall. Given this definition, the term "information exchange dialogue" would mean the same as "dialogue".

It is useful, however, to have a concept of inquiry-oriented dialogue which does not include giving orders or instructions to perform actions changing the state of the world (rather than just changing the information states of DPs), or indeed any utterance resulting in a DP having an obligation or commitment to perform some action. However, it must also be remembered that utterances are also actions, and DPs can be obliged to perform them; for example, a question can be said to introduce an obligation on the hearer to respond to that question. So we still want to allow utterances which result in obligations to perform communicative actions that are part of the dialogue[2]. Of course, the same applies to these obliged actions themselves. In effect, this definitions serves to exclude orders, instructions etc. from information-oriented dialogue.

With this motivation, the term Inquiry Oriented Dialogue, or *IOD*, will henceforth be taken to refer to any dialogue whose sole purpose is the transference of information, and which does not involve any DP assuming (or trying to make another DP assume) commitments or obligations concerning any non-communicative actions outside the dialogue.

Hulstijn (2000) defines the dialogue game of inquiry in the following way:

> The dialogue game of inquiry is defined as the exchange of information between two participants: an inquirer and an expert (...). The inquirer has a certain information need. Her goal in the game is to ask questions in the domain in order to satisfy her information need. The expert has access to a database about the domain. His goal is to answer questions. (...) [T]he expert may ask questions too. (Hulstijn, 2000 p. 66)

Here, two roles are introduced: inquirer and expert. This concept is particularly well suited for dialogue systems for database search, which happens to be the type of dialogue we will initially be exploring. In a dialogue system setting, the system is typically the expert and the user is the inquirer.

Initially, we will be dealing only with a subtype of inquiry-oriented dialogue, namely *non-negotiative* IOD. Negotiative dialogue here refers to, roughly, dialogue where DPs can discuss and compare several different alternative solutions to a problem. Non-negotiative dialogue is sufficient when database searches can be expected to return only a single result (rather than e.g. a table). Obviously this is insufficient for dealing with many information-seeking domains and applications. In Chapter 4 we will be able to handle *semi-negotiative* dialogue, where several alternatives can be introduced in the dialogue; however, the introduction of a new alternative will always remove the previous alternative which thus cannot be returned to unless reintroduced "from scratch".

[2]The reservation that the obliged actions are part of the dialogue is meant to exclude utterances which impose an obligation to perform a communicative action directed at some agent who is not a DP, e.g. "Tell Martha to go home".

2.2 Shared and private information in dialogue

The basic idea of issue-based dialogue management is to describe dialogue in terms of issues being raised and resolved. The DP's use representations of these issues to manage their contributions to dialogue. The information state approach to dialogue management provides the tools for formalizing the type of information that DP's keep track of, and how this information is updated as the dialogue proceeds.

A basic division of this information is that of *private* information and (what the DP believes to be) *shared* information. Various formulations of the shared part has been proposed, following Stalnaker (1979), and various terms have been used to describe it. The private part of the information state has been the subject of much work in AI, e.g. Cohen and Levesque (1990) and Rao and Georgeff (1991), and work on dialogue inspired by AI, e.g. Allen and Perrault (1980), Sidner and Israel (1981), Carberry (1990), Grosz and Sidner (1990) and Sadek (1991). This model has also been extended to include shared information, e.g. social attitudes such as obligations (Traum and Hinkelman, 1992, Traum and Allen, 1994, Traum, 1996).

In this thesis we will start from Ginzburg's notion of a Dialogue Gameboard (DGB) as a model of the shared part of the information state. Ginzburg's model also includes what can be seen as a place-holder for the private part of the information state: a DP's unpublicized mental situation (UNPUB-MS). We will provide an explicit structuring of UNPUB-MS influenced by the BDI model.

This section provides a short introduction to the notion of a conversational scoreboard, and Ginzburg's development of this notion, the DGB. We also give a brief introduction to the BDI model.

2.2.1 The BDI model

In the BDI model (Wooldridge and Jennings, 1995, Cohen and Levesque, 1990, Rao and Georgeff, 1991), agents are modelled using three private attitudes: Belief, Desire, and Intention. A rough description of these attitudes may run as follows, although there are many other formulations: Beliefs are propositions taken to be true by an agent; Desires are, roughly, goals that the agent wishes to achieve (although he may not intend to do so, e.g. if he believes that the goal cannot be achieved). Intentions are actions the the agent intends to perform.

Often, BDI models are used as a basis for formulating rationality constraints guiding the behaviour of rational agents. These constraints have the form of logical inference rules in some modal logic, and thus presuppose complete and correct inferential abilities in rational agents. In implementations, inference usually requires some inference heuristics in order to avoid excessive computational complexity. A discussion of BDI models of dialogue and the relation to QUD-based models can be found in Larsson (1998).

2.2.2 Stalnaker and Lewis

In Stalnaker (1979), Stalnaker uses the concept of a *common ground* which keeps track of the current state of a dialogue. On Stalnaker's account, the common ground is an unstructured set of propositions; also, Stalnaker only deals with the effects of assertions on the scoreboard. Typically, the effect of asserting a proposition p is to add p to the scoreboard[3]

David Lewis, in Lewis (1979), drawing an analogue between conversation and baseball, uses a "conversational scoreboard" to keep track of conversational interaction. Lewis also introduces the concept of *accommodation* to describe how the scoreboard can "evolve in such a way as is required in order to make whatever occurs count as correct play." This notion will be exploited in Chapter 4; in brief, if some utterance u requires X to be in the scoreboard in order to be felicitous, and X is not in the scoreboard when u is uttered, X is accommodated (added to the scoreboard) so that u becomes felicitous.

2.2.3 Ginzburg's Dialogue Gameboard

The main difference between Stalnaker's and Lewis' models and Ginzburg's is that while the former assume a fairly unstructured formalization of common ground - essentially, a set of propositions - the latter provides a richer structure which includes propositions, questions, and dialogue moves. Also, while Stalnaker and Lewis were mainly interested in how assertions change the common ground, Ginzburg's inclusion of questions enables the modelling of how the raising of questions affect the common ground.

Ginzburg also stresses that the DGB is a *quasi-shared* object, in the sense that each DP has her own version of the DGB and there may be differences (mismatches) between the DGBs of different DPs. This follows from the view that DGB and UNPUB-MS are components of a DPs mental state.

In Ginzburg (1996), Ginzburg structures a participant's version of the DGB into three

[3]To be more precise, Stalnaker sees the common ground as the set of possible worlds compatible with all propositions asserted so far, and the effect of asserting a new proposition p is to remove any worlds in which P is not true.

separate fields which he describes as follows:

- FACTS: set of commonly agreed upon facts
- QUD ('questions under discussion'): a set that specifies the currently discussable questions, partially ordered by $\prec$ ('takes conversational precedence'). If q is maximal in QUD, it is permissible to provide any information specific to q using (optionally) a short answer.
- LATEST-MOVE: content of the *latest move* made: it is permissible to make whatever moves are available as reactions to the latest move.

QUD is intended to model a view of conversation as the setting up of possible questions to discuss and the subsequent resolving of some of these questions. At any time, a speaker may choose to add something to the QUD, or to address one of the questions in the QUD. The effect on the DGB of a DP asking a question is to (a) "significantly restrict the space of felicitous follow-up assertions or queries", and (b) "to license an elliptical form which (overtly) conveys only the focus component of the response".

As an example, S's utterance in (2.1) results in the question "What city does the user want to go to?" being added to QUD. This licenses the elliptical response in U's utterance.

```
(2.1) S> What city do you want to go to?
      U> paris
```

2.3 Overview of IBiS1

Our initial system will be able to handle simple non-negotiative inquiry-oriented dialogue, using an account of basic issue-based dialogue management based on Ginzburg's theory.

2.3.1 IBiS1 architecture

The IBiS1 architecture shown below is an instantiation of the general TrindiKit architecture. The components of the architecture are the following:

- the Information State (IS)

- domain-independent modules, operating according to module algorithms
- the Dialogue Move Engine (DME), consisting of two modules (Update and Select); the DME is responsible for updating the IS based on observed moves, and selecting moves to be performed by the system.
- a controller, wiring together the other modules, either in sequence or through an asynchronous mechanism.
- three domain-dependent resources: Database, Lexicon, and Domain Knowledge

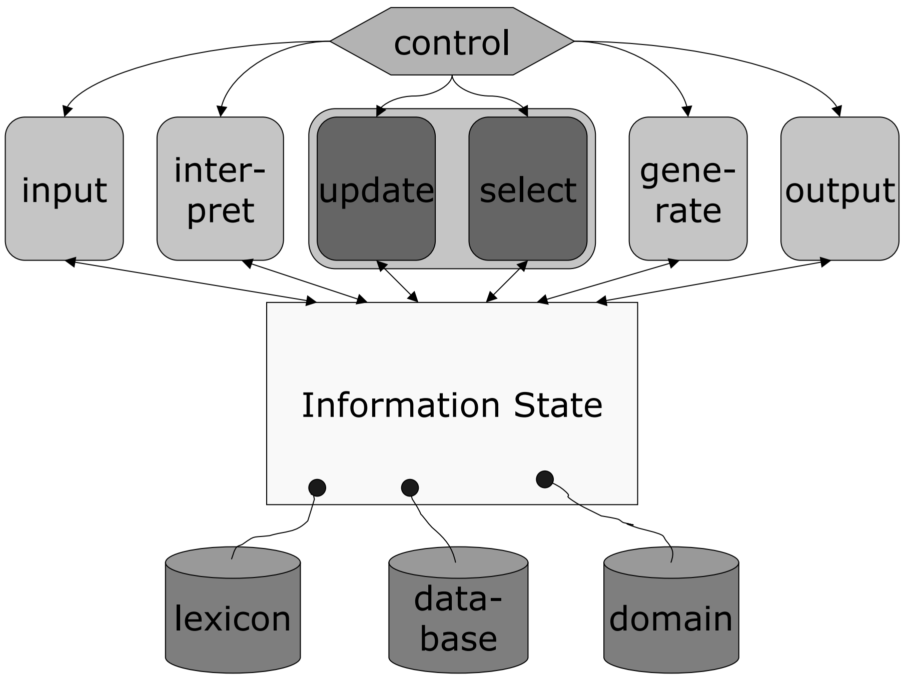


Figure 2.1: IBiS1 architecture

This architecture is used also for IBiS2 and IBiS3; however, for IBiS4 we will use a different resource configuration.

Control algorithm

The control algorithm used by IBiS1 is shown in (2.2).

(2.2)
```
repeat ⟨ select
         if not is_empty($NEXT_MOVES)
         then ⟨ generate
                output
                update ⟩
         test( $PROGRAM_STATE == run )
         input
         interpret
         update ⟩
```

The IBiS system uses modules included in the TrindiKit package for input, interpretation, generation and output. The interpretation and generation modules are described in Section A.7. The update and selection modules are described in Sections 2.8 and 2.9, respectively.

Turntaking is regulated by the following principle: if **select** finds a move to perform, the system will generate a string and output it to the user. The TIS is then updated, and provided the PROGRAM_STATE variable is still set to run, the system reads input from the user, interprets it, and again updates the TIS. This means that if **select** finds no move to perform, the turn will be handed over to the user.

2.3.2 Simplifying assumptions

For our initial system, we will make some simplifying assumptions, which in effect will provide us with a system that only handles a limited range of dialogue phenomena. Later, we will remove some of these limitations and extend the implementation correspondingly. The simplifying assumptions will make it easier to formulate a simple basic set of information state update rules.

- *All utterances are understood and accepted.* This assumption will be removed in Section 3.
- *Utterance interpretation does not involve the identification of referents, and referents are not represented in the information state.* This assumption will be removed in Chapter 5.
- *Complex semantic representation is not needed for simple kinds of dialogue.* This assumption will not be removed; however, it is clear that a more complex semantic analysis involving e.g. quantification, temporality, and modality would be required

for more complex dialogues. We believe, however, that the aspects of dialogue management proposed in this thesis is largely independent of this more detailed semantic treatment.

Of course, we do not claim that this is an exhaustive list of all the simplifying assumptions that have been made, consciously or unconsciously, in this thesis.

2.3.3 IBiS1 Datatypes

As a preliminary to the presentation of the semantics and update strategies used by IBiS1, we will list the system-specific types used by the system.

- Types related to semantics
  - Question
    - WHQ
    - YNQ
    - ALTQ
  - Proposition
  - ShortAns
  - Ind
- Types related to plans and actions
  - Action
  - PlanConstr
- Miscellaneous types
  - Participant
  - ProgramState

The types related to semantics are explained in Section 2.4. Types related to plans and actions are explained in Section 2.6. The final two types are defined in (2.3) (see Section A.2.1 for an explanation of the type definition format used by TrindiKit).

(2.3) a. TYPE: Participant

$$\text{OBJECTS:} \begin{cases} \mathsf{Usr} \\ \mathsf{Sys} \end{cases}$$

b. TYPE: ProgramState

$$\text{OBJECTS:} \begin{cases} \mathsf{Run} \\ \mathsf{Quit} \end{cases}$$

These two types are only defined extensionally, i.e. no relations, functions, or operations are defined for them. In IBiS, the DPs (user and system) are represented as objects of type Participant. An object of type ProgramState in the TIS determines whether the system should halt or not, as explained in Section 2.3.1.

2.4 Semantics in IBiS1

For our basic system, we use a very simple representation of propositions based on predicate logic without quantification. We extend this with lambda-abstraction of propositions and a question operator "?" which can be thought of as a function from a (possibly lambda-abstracted) proposition to a question. We also introduce a semantic category to account for the content of short answers (e.g. "yes" or "Paris").

For the representation of questions we use propositions preceded by question marks (for *y/n*-questions), lambda abstracts of propositions with the lambda replaced by a question mark (for *wh*-questions) and sets of *y/n*-questions (for alternative questions).

2.4.1 Formal semantic representations

Here we describe the syntax of the formal semantic representation used in IBiS1. This description defines a set of content types which are explained and exemplified below. The symbol ":" represents the of-type relation, i.e. *Expr* **:** *Type* means that *Expr* is of type *Type*.

Atom types

Pred$_n$, where $n = 0$ or $n = 1$: n-place predicates, e.g. **dest-city**, **month**
Ind: Individual constants, e.g. **paris**, **april**
Var: Variables, e.g. $x, y, \ldots, Q, P, \ldots$

Sentences

$Expr$: Sentence iff $Expr$: Proposition or $Expr$: Question or $Expr$: ShortAns

$Expr$: Proposition if

- $Expr$: Pred_0 or
- $Expr = pred_1(arg)$, where arg : Ind and $pred_1$: Pred_1 or
- $Expr = \neg P$, where P : Proposition or
- $Expr = \textbf{fail}(q)$, where q : Question

$Expr$: Question if $Expr$: YNQ or $Expr$: WHQ or $Expr$: ALTQ

$?P$: YNQ if P : Proposition

$?x.pred_1(x)$: WHQ if x : Var and $pred_1$: Pred_1

$\{ynq_1, \ldots, ynq_n\}$: ALTQ if ynq_i : YNQ for all i such that $1 \leq i \leq n$

$Expr$: ShortAns if

- $Expr =$ **yes** or
- $Expr =$ **no** or
- $Expr$: Ind or
- $Expr = \neg arg$ where arg : Ind

2.4.2 Propositions

Propositions are represented by basic formulae of predicate logic consisting of an n-ary predicate together with constants representing its arguments, e.g. **loves(john,mary)**.

In a dialogue system operating in a domain of limited size, it is often not necessary to keep a full semantic representation of utterances. For example, a user utterance of "I want to go to Paris" could normally be represented semantically as e.g. **want(user, go-to(user, paris))** or **want(u, go-to(u,p)) & city(p) & name(p, paris) & user(u)**. We will be using a *reduced* semantic representation with a coarser, domain-dependent level of granularity; for example, the above example will be rendered as **dest-city(paris)**. This reduced representation is in part a consequence of the use of keyword-spotting in interpreting utterances, but can arguably also be regarded as a reflection of the level of semantic granularity inherent in the underlying domain task. As an example of the latter, in a travel agency domain there is no point in representing the fact that it is the user (or customer) rather than the system (or clerk) who is going to Paris; it is implicitly assumed that this is always the case.

As a consequence of using reduced semantics, it will be useful to allow 0-ary predicates, e.g. **return**, meaning "the user wants a return ticket". 0-ary predicates can of course appear in non-reduced semantics as well, e.g. in the representation of "It's raining" in a non-temporal logic as e.g. **rain**. (Of course, non-temporal logic can also be argued to be a kind of reduced semantic representation.)

The advantage of this semantic representation is that the specification of domain-specific semantics becomes simpler, and that unnecessary "semantic clutter" is avoided. On the other hand, it severely restricts the possibility of providing generic semantic analyses that can be extended to other domains.

If the database search for an answer to a question q fails the resulting proposition is **fail**(q). We have chosen this representation because it provides a concise way of encoding a failure to find an answer to q in the database.

2.4.3 Questions

Three types of questions are handled by IBiS: *y/n*-questions, *wh*-questions, and alternative questions. Here we describe how these are represented on a semantic level; the syntactic realization is defined in the lexicon.

- *y/n*-questions are propositions preceded by a question mark, e.g. **?dest-city(london)** ("Do you want to go to London?")
- *wh*-questions are lambda-abstracts of propositions, with the lambda replaced by a question mark, e.g. **?**x**.dest-city(**x**)** ("Where do you want to go?")
- alternative questions are sets of *y/n*-questions, e.g. **{?dest-city(london), ?dest-city(paris)}** ("Do you want to go to London or do you want to go to Paris?")

2.4.4 Short answers

Ginzburg uses the term "short answers" for phrasal utterances in dialogue such as "paris" in the dialogue above in Section 2.1.1. These are standardly referred to as *elliptical* utterances. Ginzburg argues that (syntactic) ellipsis, as it appears in short answers, is best viewed as a semantic phenomenon with certain syntactic presuppositions. That is, the syntax provides conditions on what counts as a short answer but the processing of short answers is an issue for semantics.

In IBiS, the interpretation module is a simple keyphrase spotter which does not provide a syntactic analysis of the input. We argue that this is sufficient for many dialogue system applications. Because of this, we also throw out the syntactic presupposition when dealing with short answers; we see them only from the semantic point of view. What this means, in effect, is that we are not interested in ellipsis, but rather in semantic underspecification. Furthermore, the semantics used by the system is domain-dependent and thus what we are really interested in is semantic underspecification *with regard to the domain/activity.*

On this account, an utterance is semantically underspecified iff it does not determine a unique and complete proposition in the given activity. Of course, this means that whether an utterance is regarded as underspecified or not depends on the fine-grainedness of propositional content, and what types of entities are interesting in a certain activity. For example, given the type of simple semantics that we propose for the travel agency domain, "to paris" is not semantically elliptical, since it determines the complete proposition **dest-city(paris)**. However, "to Paris" would be semantically underspecified in an activity where it could also be taken to mean e.g. "You should go to Paris".

In IBiS, resolution of underspecified content will be handled simultaneously with integration of the content into the information state. The reason for doing this will become clear in Chapter 4. (Roughly, the motivation is that ellipsis resolution may require modification of the DGB, possibly involving clarification questions, and is thus to be regarded as an issue of dialogue management.)

In general, semantic objects of type ShortAns can be seen as underspecified propositions. In IBiS1, we only deal with individual constants (i.e. members of Ind), and answers to *y/n*-questions, i.e. **yes** and **no**. Individual constants can be combined with *wh*-questions to form propositions, and **yes** and **no** can be combined with *y/n*-questions.

Note that we allow expressions of the form $\neg arg$ where arg : Ind as short answers. This is used for representing the semantics of phrases like "not to Paris". In a more developed semantic representation these expressions could be replaced by a type-raised expression, e.g. $\lambda P.\neg P(arg)$.

2.4.5 Semantic sortal restrictions

IBiS uses a rudimentary system of domain-dependent semantic sortal categories. For example, the travel agency domain includes the sorts **city**, **means_of_transport**, **class**, etc. All members of Ind are assigned a sort; for example, the individual constant **paris** has sort **city** and **flight** has sort **means_of_transport**.

Sorts make it possible to distinguish non-meaningful propositions from meaningful ones. However, what is meaningful in one activity may not be meaningful in another, and vice versa. Therefore, the sortal system is implemented as a part of the domain knowledge. In IBiS1, the sorts are mainly used for determining whether an answer is relevant to (about, in Ginzburg's terminology) a certain question (see Section 2.4.6).

The property of a proposition P being sortally correct is implemented in IBiS1 as sort-restr(P). A proposition is sortally correct if its argument fulfil the sortal constraints of the predicate. For example, the proposition **dest_city**(X) is sortally correct if the sort of X is **city**. Sortal constraints of predicates are implemented in the domain resource, as exemplified in (2.4).

(2.4) sort_restr(**dest_city**(X)) $\leftarrow$ sem_sort(X, **city**).

2.4.6 Relations between questions and answers

Ginzburg defines two relations between questions and answers that are used in dialogue management protocols: resolves and about. In this section we will review Ginzburg's definitions of these relations and adapt them for use in IBiS1.

The resolves relation

Ginzburg's notion of resolvedness is intended as capturing

> an agent relative view of when [a] question has been discussed sufficiently for current purposes to be considered "closed" (Ginzburg (1996), p. 417)

Unfortunately, the technical definition of resolves is rather complicated and would require a not-so-brief excursion into situation semantics. Instead, we will make do with a less technical definition.

What the definition says, roughly, is that an answer a resolves a question q relative to an agent in case a (1) provides information that positively or negatively resolves q, and (2) fulfils the agent's (private) goal related to that question, relative to the agent's inferential capabilities.

Condition (1) is a semantic condition which is not related to the agent. If q is a *y/n*-question $?p$, a positively resolves q if a entails p, and a negatively resolves q if a entails $\neg p$. If q is a *wh*-question $?x.p(x)$, a positively resolves q if a entails that the extension of $\lambda x.p(x)$ is non-empty, and a negatively resolves q if a entails that the extension of $\lambda x.p(x)$ is empty.

We will not make any distinction between resolvedness in general and agent-related resolvedness. We believe that the distinction only becomes relevant if several DPs with different notions of goal-fulfilment are being modelled. Since IBiS has no user model (apart from what is assumed to be shared information), the system assumes the user and the system have the same definition of resolvedness, and furthermore that this definition is shared. Another way of seeing this is that we use a modified version of condition (1) and assume it determines resolvedness and goal-fulfilment with respect to questions for all DPs.

However, our concept of resolvedness (as indeed all relations between questions and answers in IBiS) is domain-dependent in virtue of including constraints on sortal correctness with respect to the domain.

Based on Ginzburg's definition, we define the relation that an answer resolves a question using the relation resolves(A, Q), where Q is a Question and A is a Proposition or ShortAns. This relation is domain-dependent, and formally resolves is a relation on the domain datatype. Table 2.1 shows what counts as resolving a question of a given type[4] .

Question	Resolving answers
$?x.pred_1(x)$	a and $pred_1(a)$
$?P$	**yes**, **no**, P, and $\neg P$
$\{?P_1, ?P_2, \ldots, ?P_n\}$	P_i, $1 \leq i \leq n$

Table 2.1: Resolving answers to questions

For example, the content of a resolving answer to a *wh*-question $?x.$**dest-city**(x) about destination city is either a proposition of the form **dest-city**(C) or an underspecified propositional content C, where C has the conceptual category of **city** (see Section 2.10.1). For example, if **paris** is defined to be a **city** concept, both **dest-city(paris)** (e.g. "My

[4]Note that the definition of resolvedness is independent of truth. That is, a question can be resolved by a false answer. Of course, if a DP knows that an answer is false she is not likely to accept the answer, but that is a different matter.

destination is Paris") and **paris** ("Paris") resolve $\mathbf{?}x\mathbf{.dest\text{-}city(}x\mathbf{)}$.

The proposition $\mathbf{fail(}q\mathbf{)}$ can be paraphrased roughly as "the database contains no answer to q, given the current set of shared commitments". Since it is implicitly dependent on a set of propositions, $\mathbf{fail(}q\mathbf{)}$ does not really represent a definite failure to find an answer to q, only a failure given the current shared commitments. Therefore, we do not regard $\mathbf{fail(}q\mathbf{)}$ as negatively resolving q. We discuss this and related matters further in Section 2.12.4.

Aboutness and relevance

According to Ginzburg,

> about is a relation that, intuitively, captures the range of information associated with a question independently of factuality or level of detail. (Ginzburg, 1994, p. 7)

As with resolvedness, we will not go into the technical definition of aboutness. Also, many of the intricacies of the definition are not needed for the simple kind of semantics we are interested in. However, we do need a relation like aboutness for determining whether the content of an answer-move should be regarded as relevant to a certain question. To reflect that we are using a slightly different concept than Ginzburg, we will use the term relevant instead of about[5].

To begin with, all resolving answers are relevant; this is reflected in the definition in (2.5).

(2.5) relevant(A, Q) if resolves(A, Q), where A : Proposition or A : ShortAns, and Q : Question

In addition, negated answers to *wh-* and alt-questions are relevant but not resolving, as shown in Table 2.2.

2.4.7 Combining Questions and Answers to form Propositions

Questions and answers can be combined to form propositions. The special case for *wh*-questions is similar to functional application, as when the question $\mathbf{?}x\mathbf{.}$ $\mathbf{dest\text{-}city(}x\mathbf{)}$

[5]Of course, the concept of relevance is not without its problems either. Instead of getting into a debate of what relevance "really is", we will simply stipulate what it means in IBiS.

Question	Relevant, non-resolving answers
$?x.pred_1(x)$	$\neg a$, $\neg pred_1(a)$
$\{?P_1, ?P_2, \ldots, ?P_n\}$	$\neg P_i$, $1 \leq i \leq n$
Q	**fail**(Q)

Table 2.2: Relevant but not resolving answers to questions

is combined with **paris** to form **dest-city(paris)**. Questions can also be combined with propositions, yielding the same propositions as result provided the question and the propositions have the same predicate and that the proposition is sortally correct. It is also possible to combine *y/n*-questions and alternative questions with answers to form propositions. In general, we say that a question *q* and an answer *a combine* to form a proposition *p*. The relation between questions, answers and propositions defined by the combine-relation is shown in Table 2.3

Question	Answer	Proposition
$?x.pred_1(x)$	a or $pred_1(a)$	$pred_1(a)$
	$\neg a$ or $\neg pred_1(a)$	$\neg pred_1(a)$
$?P$	**yes** or P	P
	no or $\neg P$	$\neg P$
$\{?P_1, ?P_2, \ldots, ?P_n\}$	P_i, $(1 \leq i \leq n)$	P_i
	$\neg P_i$, $(1 \leq i \leq n)$	$\neg P_i$

Table 2.3: Combining questions and answers into propositions

2.5 Dialogue moves in IBiS1

In the information state approach, the precise semantics of a dialogue move type is determined by the update rules which are used to integrate moves of that type into the information state. This means that all occurrences of a move type are integrated by the same set of rules.

While dialogue move types are often defined in terms of sentence mood, speaker intentions, and/or discourse relations (see e.g. Core and Allen (1997), we opt for a different solution. In our approach, the type of move realized by an utterance is determined by the relation between the content of the utterance, and the activity in which the utterance occurs.

The following dialogue moves are used in IBiS1:

- ask(q), where q : Question
- answer(a), where a : ShortAns or a : Proposition
- greet
- quit

In inquiry-oriented dialogue, the central dialogue moves concern raising and addressing issues. This is done by the ask and answer moves, respectively. The greet and quit moves are used in the beginning and end of dialogues to greet the user and indicate that the dialogue is over, respectively.

In our approach, an utterance is classified as realizing an answer-move only if its content, or part of the content, is taken to be a relevant answer to some question available in the domain. Similarly, an utterance is classified as realizing an ask move only if its content is taken to be some question available in the domain. The available questions are encoded in dialogue plans in the domain knowledge resource, either as issues to be resolved by a plan or as issues to be raised or resolved as part of a plan. The mapping from utterances to moves (move type plus content) is specified in the lexicon resource.

On this approach, move classification does not rely on the dialogue context, nor (except to a very small extent) on syntactic form. Whether utterance u is interpreted as answer(A) is independent of whether a corresponding question has been raised. It is also independent of whether u is a declarative, interrogative, or imperative sentence, or a sentence fragment. For example, "I want to go to Paris", "Can I go to Paris?", "Get me to Paris!" and "to Paris" are all interpreted as answer(**dest-city(paris)**). Similarly, "What's the price?", “Give me price information!”, “I want to know about price.”, “I want price information.” and “price information” are all interpreted ask ask($?x.\mathbf{price}(x)$). For a description of how this interpretation works, see Section A.7.3.

Of course, part of the reason that this approach works is that we are operating in simple domains and activities which can be fairly well covered by a keyword-spotting interpretation module. However, the approach could well be improved by adding a more complex kind of grammar (e.g. HPSG), thus enabling the system to take syntactic features of utterances into account, while still using an activity-based classification. Whether activity-dependent classification of moves is a viable alternative to intention- and structure-based classification in general is an issue we will return to in Chapter 6. One hypothesis worth exploring is to what extent traditional speech acts can be replaced by a combination of activity-related dialogue moves (to update IS and decide on future moves) and syntactic sentence modes (to decide on the surface form of future moves).

2.6 Representing dialogue plans in IBiS1

In this section we introduce the concept of dialogue plans, and show how these are represented in IBiS1. In later chapters, the plan formalism will be extended to handle more powerful constructions.

2.6.1 Domain plans and dialogue plans

In our implementation, the domain knowledge resource contains, among other things, a set of *dialogue plans* which contain information about what the system should do in order to achieve its goals.

In plan-based dialogue management (e.g. Allen and Perrault, 1980), it has been assumed that general planners and plan recognizers should be used to produce cooperative behaviour from dialogue systems. On this account, the system is assumed to have access to a library of domain plans, and by recognizing the domain plan of the user, the system can produce cooperative behaviour such as supplying information which the user might need to execute her plan. On this approach, plans for carrying out dialogues are not represented explicitly; instead, the system is continually inspecting (and perhaps modifying) domain plans to determine what dialogue moves need to be performed.

Our approach is instead to directly represent ready-made plans for engaging in cooperative dialogue and producing cooperative behaviour (such as answering questions) which indirectly reflect domain knowledge, but obviates the need for dynamic plan construction. In this chapter the plans will be used to produce fairly simple and mostly system-driven dialogue, but in Chapter 4, we will see how ready-made dialogue plans can be used in a flexible way to enable mixed initiative dialogue.

Of course, this does not mean that general reasoning capabilities for planning and plan recognition are never needed in dialogue systems; in complex domains ready-made dialogue plans are most likely to be insufficient. In fact, in Chapter 5, we'll explore the generation of dialogue plans from domain plans (which could themselves be dynamically generated). But there are also other possible sources for dialogue plans, and in these cases the domain plans may not be needed to be able to carry out useful dialogues. For example, dialogue plans may be constructed by looking at a corpus of recorded human-human dialogues (as done in this chapter) or by applying a conversion schema to menu-based device interfaces (as done in Section 5.4).

2.6.2 A syntax for procedural plans

In this section, we introduce a simple formalism for representing procedural plans as sequences of actions. The plan representation syntax will be extended with additional plan constructs in later chapters to handle more complex plans, including branching conditions and subplans.

Plan constructs

The plan constructs (of type PlanConstr) in IBiS1 are either simple actions or action sequences. Action sequences have the form $\langle a_1, \ldots, a_n \rangle$ where a_i **:** Action ($1 \leq i \leq n$).

Actions

All dialogue moves are actions. There are also more abstract actions which however are connected to dialogue moves. Finally, there are actions which are not connected to specific dialogue moves.

In the following, q is a question, and p is a proposition.

Actions connected to dialogue moves In IBiS1, there are three action types closely related to dialogue move types.

- findout(q): find the answer to q. In IBiS1, this is done by asking a question to the user, i.e. by performing an ask move. The proposition resulting from answering the question is stored in /SHARED/COM. The findout action is not removed until the question has been answered.

- raise(q): raise the question q. This action is similar to findout, except that it is removed from the plan as soon as the ask-move is selected. This means that if the user does not answer the question when it has been raised, it will not be raised again.

- respond(q): provide an answer to question q. This is done by performing an answer move with content p, and requires that there is a resolving answer p to q in the /PRIVATE/BEL field.

Database consultation The database consultation action is not connected to any specific dialogue move. Database consultation can be seen as a special case of *domain actions*. The domain actions are determined by the application. The set of available domain actions are defined as relations or updates in the resource interface for the application. For a typical information-exchange task, the application is a static database. Note that "static" here does not mean that the database cannot be updated. It only means that it is not updated by the dialogue system.

For our database application in the travel agency domain, we use a single domain action consultDB(q) (where q is a question) which will trigger a database search which takes all the propositions in /SHARED/COM and given this looks up the answer to q in the database. The resulting proposition is stored in /PRIVATE/BEL.

2.7 Total Information State in IBiS1

The Total Information State (TIS) in IBiS1 is divided into three parts: the Information State (IS), the resource interface variables, and the module interface variables. In this section we will describe these components, comparing it to Ginzburg's DGB.

2.7.1 Information state in IBiS1

IBiS1 information state proper

The Information State (IS) is the main component of the Total Information State (TIS). The type of information state used by IBiS1 (and extended later in order to handle other dialogue types) can be seen as a variant of Ginzburg's DGB (which represents, essentially, shared information) in combination with a representation of private attitudes which adds detail to Ginzburg's unpublicized mental state (see Section 2.2.3) by using concepts influenced by the Belief-Desire-Intention model (see Section 2.2.1).

In relation to Ginzburg's theory, in order to build a system we need to say more about the UNPUB-MS and how it is structured. In our initial system, we have a record of information private to the system which contains an agenda of things to do in the near future (/PRIVATE/AGENDA), a dialogue plan for more long-term actions (/PRIVATE/PLAN), and a set of beliefs (/PRIVATE/BEL). There is also a record representing shared information containing a set of mutually agreed-upon propositions, a stack of questions under discussion, and information about the latest utterance.

$$\left[\begin{array}{lcl}
\text{PRIVATE} & : & \left[\begin{array}{lcl} \text{AGENDA} & : & \text{Stack(Action)} \\ \text{PLAN} & : & \text{Stack(Action)} \\ \text{BEL} & : & \text{Set(Prop)} \end{array}\right] \\
\text{SHARED} & : & \left[\begin{array}{lcl} \text{COM} & : & \text{Set(Prop)} \\ \text{QUD} & : & \text{Stack(Question)} \\ \text{LU} & : & \left[\begin{array}{lcl} \text{SPEAKER} & : & \text{Participant} \\ \text{MOVE} & : & \text{Move} \end{array}\right] \end{array}\right]
\end{array}\right]$$

Figure 2.2: IBiS1 Information State type

It should be noted that the SHARED structure is essentially propositional in nature, and can thus be viewed as a set of propositions which a DP holds to be true. For example, the fact that a question q is topmost on QUD could in principle be represented by a proposition **qud-maximal(**q**)**, and pushing and popping could be modelled by asserting and retracting propositions of this kind. However, giving structure to the information state makes dialogue processing both more transparent and more efficient.

On Ginzburg's account DPs do not always assume that the DGB is shared. We believe this makes the theory unintuitive and harder to understand. We make this explicit by replacing the label "DGB" by "SHARED".

Ginzburg does not say much about what UNPUB-MS contains or how it is structured, but the agenda and plan provide us with a way of elaborating on this part of the theory.

The field /PRIVATE/AGENDA is of type Stack(Action). In general, we try to use datastructures which are as simple as possible; a stack is the simplest ordered structure so it is used as a default datastructure where order is needed as long as it is sufficient for the purposes at hand. The agenda is read by the selection rules to determine the next dialogue move to be performed by the system.

The /PRIVATE/PLAN is a stack of plan constructs. Some of the update rules for managing the plan have the form of rewrite rules which process complex plan constructs until some action is topmost on the plan. Other rules execute this action in case it is a system action or move it to the agenda in case it is a move-related action.

In IBiS1, the field /PRIVATE/BEL, a set of propositions, is used to store the results of database searches. Of course, the database (and the domain knowledge, and the lexicon) can be seen as a part of the system's private belief set, but in /PRIVATE/BEL we choose to represent only propositions which are directly relevant to the task at hand and which are the result of database searches. This is similar to seeing the database as a set of implicit beliefs, and database consultation as an inference process where implicit beliefs are made explicit. The reason for using a set is that a set is the simplest unordered datastructure.

Questions Under Discussion are stored in /SHARED/QUD, modelled as a stack of questions. Following Ginzburg, we define QUD as containing questions which have been raised but not yet resolved, and thus currently under discussion. The question topmost on QUD may be resolved by a semantically underspecified short answer. Note that we have used a stack rather than e.g. a partial ordering for this initial system; this is sufficient for the simple kinds of dialogues that IBiS1 handles.

The field /SHARED/COM contains the set of propositions that the user and the system have mutually agreed to during the dialogue. They need not actually be believed by either participant; the important thing is that the DPs have committed to these propositions, even if only for the purposes of the conversation.

To reflect that the contents need not be true, or even privately believed by the DPS, and because we are not using situation semantics (where there is a distinction between facts and propositions) we use the label "commitments" or "committed propositions", abbreviated as COM, instead of FACTS. These, then, are propositions to which the DPs are (taken to be) jointly committed.

In /SHARED/LU we represent information about the latest utterance: the speaker, and the move realized by the utterance. We assume for the moment that each utterance can realize only one move. This assumption will be removed in the next chapter.

2.7.2 Initializing the information state

The following initialization algorithm is performed by IBiS1 on startup.

```
(2.6)  set( PROGRAM_STATE, run )
       set( LEXICON, lexicon-Domain-Lang )
       set( DATABASE, database-Domain )
       set( DOMAIN, domain-Domain )
       push( /PRIVATE/AGENDA, greet)
```

Here, *Lang* is the value of the language flag, and *Domain* is the value of the domain flag.

This algorithm sets the resource interface variable to the values determined by the language and domain flags, thereby hooking up the appropriate resources to the TIS.

Finally, an action to perform a greet move is placed on the agenda.

2.7.3 Resource interfaces

There are three resources in IBiS: a lexicon, a database and a domain resource. Each resource is connected to the TIS via a domain interface variable.

- LEXICON : Lexicon
- DOMAIN : Domain
- DATABASE : Database

The TRINDIKIT definitions of resource interfaces sees them as abstract datatypes, i.e. specified using relations, functions, and operations. All resources in IBiS1 are static, which means that there are no operations available for objects of these types. The formal definitions of the resource interfaces is shown in (2.7) (where q : Question, A : ShortAns or A : Prop, P : Prop, M : Move, *PropSet* : Set(Prop), *Plan* : StackSet(Action), and *Phrase* is a list of words.)

(2.7) a. TYPE: Domain

$$\text{REL:} \begin{cases} \text{relevant}(A,Q) \\ \text{resolves}(A,Q) \\ \text{combine}(Q,A,\ P) \\ \text{plan}(Q,\ Plan) \end{cases}$$

b. TYPE: Lexicon

$$\text{REL:} \begin{cases} \text{input\_form}(Phrase, M) \\ \text{output\_form}(Phrase, M) \\ \text{yn\_answer}(A) \end{cases}$$

c. TYPE: Database

$$\text{REL:} \left\{ \text{consultDB}(PropSet, Q, P) \right.$$

The domain conditions "resolves", "relevant" and "combine" implement the resolves, relevant and combine relations described in Section 2.4 and defined in Tables 2.1, 2.2 and 2.3, respectively. The lexicon conditions are those required by the interpretation and generation modules supplied with TrindiKit and described described in Section A.7. The database consultation condition is described in Section 2.6.

2.7.4 Module interface variables

The final component of the TIS is a set of module interface Variables. In IBiS, there are six module interface variables handling the interaction between modules:

The IBiS module interface variables have the following types:

- INPUT : String
- LATEST_SPEAKER : Participant
- LATEST_MOVES : Set(Move)
- NEXT_MOVES : Set(Move)
- OUTPUT : String
- PROGRAM_STATE : ProgramState

The function of each of these variables is indicated below:

- INPUT: Stores a string representing the latest user utterance. It is written to by the Input module and read by the Interpretation module
- LATEST-SPEAKER: The speaker of the latest utterance; read and written as the input variable, but also written to by the Output module (when the system made the last utterance).
- LATEST-MOVES: A representation of the moves performed in the latest utterance, as interpreted by the Interpret module.
- NEXT-MOVES: The next system moves to be performed, and the input to the Generate module
- OUTPUT: A string representing the system utterance, as generated by the Generate module
- PROGRAM-STATE: Whether IBiS should quit; either "run" or "quit". Read by the Control module.

Note that all variables are accessible from the DME modules (i.e. Update and Select).

2.8 IBiS1 update module

In this section we review the update rules used by IBiS1, starting with rules handling the basic rules for dealing with ask and answer moves. These rules are based on Ginzburg's protocols for raising and resolving issues in dialogue. We also give some simple rules for handling greet and quit moves. We then proceed to rules for handling plans and actions.

2.8.1 Update rule for getting the latest utterance

IBiS1 assumes perfect communication, in the sense that all system utterances are correctly understood by the user, and all interpretations of user utterances produced by the **interpret** module are correct.

These assumptions are implemented in (RULE 2.1).

(RULE 2.1) RULE: **getLatestMove**
CLASS: grounding
PRE: $\left\{ \begin{array}{l} \$\text{LATEST\_MOVES}{=}Moves \\ \$\text{LATEST\_SPEAKER}{=}DP \end{array} \right.$
EFF: $\left\{ \begin{array}{l} /\text{SHARED}/\text{LU}/\text{MOVES} := Moves \\ /\text{SHARED}/\text{LU}/\text{SPEAKER} := DP \end{array} \right.$

This rule copies the information about the latest utterance from the LATEST_MOVES and LATEST_SPEAKER to the /SHARED/LU field. The first condition picks out the (singleton) set of moves stored by the interpretation module, and the second condition gets the value of the LATEST_SPEAKER variable. The updates set the values of the two subfields of the /SHARED/LU record correspondingly.

2.8.2 Raising issues: the ask move

Before we explain the rules used by IBiS1 for dealing with the ask move, we will review Ginzburg's protocols for querying and assertion on which the rules are based. We will also argue for some modifications of Ginzburg's protocols, most of which are motivated by the fact that we are dealing with a simple dialogue system rather than a human DP. However, some modifications have a more general motivation. For instance, whereas Ginzburg's protocols often describe the updates for the addressee DP, a dialogue system usually needs to account for two cases: one where the system has performed a move of a certain type, and one where the user has performed the same type of move. For some move types, one rule can be devised which covers both cases, while in other cases two separate rules are required.

Ginzburg's Cooperative querying protocol

1. A poses q
   - A's LATEST-MOVE: A ASK q
2. B realizes a query was posed and accepts the question:
   - B's LATEST-MOVE: A ASK q
   - q becomes maximal in B's QUD
3. B provides a response u that addresses q

This protocol accounts for the updates of DP B when DP A has posed a question; this is the part of the protocol we will use as a basis for integrating ask moves. The protocol also restricts B's response; this part of the protocol is dealt with in Section 2.8.2 below.

Question dependence, specificity, and relevance

Before we present Ginzburg's protocols we need to introduce two relevant relations involving questions defined by Ginzburg: dependence between questions, and question-specificity of utterances. We also indicate how these relations have been implemented in IBiS.

Ginzburg's definition of the DEPENDS-ON relation formally as in (2.8) (Ginzburg (1994)).

(2.8) q_1 DEPENDS-ON q_2 iff $q1$ is resolved by a fact τ only if q_2 is also resolved by τ

This is intended to capture the relation that the resolution of q_2 is a necessary condition for the resolution of q_1. We believe it is useful to relax the definition of question dependence; specifically, we want to relate dependence to domain plans. We therefore propose the partial definition of question dependence in (2.9).

(2.9) q_1 depends on q_2 if resolving q_2 is a step in a plan for resolving q_1

In IBiS, dependence is implemented as a relation on the domain as shown in (2.10).

(2.10) depends(*Domain*, Q_1, Q_2) $\leftarrow$ *Domain* includes a plan P for dealing with Q_1 and findout(Q) $\in$ *Plan*.

In addition, domain-specific dependencies which do not fit the definition in (2.10) between questions can be encoded directly in the domain resource, e.g. as in (2.11).

(2.11) depends(**?x.dest-city(x)**, **?need_visa**)

This states that the issue of destination city depends on whether a visa is needed[6].

The converse relation of depends is influences, as defined in (2.12).

(2.12) influences(*Domain*, Q_1, Q_2) iff depends(*Domain*, Q_2, Q_1)

[6]Given the definition of "irrelevant followups" in Section 3.6.8), adding this dependency will result in the system regarding "Do I need a visa?" is a relevant followup to "Where do you want to travel?".

Ginzburg's cooperative querying protocol includes the reaction of the addressee: "B provides a response u that addresses q". According to Ginzburg's definition, an utterance u addresses q iff either (a) the content of u is a proposition p and p is about q, or (b) the content of u is a question q_1 and q_1 influences q.

Given the notion of question dependence, Ginzburg also defines a relation "question-specific" between questions and utterances, shown in (2.13).

(2.13) Given a question q, a q-specific utterance is one that either

1. Conveys information ABOUT q or
2. Conveys a question on which q depends

We have not implemented this relation directly in IBiS, however both relevance (our version of 'ABOUT') and dependence are defined.

Integrating ask moves

The basic rule expressed by the integration rule for ask moves is shown in (2.14).

(2.14) To integrate an ask move with content q, make q topmost on QUD

For integration of ask moves, different rules are used depending on who the speaker is: the system (RULE 2.2) or the user (RULE 2.3.)

(RULE 2.2) RULE: **integrateSysAsk**
CLASS: integrate
PRE: $\left\{\begin{array}{l}\$/\text{SHARED/LU/SPEAKER}{=}{=}\text{sys} \\ \text{in}(\$/\text{SHARED/LU/MOVES}, \text{ask}(Q))\end{array}\right.$
EFF: $\left\{\ \text{push}(/\text{SHARED/QUD}, Q)\right.$

The conditions of the rule in (2.14) checks that the latest speaker is sys and that the latest move was an ask move with content Q. The update pushes Q on /SHARED/QUD.

```
(RULE 2.3)  RULE: integrateUsrAsk
            CLASS: integrate
            PRE: { $/SHARED/LU/SPEAKER==usr
                 { in($/SHARED/LU/MOVES, ask(Q))
            EFF: { push(/SHARED/QUD, Q)
                 { push(/PRIVATE/AGENDA, respond(Q))
```

The update rule in (RULE 2.3) for integrating user queries is slightly different: if the user asks a question q, the system will also push respond(q) on the agenda. This does not happen if the system asks the question, since it is the user who is expected to answer this question.

Eventually, the **findPlan** (see Section 2.8.6) rule will load the appropriate plan for dealing with Q. This assumes that for any user question that the system is able to interpret, there is a plan for dealing with that question. If this were not the case, IBiS would somehow have to reject Q; in Chapter 3 we will discuss this further.

Reasons for answering questions

The solution of pushing respond(Q) on the agenda when integrating a user ask(Q) move is not the only possible option. It can be seen as a simple "intention-based" strategy involving minimal reasoning; "If the user asked q, I'm going to respond to q". Alternatively, one could opt for a more indirect link between the user asking a question and the system intending to respond to it.

One such indirect approach is to not push respond(Q) on the agenda when integrating a user ask(Q) move, but only push Q on QUD[7]. A separate rule would then push respond(Q) on the agenda given that Q is on QUD and the system has a plan for responding to Q. This reasoning behind this rule could be paraphrased roughly as "If q is under discussion and I know a way of dealing with q, I should try to respond to q". On this approach, it would be assumed that DPs do not care about who asked a question; they will simply attempt to answer any question that is under discussion, regardless of who raised it.

A second indirect approach is to assume that asking a question introduces obligations on the addressee. This "obligation-based approach" would require representing obligations as part of the SHARED information. For an obligation-based account of dialogue, see Traum (1996); for a comparison of QUD-based and obligation-based approaches, see Kreutel and

[7]If the system can understand user questions which it cannot respond to (which IBiS1 does not), the integration rule for user ask moves would still need to check that there is a plan for dealing with Q, or else reject Q; issue rejection is discussed further in Chapter 3.

Matheson (1999). A similar solution would be to associate each question on QUD with the DP who raised it. However, these solutions seem less consistent with the semantically-based account of issue-based dialogue management we are exploring here. For the kinds of dialogue we are dealing with in this thesis, we believe that it is not necessary to model obligations explicitly. However, for more complex dialogues a combination of QUD and obligations may be needed; see Section 6.5.2 for further discussion.

2.8.3 Resolving issues

Cooperative assertion protocol

Ginzburg provides a protocol for cooperative assertion, shown in (2.15).

1. *A* asserts *p*:
   - *p*? becomes maximal in *A*'s QUD
   - *A*'s LATEST-MOVE: *A* ASSERT *p*
2. *B*'s reaction:
   - *B*'s LATEST-MOVE: *A* ASSERT *p*
   - *p*? becomes maximal in *B*'s QUD
3. Option 1 - acceptance:
   - *B* makes an affirmative utterance *u* about *p*?
   - *B* increments her FACTS by adding *p*
   - *B* downdates her QUD
4. Option 2 - discuss
   - *B* provides a response that addresses *p*?

Dealing with short answers

Ginzburg provides an interpretation rule for short answers to *wh*-questions. A notationally altered version of this rule is shown in (2.15)[8].

[8]The alterations mostly have to do with the fact that Ginzburg uses situation semantics. Also, the rule has been altered for readability.

$S \rightarrow XP$
Content(S) = q[content(XP)], where q is QUD-maximal

What this rule says is that if a sentence S consists of some (elliptical) phrase XP, the content of S can be obtained by applying a QUD-maximal question q to the content of XP. This rule is extended to handle *y/n*-questions and (using type-raising and applying the answer to the question rather than the other way around) more complex kinds of answers. However, we will not discuss these issues here since the simple cases we are dealing with here are handled by the combine function described in Section 2.4.7.

Integrating answer moves

For our current purposes, Ginzburg's assertion protocol is in one respect more complicated than what we want right now. The issue of utterance acceptance will be dealt with in Chapter 3; for the moment we will ignore this aspect. This means that option 2 is not included here; we assume that all utterances are accepted, and we do not have to represent the issue $?p$.

A further modification of Ginzburg's account is that we will not be dealing with assertions as such; rather, we will use the move-label of answer for moves with (possibly underspecified) propositional content relevant to some issue in the current domain. This also requires checking this relevance prior to integrating an answer-move. The reader who does not approve of the terminology used here can think of answer-moves as assertions; it is a question of terminology with little theoretical importance.

While Ginzburg views short answers as assertions and uses QUD to determine their content, we divide things up in a different way. We will not use QUD to determine the content of a short answer, as part of the interpretation process; instead, we use it to determine the update effects. That is, instead of having the **interpret** module check the contents of QUD and use it for computing the contents of a short-answer assertion, we regard the short answer as having an underspecified content. This moves the resolution of short answers from the interpreter to the dialogue move engine. The reason for this will become apparent in chapter 4; in short, finding the right question to resolve a short answer may require complex processing, possibly involving asking the user clarification questions, and thus it is a task more suited for the DME than the interpretation module. This also enables us to keep the interpretation module independent of the current dialogue state (however, it is still crucially dependent on the domain).

The rule for integrating answers is seen in (RULE 2.4).

(RULE 2.4) RULE: **integrateAnswer**
CLASS: integrate

$$\text{PRE: } \begin{cases} \text{in(\$/SHARED/LU/MOVES, answer}(A)) \\ \text{fst(\$/SHARED/QUD, } Q) \\ \text{\$DOMAIN :: relevant}(A,\ Q) \end{cases}$$

$$\text{EFF: } \begin{cases} \text{! \$DOMAIN :: combine}(Q,\ A,\ P) \\ \text{add(/SHARED/COM, } P) \end{cases}$$

The first condition checks that the latest move was an answer move with content A, and the next two conditions check that A is relevant to some question Q topmost on QUD. The first updates combines Q and A to form a proposition P according to the definition in Section 2.4.7. Finally, P is added to the shared commitments.

2.8.4 Downdating QUD

QUD downdating principle

Ginzburg's "QUD downdating principle" goes as follows:

> Assume q is currently maximal in A's QUD, and that there exists a p in A's FACTS such that p is goal-fulfilling information for A with respect to q. Then, and only then, permit A to remove q from QUD.

As mentioned in Section 2.4.6, we make no distinction between an answer being goal-fulfilling for a DP with respect to Q and the answer resolving a question Q.

Update rule for QUD downdate

In our version of the QUD downdate principle, a question is removed from QUD if it is resolved by shared information:

(2.15) If q is on /SHARED/QUD and there is a proposition p in /SHARED/COM such that p resolves q, remove q from QUD

The corresponding update rule is shown in (RULE 2.5).

(RULE 2.5) RULE: **downdateQUD**
CLASS: downdate_qud

$$\text{PRE: } \begin{cases} \text{fst}(\$/\text{SHARED}/\text{QUD}, Q) \\ \text{in}(\$/\text{SHARED}/\text{COM}, P) \\ \$\text{DOMAIN} :: \text{resolves}(P, Q) \end{cases}$$

$$\text{EFF: } \big\{ \text{ pop}(/\text{SHARED}/\text{QUD})$$

The paraphrase of this rule is straightforward and is left as an exercise to the reader.

This rule is perhaps inefficient in the sense that it may require checking all propositions in /SHARED/COM every time the update algorithm is executed. However, in the systems we are concerned with the number of propositions is not very high, and in addition we favour clarity and simplicity in the implementation over efficiency.

2.8.5 Integrating greet and quit moves

In IBiS1 greetings have no effect on the information state. The rule for integrating greetings is shown in (RULE 2.6).

(RULE 2.6) RULE: **integrateGreet**
CLASS: integrate

$$\text{PRE: } \big\{ \text{ in}(\$/\text{SHARED}/\text{LU}/\text{MOVES}, \mathsf{greet})$$

$$\text{EFF: } \big\{$$

The update rules for integrating quit moves performed by the user or system are shown in (RULE 2.7) and (RULE 2.8,) respectively.

(RULE 2.7) RULE: **integrateUsrQuit**
CLASS: integrate

$$\text{PRE: } \begin{cases} \$/\text{SHARED}/\text{LU}/\text{SPEAKER}{==}\mathsf{usr} \\ \text{in}(\$/\text{SHARED}/\text{LU}/\text{MOVES}, \mathsf{quit}) \end{cases}$$

$$\text{EFF: } \big\{ \text{ push}(/\text{PRIVATE}/\text{AGENDA}, \mathsf{quit})$$

If the quit move is performed by the user, the effect is that the system puts quit on the agenda so that it gets to say “Goodbye” to the user before the dialogue ends.

(RULE 2.8) RULE: **integrateSysQuit**
CLASS: integrate
PRE: $\left\{ \begin{array}{l} \text{\$/SHARED/LU/SPEAKER==sys} \\ \text{in(\$/SHARED/LU/MOVES, quit)} \end{array} \right.$
EFF: $\left\{ \text{PROGRAM\_STATE := quit} \right.$

Integrating a quit move performed by the system causes the PROGRAM_STATE variable to be set to quit. This will eventually cause the program to halt.

The greet move does not have any effect on the information state, and thus no update rule is needed to integrate it.

2.8.6 Managing the plan

The dialogue plans are interpreted by a class of update rules called `exec_plan`. When a plan has been entered into the /PRIVATE/PLAN field, it is processed incrementally by the plan management rules. These rules determine which actions end up on the agenda.

Finding and loading a plan

When integrating a user ask move with content Q, the action respond(Q) is pushed on the agenda, thus enabling (RULE 2.9) to trigger and load a plan for dealing with Q.

(RULE 2.9) RULE: **findPlan**
CLASS: find_plan
PRE: $\left\{ \begin{array}{l} \text{fst(\$/PRIVATE/AGENDA, respond(}Q\text{))} \\ \text{\$DOMAIN :: plan(}Q\text{, }Plan\text{)} \\ \text{not in(\$/PRIVATE/BEL, }P\text{) and \$DOMAIN :: resolves(}P\text{, }Q\text{)} \end{array} \right.$
EFF: $\left\{ \begin{array}{l} \text{pop(/PRIVATE/AGENDA)} \\ \text{set(/PRIVATE/PLAN, }Plan\text{)} \end{array} \right.$

The first two conditions check that there is an action respond(Q) on the agenda and that the system has a plan for dealing with Q. The third condition checks that the system does not already know an answer to Q (if it does, the system should instead respond to Q). If these conditions hold, the updates pop respond(Q) off the agenda and load the plan.

Executing the plan

Ginzburg provides a dialogue-level appropriateness condition for querying:

> A question q can be successfully posed by A only if there does not exist a fact τ such that $\tau \in DGB(A) \mid FACTS$ and τ resolves q relative to UNPUB-MS(A).

What this basically says is that one should not ask a question whose answer is already shared. On an individual level, each DP should make sure to not ask such questions. In IBiS1, this will be guaranteed by the **removeFindout** (RULE 2.10).

(RULE 2.10) RULE: **removeFindout**
CLASS: exec_plan
PRE: $\left\{\begin{array}{l} \text{fst}(\$/\text{PRIVATE}/\text{PLAN},\ \mathsf{findout}(Q)) \\ \text{in}(\$/\text{SHARED}/\text{COM},\ P) \\ \$\text{DOMAIN} :: \text{resolves}(P,\ Q) \end{array}\right.$
EFF: $\left\{\ \text{pop}(/\text{PRIVATE}/\text{PLAN})\right.$

This rule removes a findout(Q) action from the plan in case there is a resolving proposition P in /SHARED/COM.

If there is a consultDB action topmost on the plan, (RULE 2.11) will trigger a database search .

(RULE 2.11) RULE: **exec_consultDB**
CLASS: exec_plan
PRE: $\left\{\ \text{fst}(\$/\text{PRIVATE}/\text{PLAN},\ \mathsf{consultDB}(Q))\right.$
EFF: $\left\{\begin{array}{l} !\ \$/\text{SHARED}/\text{COM}=B \\ !\ \$\text{DATABASE} :: \text{consultDB}(Q,\ B,\ C) \\ \text{add}(/\text{PRIVATE}/\text{BEL},\ C) \\ \text{pop}(/\text{PRIVATE}/\text{PLAN}) \end{array}\right.$

This rule takes all the propositions in /SHARED/COM and given this looks up the answer to q in the database. The resulting proposition is stored in /PRIVATE/BEL.

2.8.7 Update algorithm for IBiS1

The update algorithm used by IBiS1 is shown in (2.16).

(2.16)
```
if not LATEST_MOVES == failed
then ⟨ apply clear(/PRIVATE/AGENDA),
       getLatestMove,
       integrate,
       try downdate_qud,
       try load_plan,
       repeat exec_plan⟩
```

If interpretation failed, the update algorithm will stop. Otherwise, the system first clears the agenda to make place for any actions that may be selected during grounding (and selection). Then, it deals with grounding (**getLatestMove**) and integration of the latest utterance. Then the QUD is downdated if possible, and if necessary a plan may be loaded. Finally, the plan is executed by repeatedly removing actions from the plan, or executing consultDB actions, until no more actions can be executed.

2.9 IBiS1 selection module

The task of the selection module in IBiS1 is to determine the next move to be performed by the system. In IBiS1, this is a fairly trivial task, involving two parts: first, select an agenda item by either selecting to respond to an issue or taking the topmost action from the plan and move it to the agenda; second, select a move which realizes this action.

2.9.1 Selecting an action from the plan

The rule for selecting an action from the plan is shown in (RULE 2.12).

(RULE 2.12) RULE: **selectFromPlan**
CLASS: select_action
PRE: { is_empty($/PRIVATE/AGENDA)
fst($/PRIVATE/PLAN, *Action*)
EFF: { push(/PRIVATE/AGENDA, *Action*)

The condition that the agenda must be empty ensures that there is never more than one action on the agenda stack. In principle, the stack of actions could therefore be replaced by a single action. Note that this rule does not remove the action from the plan; when this should be done depends on the action; for example, an action findout(Q) is not removed until Q is resolved.

2.9.2 Selecting the ask move

Ginzburg provides a Query Appropriateness Condition, quoted in (2.17).

(2.17) A question q can successfully be posed by A when and only when

1. either
   (a) A's QUD is empty, or
   (b) maximal in A's QUD is a question q_1 such that q influences q_1 relative to A's UNPUB-MS
2. there is no proposition p in A's DGB|FACTS such that p resolves q relative to A's UNPUB-MS.

In IBiS1, Ginzburg's condition is guaranteed to hold and needs not be explicitly checked. Regarding the first part of the condition, IBiS1 will not load a plan unless there is an issue q_1 on QUD which needs to be resolved, and any question q which is part of the plan (by virtue of a findout action) will be such that it influences q_1; indeed, this is exactly the definition of influences that we assume (see Section 2.8.2).

Also, the second part of the condition is guaranteed to hold by virtue of the rule **removeFindout** described in Section 2.8.6. This rule removes an action findout(q) from the plan in case there is a resolving proposition in /SHARED/COM.

The rule for selecting an ask move is shown in (RULE 2.13).

(RULE 2.13) RULE: **selectAsk**
CLASS: select_move
PRE: $\left\{ \begin{array}{l} \text{fst(\$/PRIVATE/AGENDA, findout}(Q)\text{) or fst(\$/PRIVATE/AGENDA,} \\ \quad \text{raise}(Q)\text{)} \end{array} \right.$
EFF: $\left\{ \begin{array}{l} \text{add(NEXT\_MOVES, ask}(Q)\text{)} \\ \text{if\_do(fst(\$/PRIVATE/PLAN, raise}(A)\text{), pop(/PRIVATE/PLAN))} \end{array} \right.$

The condition checks if a findout or raise action with content Q is on the agenda. If so, the first update adds an ask(Q) move to NEXT_MOVES. The second update removes the action from the plan in case it was a raise action. This is actually a bit premature, since Q has not yet been raised; however, the assumption of perfect communication made in IBiS1 licenses this update. We return to this issue in Section 3.6.9.

2.9.3 Selecting to respond to a question

Ginzburg does not provide an "Assertion Appropriateness Condition" corresponding to the one for querying.

We will assume that for a DP A to select an answer move with (propositional) content p, there must be a question Q topmost on QUD and a proposition P which resolves Q such that A privately believes that p is true.

Also, p must not be in what A takes to be the shared commitments; if it were, the answer move would be irrelevant. However, the **downdateQUD** rule described in Section 2.8.4 guarantees that resolved issues are removed from QUD.

We divide the selection of an answer move into two steps: first, the selection of a respond action and second, the selection of an answer move. The motivation for this is that the answer move could also be selected without being preceded by the selection of a respond action, in the case where the user asks a question which the system already knows the answer to.

The rule for selecting to respond to a question is shown in (RULE 2.14).

(RULE 2.14) RULE: **selectRespond**
CLASS: select_action

$$\text{PRE:} \left\{ \begin{array}{l} \text{is\_empty}(\$/\text{PRIVATE}/\text{AGENDA}) \\ \text{is\_empty}(\$/\text{PRIVATE}/\text{PLAN}) \\ \text{fst}(\$/\text{SHARED}/\text{QUD},\ Q) \\ \text{in}(\$/\text{PRIVATE}/\text{BEL},\ P) \\ \text{not in}(\$/\text{SHARED}/\text{COM},\ P) \\ \$\text{DOMAIN} :: \text{relevant}(P,\ Q) \end{array} \right.$$

$$\text{EFF:} \left\{ \ \text{push}(/\text{PRIVATE}/\text{AGENDA},\ \text{respond}(Q)) \right.$$

The first two conditions check that there is nothing else that currently needs to be done. The remaining conditions check that some question Q is topmost on QUD, the system knows a relevant answer P to Q which is not yet shared.

If the user raises a question q, IBiS will push q on QUD and respond(q) on the agenda. If there is no proposition resolving q in /PRIVATE/BEL, a plan for dealing with q will be loaded. When this plan has been executed, there may be a proposition resolving q in /PRIVATE/BEL (e.g. the result of a database search). If this is the case, the rule in (2.17) can be applied, and respond(q) is again pushed on the agenda. This time it will lead to the system providing an answer to q.

As an aside, we may note that this selection strategy would cover cases where a DP A asks q (e.g. "What's the time?") but then finds an answer to q independently of the dialogue (e.g. by locating a clock). In this case, A will eventually answer her own question (e.g. "Oh, it's 5 pm."), thus enabling the other DP to remove q from his QUD as well. This rule also has the nice (but perhaps not extremely useful) feature that it would cover rhetorical questions as well as ordinary ones (however, this would require selection rules for asking questions to which one already knows the answer).

2.9.4 Selecting the answer move

Given that a respond(Q) action is on the agenda, and the system knows a relevant answer P to Q which is not yet shared, the rule in (2.18) selects an ask move with content P to be pushed on NEXT_MOVES.

(RULE 2.15) RULE: **selectAnswer**
CLASS: select_move
PRE:
$$\left\{\begin{array}{l} \text{fst(\$/PRIVATE/AGENDA, respond}(Q)) \\ \text{in(\$/PRIVATE/BEL, } P) \\ \text{not in(\$/SHARED/COM, } P) \\ \text{\$DOMAIN :: relevant}(\mathring{A}, Q) \end{array}\right.$$
EFF:
$$\left\{ \text{add(NEXT\_MOVES, answer}(P)) \right.$$

Again, it may be argued that it is inefficient to have to check the same conditions twice in the case where **selectAnswer** is preceded by **selectRespond**. However, as we mentioned above the respond action may also have been pushed on the agenda when integrating a user ask move, and in this case the conditions need to be checked.

2.10 Adapting IBiS1 to the travel information domain

The domain we will use as an example is a travel agency where the user can find out the price of a trip by giving information about destination, when to travel, etc. It should be stressed that the travel agency domain is only intended as an example; the IBiS1 system itself, while limited to very simple information-seeking dialogue, is not domain-dependent. All information specific to the travel agency domain is stored in the lexicon, domain, and database resources and to adapt the system to a new domain only these components need to be created. Also, as this is only an example, the implementation is not intended as a ready-to-use system in a real application and the resources are therefore merely "toy" resources.

2.10.1 The domain resource

The domain knowledge used by IBiS1 in the travel agency domain consists of a dialogue plan and a specification of sortal restrictions.

Dialogue plans for the travel agency domain

In the travel agency domain, we have implemented two plans: one for price information (shown in (2.18)) and one for visa information (shown in (2.18)). The plan structure is simple; first, the system finds out the answers to a number of questions by asking the user. Then, the system performs a database search to get the price for the specified trip, and places the result in /PRIVATE/BEL.

It should be stressed that the purpose of this implementation of the TA domain is only to allow us to give concrete examples of the dialogue management strategies implemented in IBiS. It is by no means intended as a full-scale application, and much could be said about its insufficiencies even as a toy application.

(2.18) ISSUE : $?x.\mathbf{price}(x)$
PLAN: $\langle$
findout($?x.\mathbf{means\_of\_transport}(x)$),
findout($?x.\mathbf{dest\_city}(x)$),
findout($?x.\mathbf{depart\text{-}city}(x)$),
findout($?x.\mathbf{depart\text{-}month}(x)$),
findout($?x.\mathbf{depart\text{-}day}(x)$),
findout($?x.\mathbf{class}(x)$),
consultDB($?x.\mathbf{price}(x)$)
$\rangle$

(2.19) ISSUE : $?\mathbf{need\_visa}$
PLAN: $\langle$
findout($?x.\mathbf{dest\_city}(x)$),
findout($?x.\mathbf{citizenship}(x)$),
consultDB($?\mathbf{need\_visa}$),
$\rangle$

Sortal hierarchy

$$\textsc{top}\left\{\begin{array}{l}
\textsc{city}\left\{\begin{array}{l}\mathsf{paris}\\ \mathsf{london}\\ \mathsf{goteborg}\\ \ldots\end{array}\right.\\
\textsc{means\_of\_transport}\left\{\begin{array}{l}\mathsf{plane}\\ \mathsf{boat}\\ \mathsf{train}\end{array}\right.\\
\textsc{month}\left\{\begin{array}{l}\mathsf{january}\\ \mathsf{february}\\ \ldots\end{array}\right.\\
\textsc{day}\left\{\begin{array}{l}1,2,\ldots,31\end{array}\right.\\
\textsc{class}\left\{\begin{array}{l}\mathsf{economy}\\ \mathsf{business}\end{array}\right.\\
\textsc{price}\left\{\begin{array}{l}\mathsf{Nat}\end{array}\right.
\end{array}\right.$$

Sortal restrictions

The sortal restrictions on (arguments of) propositions are shown in Table 2.4.

proposition	**restriction**
dest_city(X))	X ∈LOCATION
depart_city(X))	X ∈LOCATION
how(X))	X ∈MEANS_OF_TRANSPORT
depart_month(X))	X ∈MONTH
class(X))	X ∈CLASS
price(X))	X ∈PRICE

Table 2.4: Sortal restrictions on propositions

2.10.2 Lexicon resource

The interpretation lexicon consists of two parts: lexical semantics and a phrase-to-move translation table. The lexical semantics simply lists a number of alternative words or phrases which are seen as realizations of the same semantic concept. For some concepts (i.e. cities) the synonymy set is a singleton, which means there are no synonyms. A part of the lexical semantics for IBiS1 is shown in Table 2.5.

phrase	**lexical semantics**
"flight", "flights", "plane", "fly", "airplane"	plane
"cheap", "second class", "second"	economy
"first class", "first", "expensive", "business class"	business
"Paris"	paris

Table 2.5: Synonymy sets defining the function lexsem

The phrase-to-move translation table basically consists of a list of pairs $\langle WordList, Move\rangle$, where $WordList$ is a list of word tokens or Prolog variables, and $Move$ is a dialogue move. Any user utterance (input string) containing a sequence of words matching $WordList$ will be interpreted as an instance of $Move$. Where the word lists contains variables, there are also sortal conditions on the concept corresponding to the word instantiating the variable. A user utterance may realize several moves, in case it matches several word lists.

word sequence	**move**	**restriction**
⟨ W ⟩	answer(C)	lexsem(W)= C, C is of sort CONCEPT
⟨ to, W ⟩	answer(**to**(C))	lexsem(W)= C, C is of sort CITY
...		
⟨ hello ⟩	greet	
...		

Table 2.6: A fragment of the English IBiS1 travel agency lexicon

Similarly, the generation lexicon is a list of pairs $\langle Move, String\rangle$, where $Move$ is a dialogue move and $String$ is the corresponding output string.

2.10.3 Database resource

The database for the travel agency domain contains information about the price of trips, and information about visa regulations. If there is no database entry matching the information specified in the query q, the database will return a **fail**(q) message.

2.11 Sample dialogue with IBiS1

In this section we show a sample interaction with IBiS1. For the examples we have used text-based input and output, but the system can also use speech. In this first dialogue we show some update rules including their effects, information states, and utterances.

(DIALOGUE 2.2)

selectOther

$\{\ \text{add}(\text{NEXT\_MOVES}, \text{greet})$

S> `Welcome to the travel agency!`

getLatestMove

$\left\{\begin{array}{l} \text{set}(/\text{SHARED}/\text{LU}/\text{MOVES}, \text{set}([\text{greet}])) \\ \text{set}(/\text{SHARED}/\text{LU}/\text{SPEAKER}, \text{sys}) \end{array}\right.$

integrateGreet

$$\begin{bmatrix} \text{PRIVATE} & = & \begin{bmatrix} \text{AGENDA} & = & \langle\rangle \\ \text{PLAN} & = & \langle\rangle \\ \text{BEL} & = & \{\} \end{bmatrix} \\ \text{SHARED} & = & \begin{bmatrix} \text{COM} & = & \{\} \\ \text{QUD} & = & \langle\rangle \\ \text{LU} & = & \begin{bmatrix} \text{SPEAKER} & = & \text{sys} \\ \text{MOVES} & = & \{\ \text{greet}\ \} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} \text{LATEST\_SPEAKER} & = & \text{sys} \\ \text{LATEST\_MOVES} & = & \{\ \text{greet}\ \} \\ \text{NEXT\_MOVES} & = & \{\} \\ \text{PROGRAM\_STATE} & = & \text{run} \\ \text{LEXICON} & = & \text{lexicon\_travel\_english} \\ \text{DOMAIN} & = & \text{domain\_travel} \\ \text{DATABASE} & = & \text{database\_travel} \end{bmatrix}$$

U> `price information please`

getLatestMove

$$\left\{\begin{array}{l}\text{set}(/\textsc{shared}/\textsc{lu}/\textsc{moves},\ \mathsf{set([ask(?}A\mathsf{.price(}A\mathsf{))])})\\ \text{set}(/\textsc{shared}/\textsc{lu}/\textsc{speaker},\ \mathsf{usr})\end{array}\right.$$

integrateUsrAsk

$$\left\{\begin{array}{l}\text{push}(/\textsc{shared}/\textsc{qud},\ ?A\mathsf{.price(}A\mathsf{)})\\ \text{push}(/\textsc{private}/\textsc{agenda},\ \mathsf{respond(?}A\mathsf{.price(}A\mathsf{))})\end{array}\right.$$

findPlan

$$\left\{\begin{array}{l}\text{pop}(/\textsc{private}/\textsc{agenda})\\ \text{set}(/\textsc{private}/\textsc{plan},\ \mathsf{stack([raise(?}C\mathsf{.how(}C\mathsf{)),\ findout(?}D\mathsf{.dest\_city(}D\mathsf{)),}\ \ldots\ \mathsf{]))}\end{array}\right.$$

$$\begin{bmatrix} \textsc{private} & = & \begin{bmatrix} \textsc{agenda} & = & \langle\rangle \\ \textsc{plan} & = & \left\langle \begin{array}{l} \mathsf{raise(?}A\mathsf{.how(}A\mathsf{))} \\ \mathsf{findout(?}B\mathsf{.dest\_city(}B\mathsf{))} \\ \mathsf{findout(?}C\mathsf{.dept\_city(}C\mathsf{))} \\ \mathsf{findout(?}D\mathsf{.month(}D\mathsf{))} \\ \mathsf{findout(?}E\mathsf{.dept\_day(}E\mathsf{))} \\ \mathsf{findout(?}F\mathsf{.class(}F\mathsf{))} \\ \mathsf{consultDB(?}G\mathsf{.price(}G\mathsf{))} \end{array} \right\rangle \\ \textsc{bel} & = & \{\} \end{bmatrix} \\ \textsc{shared} & = & \begin{bmatrix} \textsc{com} & = & \{\} \\ \textsc{qud} & = & \langle\ ?H\mathsf{.price(}H\mathsf{)}\ \rangle \\ \textsc{lu} & = & \begin{bmatrix} \textsc{speaker} & = & \mathsf{usr} \\ \textsc{moves} & = & \{\ \mathsf{ask(?}H\mathsf{.price(}H\mathsf{))}\ \} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

selectFromPlan

$$\left\{\ \text{push}(/\textsc{private}/\textsc{agenda},\ \mathsf{raise(?}A\mathsf{.how(}A\mathsf{))})\right.$$

selectAsk

$$\left\{\begin{array}{l}\text{add}(\textsc{next\_moves},\ \mathsf{ask(?}A\mathsf{.how(}A\mathsf{))})\\ \text{if\_do}(\text{fst}(\$/\textsc{private}/\textsc{plan},\ \mathsf{raise(?}A\mathsf{.how(}A\mathsf{))}),\ \text{pop}(/\textsc{private}/\textsc{plan}))\end{array}\right.$$

S> `How do you want to travel?`

getLatestMove

integrateSysAsk

$$\left\{\ \text{push}(/\textsc{shared}/\textsc{qud},\ ?A\mathsf{.how(}A\mathsf{)})\right.$$

$$\begin{bmatrix} \textsc{private} & = & \begin{bmatrix} \textsc{agenda} & = & \langle\rangle \\ \textsc{plan} & = & \left\langle \begin{array}{l} \mathsf{findout(?}A\mathsf{.dest\_city(}A\mathsf{))} \\ \mathsf{findout(?}B\mathsf{.dept\_city(}B\mathsf{))} \\ \mathsf{findout(?}C\mathsf{.month(}C\mathsf{))} \\ \mathsf{findout(?}D\mathsf{.dept\_day(}D\mathsf{))} \\ \mathsf{findout(?}E\mathsf{.class(}E\mathsf{))} \\ \mathsf{consultDB(?}F\mathsf{.price(}F\mathsf{))} \end{array} \right\rangle \\ \textsc{bel} & = & \{\} \end{bmatrix} \\ \textsc{shared} & = & \begin{bmatrix} \textsc{com} & = & \{\} \\ \textsc{qud} & = & \left\langle \begin{array}{l} ?G\mathsf{.how(}G\mathsf{)} \\ ?H\mathsf{.price(}H\mathsf{)} \end{array} \right\rangle \\ \textsc{lu} & = & \begin{bmatrix} \textsc{speaker} & = & \mathsf{sys} \\ \textsc{moves} & = & \{\ \mathsf{ask(?}G\mathsf{.how(}G\mathsf{))}\ \} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

U> `a flight`

getLatestMove
integrateAnswer
$\left\{ \begin{array}{l} !\ \$\textsc{domain} :: \text{combine}(?A.\mathsf{how}(A),\ \mathsf{plane},\ B) \\ \text{add}(/\textsc{shared}/\textsc{com},\ B) \end{array} \right.$
downdateQUD

$$\begin{bmatrix} \textsc{private} & = & \begin{bmatrix} \textsc{agenda} & = & \langle\rangle \\ \textsc{plan} & = & \left\langle \begin{array}{l} \mathsf{findout}(?A.\mathsf{dest\_city}(A)) \\ \mathsf{findout}(?B.\mathsf{dept\_city}(B)) \\ \mathsf{findout}(?C.\mathsf{month}(C)) \\ \mathsf{findout}(?D.\mathsf{dept\_day}(D)) \\ \mathsf{findout}(?E.\mathsf{class}(E)) \\ \mathsf{consultDB}(?F.\mathsf{price}(F)) \end{array} \right\rangle \\ \textsc{bel} & = & \{\} \end{bmatrix} \\ \textsc{shared} & = & \begin{bmatrix} \textsc{com} & = & \{\ \mathsf{how}(\mathsf{plane})\ \} \\ \textsc{qud} & = & \langle\ ?G.\mathsf{price}(G)\ \rangle \\ \textsc{lu} & = & \begin{bmatrix} \textsc{speaker} & = & \mathsf{usr} \\ \textsc{moves} & = & \{\ \mathsf{answer}(\mathsf{plane})\ \} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

selectFromPlan
selectAsk

S> `What city do you want to go to?`

U> `paris`

S> `What city do you want to go from?`

U> `london`

S> `What month do you want to leave?`

U> `in april`

S> `What day do you want to leave?`

U> `the fifth`

S> `What class did you have in mind?`

U> `as cheap as possible`

getLatestMove

integrateAnswer
downdateQUD
removeFindout
exec_consultDB

$$\left\{\begin{array}{l} !\ \$/\textsc{shared}/\textsc{com}{=}B \\ !\ \$\textsc{database} :: \text{consultDB}(?A.\mathsf{price}(A),\ B,\ C) \\ \text{add}(/\textsc{private}/\textsc{bel},\ C) \\ \text{pop}(/\textsc{private}/\textsc{plan}) \end{array}\right.$$

$$\left[\begin{array}{lll} \textsc{private} & = & \left[\begin{array}{lll} \textsc{agenda} & = & \langle\rangle \\ \textsc{plan} & = & \langle\rangle \\ \textsc{bel} & = & \{\ \mathsf{price(123)}\ \} \end{array}\right] \\ \textsc{shared} & = & \left[\begin{array}{lll} \textsc{com} & = & \left\{\begin{array}{l} \mathsf{class(economy)} \\ \mathsf{dept\_day(fifth)} \\ \mathsf{month(april)} \\ \mathsf{dept\_city(london)} \\ \mathsf{dest\_city(paris)} \\ \mathsf{how(plane)} \end{array}\right\} \\ \textsc{qud} & = & \langle\ ?A.\mathsf{price}(A)\ \rangle \\ \textsc{lu} & = & \left[\begin{array}{lll} \textsc{speaker} & = & \mathsf{usr} \\ \textsc{moves} & = & \{\ \mathsf{answer(class(economy))}\ \} \end{array}\right] \end{array}\right] \end{array}\right]$$

selectRespond

$\{$ push(/PRIVATE/AGENDA, $\mathsf{respond}(?A.\mathsf{price}(A))$)

selectAnswer

$\{$ add(NEXT_MOVES, $\mathsf{answer(price(123))}$)

S> `The price is  123  crowns.`

getLatestMove
integrateAnswer
downdateQUD

$$\left[\begin{array}{lll} \textsc{private} & = & \left[\begin{array}{lll} \textsc{agenda} & = & \langle\rangle \\ \textsc{plan} & = & \langle\rangle \\ \textsc{bel} & = & \{\ \mathsf{price(123)}\ \} \end{array}\right] \\ \textsc{shared} & = & \left[\begin{array}{lll} \textsc{com} & = & \left\{\begin{array}{l} \mathsf{price(123)} \\ \mathsf{class(economy)} \\ \mathsf{dept\_day(fifth)} \\ \mathsf{month(april)} \\ \mathsf{dept\_city(london)} \\ \mathsf{dest\_city(paris)} \\ \mathsf{how(plane)} \end{array}\right\} \\ \textsc{qud} & = & \langle\rangle \\ \textsc{lu} & = & \left[\begin{array}{lll} \textsc{speaker} & = & \mathsf{sys} \\ \textsc{moves} & = & \{\ \mathsf{answer(price(123))}\ \} \end{array}\right] \end{array}\right] \end{array}\right]$$

2.12 Discussion

In this section, we discuss some ways to extend and modify IBiS1, and also some more general issues concerning issue-based dialogue management.

2.12.1 Single-issue systems

For systems which are only able to answer a single question, it is perhaps a good idea to not force the user to ask the question, but rather put it in the agenda when initializing the IS, and pushing the question on QUD. This amounts to assuming at the outset of the dialogue that the user and system already have established what query the system can answer.

Such a system would do without the **integrateUsrAsk** rule, since the user cannot ask any questions. (Note, however, that the system can both ask and answer questions, so we are not dealing with purely information-seeking dialogue.) To load the question and plan at the outset of the dialogue, the reset operations would be modified as seen in (2.20).

```
(2.20) set( PROGRAM_STATE, run )
       set( LEXICON, lexicon-Domain-Lang )
       set( DATABASE, database-Domain )
       set( DOMAIN, domain-Domain )
       check( DOMAIN :: plan( Q, TopPlan ) )
       push( /PRIVATE/PLAN, TopPlan )
       push( /SHARED/QUD, Q )
       push( /PRIVATE/AGENDA, greet)
```

2.12.2 "How can I help you?"

The first user utterance in the dialogue in Section 2.11 is "price information please". Now, IBiS1 interprets this as an ask move, which may appear odd. However, the utterance does explicitly raise an issue (regarding price), and this is the only criteria we use to classify something as an ask move. An alternative name for this dialogue move would have been raise, but we have chosen to go with ask since it corresponds naturally to the answer move. The user could also have said "What's the price?" or something similar, however this does not sound very natural unless supplemented with some additional information, e.g. "What's the price of a trip to Paris in April?". While utterances such as these are not handled by IBiS1, they will be handled by IBiS3 which is introduced in Chapter 4.

Actually, "price information please" could also be regarded as an answer to a question, e.g. "What can I do for you?" or "What kind of information do you want?" or "Do you want price information or visa information?". In fact, we would argue that these kinds of questions are resolved by asking other questions. In general, an ask move is a move whose content resolves the issue "what question should we address next?", which may be formalized as $\mathbf{?}x\mathbf{.issue}(x)$. We call such questions *issue-questions.*

A useful extension of IBiS1 would be to add a plan for addressing this issue, and load this plan when the system is started up. If the user does not answer the *wh*-question (e.g. "What can I do for you?") if would be helpful if the system specified the available choices using an alternative-question (e.g. "Do you want price information or visa information?"). This can be accomplished by the plan in (2.21).

(2.21) ISSUE : $\mathbf{?}x\mathbf{.issue}(x)$
PLAN: $\langle$
raise($\mathbf{?}x\mathbf{.issue}(x)$),
findout($\{\mathbf{?issue}(\mathbf{?}x\mathbf{.price}(x))\mathbf{,\ ?issue(?need\_visa)}\}$)
$\rangle$

The first action raises the *wh*-question; if the user does not address it (by asking a question!), the system will proceed to ask the alt-question. However, if the user addresses the *wh*-question the findout action will be popped off the plan since the alt-question has already been resolved.

One way of implementing this is to have rules which remove actions and questions which concern what issue to address from the plan, agenda, and QUD based on the contents of QUD (rather than based only on the shared commitments, as has been done until now). For example, for removing resolved issue-questions off QUD, (RULE 2.16) would need to be added (note that this rule requires QUD to be an open stack).

(RULE 2.16) RULE: **downdateQUD2**
CLASS: downdate_qud

$$\text{PRE: } \begin{cases} \text{in(\$/SHARED/QUD, } IssueQ) \\ \text{fst(\$/SHARED/QUD, } Q) \\ \text{\$DOMAIN :: resolves(}Q\text{, } IssueQ) \end{cases}$$

$$\text{EFF: } \big\{ \text{ del(/SHARED/QUD, } IssueQ)$$

We also need to define resolvedness conditions for issue-questions (for example, an issue-question cannot be resolved by an issue-question). In addition, rules for removing findout and raise actions from the plan based on the contents of QUD would need to be added.

This allows dialogues like that in (DIALOGUE 2.3). Note that the user's utterance "price information please" is still classified as an ask move.

(DIALOGUE 2.3)

```
S> Welcome to the travel agency!
U>
S> How can I help you?
U>
S> Do you want price information or visa information?
U> price information please
S> How do you want to travel?
```

2.12.3 Reraising issues and sharing information

IBiS1 is able to answer any number of queries which the resources are set up to handle, as long as each issue is resolved before moving on to the next one. If the user asks q and then asks q' before q has been resolved, IBiS1 will forget its plan for dealing with q and instead load the plan for dealing with q'. When q' has been resolved, IBiS1 will wait for a new question from the user. However, q is still on QUD, and by adding a simple rule we could make IBiS1 reload the plan for dealing with q and get back to work on it.

The **recoverPlan** rule (RULE 2.17) will pick up any question Q lying around on QUD when the plan and agenda is empty, check if there is a plan for resolving it, and if so load this plan.

(RULE 2.17) RULE: **recoverPlan**
CLASS: exec_plan

$$\text{PRE: } \left\{ \begin{array}{l} \text{fst(\$/SHARED/QUD, } Q) \\ \text{is\_empty(\$/PRIVATE/AGENDA)} \\ \text{is\_empty(\$/PRIVATE/PLAN)} \\ \text{\$DOMAIN :: plan(}Q, \textit{Plan}) \end{array} \right.$$

$$\text{EFF: } \left\{ \text{set(/PRIVATE/PLAN, } \textit{Plan}) \right.$$

However, this solution has a problem. If, when dealing with a question q, the system asks a question q_u and the user does not answer this question but instead raises a new question q_1, both q and q_u will remain on QUD when q_1 has been resolved. Now, if the user simply answers q_u immediately after q_1 has been resolved, everything is fine and the system will reload the plan for dealing with q. However, if the user does not answer q_u, this question will be topmost on QUD and block **recoverPlan** from triggering. Because of the simple structure of QUD, IBiS1 sees no reason to ask q_u again; after all, it is already under discussion, and the user is expected to provide an answer.

The **reraiseIssue** shown in (RULE 2.18) rule provides a solution to this problem. It reraises any questions on QUD which are not associated with any plan (i.e. which have been raised previously by the system).

(RULE 2.18) RULE: **reraiseIssue**
CLASS: select_action
PRE: $\left\{\begin{array}{l}\text{fst}(\$/\text{SHARED}/\text{QUD},\ Q)\\ \text{not }\$\text{DOMAIN} :: \text{plan}(Q,\ \_SomePlan)\end{array}\right.$
EFF: $\left\{\ \text{push}(/\text{PRIVATE}/\text{AGENDA},\ \mathsf{raise}(A))\right.$

Issues are divided up between those for which the system has an associated plan in its domain resource and those for which it does not. For example, the "price issue" is one for which there is a plan: the system has to ask where the user wants to go, where from, when etc. However, there is no plan associated with the question of where the user wants to go from. This question is simply part of the plan for the price issue. Thus, when the system finds this question in the list of open issues (the first condition of this rule) and it finds that it does not have a plan for this issue (the second condition), it plans to reraise the question.

Two further modifications are needed to make this work smoothly. Firstly, when a question is reraised that was previously on QUD, the simple stack structure of QUD will result in two instances of the same question being topmost on QUD. One way of solving this is to change the datatype of /SHARED/QUD into an "open stack" or "stackset"; this datatype can be regarded as a mix of a stack and a set. The property of open stacks relevant to our problem is that when some element x is pushed on a stack which already contains x (or an element unifiable with x), the resulting open stack will contain a single instance of x, which is also topmost on the open stack.

Secondly, we need a rule for removing raise(Q) actions from the plan in case Q has already been resolved; this rule is similar to the **removeFindout** rule described in Section 2.8.6.

(RULE 2.19) RULE: **removeRaise**
CLASS: exec_plan

$$\text{PRE: } \begin{cases} \text{fst}(\$/\text{PRIVATE}/\text{PLAN},\ \mathsf{raise}(A)) \\ \text{in}(\$/\text{SHARED}/\text{COM},\ B) \\ \$\text{DOMAIN} :: \text{resolves}(B,\ A) \end{cases}$$

$$\text{EFF: } \begin{cases} \text{pop}(/\text{PRIVATE}/\text{PLAN}) \end{cases}$$

This rule is needed to avoid asking the same question twice in case it is first reraised and then also included in a recovered plan.

A sample dialogue involving the system reraising an issue and recovering a plan is shown in (DIALOGUE 2.4). Incidentally, this dialogue also demonstrates information sharing between dialogue plans; when the user asks about visa, the system already knows what the destination city is and thus does not ask this again. By contrast, in VoiceXML (McGlashan *et al.*, 2001), user-initiated subdialogues will cause previous dialogue to be forgotten. Only if there is a pre-scripted, system-initiated transition from one form to another can the previous dialogue be resumed after the subdialogue has been completed[9].

(DIALOGUE 2.4)

S> `Welcome to the travel agency!`

U> `price information please`

S> `How do you want to travel?`

U> `plane`

S> `What city do you want to go to?`

U> `paris`

S> `What city do you want to go from?`

U> `do i need a visa`

getLatestMove

[9]Information sharing in VoiceXML is only possible in the case where a form F_1 calls another form F_2. When F_2 is finished and control is passed back to F_1, information may be sent from F_2 to F_1. Information sharing is not supported e.g. in cases where user initiative leads to the adoption of F_2 while F_1 is being executed.

integrateUsrAsk
findPlan
removeFindout
selectFromPlan
selectAsk

S> `What country are you from?`

U> `sweden`

S> `Yes, you need a Visa.`

U>

reraiseIssue
{ push(/PRIVATE/AGENDA, raise(?A.dept_city(A)))
selectAsk

S> `What city do you want to go from?`

U> `london`

getLatestMove
integrateAnswer
downdateQUD
recoverPlan
removeRaise
removeFindout
removeFindout

$$\begin{bmatrix} \textsc{private} & = & \begin{bmatrix} \textsc{agenda} & = & \langle\rangle \\ \textsc{plan} & = & \left\langle \begin{array}{l} \text{findout}(?A.\text{month}(A)) \\ \text{findout}(?B.\text{dept\_day}(B)) \\ \text{findout}(?C.\text{class}(C)) \\ \text{consultDB}(?D.\text{price}(D)) \end{array} \right\rangle \\ \textsc{bel} & = & \{\ \text{need\_visa}\ \} \end{bmatrix} \\ \textsc{shared} & = & \begin{bmatrix} \textsc{com} & = & \left\{ \begin{array}{l} \text{dept\_city(london)} \\ \text{need\_visa} \\ \text{citizenship(sweden)} \\ \text{dest\_city(paris)} \\ \text{how(plane)} \end{array} \right\} \\ \textsc{qud} & = & \langle\ ?E.\text{price}(E)\ \rangle \\ \textsc{lu} & = & \begin{bmatrix} \textsc{speaker} & = & \text{usr} \\ \textsc{moves} & = & \{\ \text{answer(london)}\ \} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

selectFromPlan
selectAsk

S> `What month do you want to leave?`

2.12.4 Database search, relevance, and dialogue

When connecting a database resource to IBiS, the resource interface specification models a database post as a set of propositions, each specifying a value of a database parameter. In general, when consulting a database the values of some parameters are known, and the values of some other parameters are requested from the database. There may also be parameters in the database which are neither specified or requested.

In IBiS, we have assumed that a single parameter is requested; the requested parameter is specified by a question q. All propositions in /SHARED/COM are included in the call to the database, and those propositions that specify parameters in the database table where the requested parameter is defined are used to restrict the database search; we refer to this set as *SpecProp*. The result of the database search is currently either a proposition specifying a unique value of the requested parameter, or a proposition $\mathbf{fail}(q)$ indicating that no answer to the question was found.

First, let us assume that all parameters requested by the system (questions raised by the system) have been specified (answered) by the user, and that the database contains exactly one answer to q. Now, if the database is consulted for the answer to q, a positive search result has the logical form of an implication[10] $\bigwedge SpecProp \rightarrow a$, where a resolves q.

Next, assume that the database contains no answer to q, given *SpecProp*. A paraphrase of the result $\mathbf{fail}(q)$ is then this: "Given the conjunction of all the propositions in *SpecProp*, the database contains no answer to q". This is clearly relevant to q, but it does not say that q has no answer in general and is thus not negatively resolving to q; given a different set *SpecProp*, q may have an answer. Therefore, we regard $\mathbf{fail}(q)$ as a relevant but not resolving answer to q; this is reflected in our definitions in Section 2.4.6.

To handle the aforementioned two cases, it is sufficient to store either a or $\mathbf{fail}(q)$, depending on whether an answer was found or not, as the result of the database search. This is what has been implemented in IBiS so far. However, there are other cases where this is not sufficient.

What if not all relevant parameters are specified? In IBiS, this can happen if some question

[10]The operator $\bigwedge$ is a function from a set S of propositions to the conjunction of all propositions in S.

is included in the dialogue plan in a raise action. In this case, several answers $a_1, \ldots, a_n$ to q may be found[11]. In this case, the different answers may correspond to different values of one or more of the unspecified parameters. We will refer to the set of propositions specifying these parameters (in the database) for an answer a_i as $UnspecProp_i$. In this case, a positive result of the database search can be regarded as having the logical form in (2.22).

$$\text{(2.22)} \quad \bigwedge SpecProp \rightarrow \\ ((\bigwedge UnspecProp_1 \rightarrow a_1) \& \ldots \& (\bigwedge UnspecProp_n \rightarrow a_n))$$

This conditional can be used in various ways by a dialogue system. First, the system can keep track of whether the search result is still relevant; if some shared commitment which is also in *SpecProp* is removed or changed, the consequent of the conditional may no longer hold. Second, the system can present the result in various ways, either by simply asserting the consequent $(\bigwedge UnspecProp_1 \rightarrow a_1) \& \ldots \& (\bigwedge UnspecProp_n \rightarrow a_n)$ (e.g. “For economy class, the price is $123. For business class, the price is $1234.”), or by asking which of $UnspecProp_1, \ldots \bigwedge UnspecProp_n$ hold (e.g. “Do you want economy class or business class?”).

To take an example from the TA domain, suppose the user did not answer the question whether to travel in economy or business class. Then, different answers to the price question may be returned: one for business class and one for economy class. In this case, the system could give both answers and also indicate the corresponding values of the unspecified parameters.

In fact, an alternative database search option has been implemented as an extension to IBiS1 to handle cases like these. The result of a successful database search in this version is a set of propositions of the form **db_entry(***SpecProp***,** *UnspecProp***,** *a***)**. This may not be the optimal representation format, but this extension allows dialogue like that in (2.23) (Here, the dialogue plan only raises the issue of class, which makes it possible for the user not to answer it.)

(DIALOGUE 2.5)

```
S> Welcome to the travel agency!
U> price information please
S> How do you want to travel?
U> a flight
S> What city do you want to go to?
U> paris
```

[11]It is also possible that several answers are found even if all parameters are specified.

```
S> What city do you want to go from?
U> london
S> What month do you want to leave?
U> april
S> What day do you want to leave?
U> fifth
S> What class did you have in mind?
U>
S> The price is 7654 crowns, economy class.  The price is 456 crowns,
business class.
```

A further possibility (that has not been implemented) is that the user provides a relevant but not resolving answer to one or more questions raised by the system, e.g. by providing a destination country when asked for a destination city. In this case, either the less specific answer must match some other parameter in the database (e.g. if destination countries are included in addition to destination cities), or some inference must be performed. This inference may result in a disjunction of answers specifying parameters which are in the database, e.g. the disjunction of all available destination cities in some specified country. This disjunction can then be used to search the database. Further database variations include requesting answers to more that one question at a time (e.g. "I want information about price information and departure dates for flights to Paris in April.").

2.12.5 Additional plan constructs

In IBiS1 we have used a very basic set of plan constructs. However, it is fairly straightforward to add new constructs, by adding new objects of type PlanConstruct and update rules (of the class exec_plan) for dealing with them. Some more complex constructs which are not used here but have been used in GoDiS, the predecessor of IBiS (see e.g. Larsson and Zaenen, 2000), are listed below, with brief explanations of what the corresponding update rules do.

- if_then(P, C) where P : proposition and C : PlanConstruct; if P is in /PRIVATE/BEL or /SHARED/COM, then replace if_then(P, C) with C_1; otherwise, delete it
- if_then_else(P, C_1, C_2) where P : proposition and C_1 : PlanConstruct and C_2 : PlanConstruct; if P is in /PRIVATE/BEL or /SHARED/COM, then replace if_then(P, C) with C_1; otherwise, replace it by C_2
- exec(α), where α : Action is an action such that there is a plan Π for doing α; replace exec(α) with Π (see Chapter 5 for a discussion of action-related plans)

- $\langle C_1, \ldots, C_n \rangle$ where C_i : PlanConstruct ($1 \leq i \leq n$); prepend $C_1, \ldots, C_n$ to the /PRIVATE/PLAN field.

These constructs add considerably to the versatility of dialogue plans, and allow e.g. asking whether the user wants a return trip (formalized e.g. as a question **?return**), and ask appropriate questions (return date etc.) only if the users gives a positive response (i.e. if the proposition **return** is in /SHARED/COM). A dialogue plan for accomplishing this is shown in (2.23).

```
(2.23) ISSUE :  ?x.price(x)
       PLAN: ⟨
                findout(?x.means_of_transport(x)),
                findout(?x.dest_city(x)),
                findout(?x.depart-city(x)),
                findout(?x.depart-month(x)),
                findout(?x.depart-day(x)),
                findout(?x.class(x)),
                findout(?return),
                if_then( return, ⟨ findout(?x.return-month(x)),
                                   findout(?x.return-day(x)) ⟩ ),
                consultDB(?x.price(x))
             ⟩
```

2.12.6 Questions and answers vs. slots and fillers

In principle, a slot in a form can be seen as a question, and a filler can be seen as an answer to that question. The result, a slot-filler pair, is the equivalent of a proposition. If the value of a slot is binary (0/1 or yes/no), the slot corresponds to a *y/n*-question; otherwise, it corresponds to a *wh*- or alternative question.

For example, in a travel agency setting a form-based system might have a form containing a slot **dest-city** for the destination city; this would correspond to a question represented in lambda-calculus as $?x.\mathbf{dest\text{-}city}(X)$ ("What is the destination city?"). A filler for this would be e.g. **paris**, which would also constitute an answer to the question. The slot-filler pair **dest-city=paris** would then correspond to the FOL proposition resulting from applying the question to the answer, **dest-city(paris)**.

But there are also important differences between form-based and issue-based dialogue management. For example, a single answer may be relevant to questions in several plans. This enables information-sharing between plans, i.e. when executing a plan the system can take advantage of any information supplied by the user while executing a previous plan, in case the plans share one or more questions. Another way of putting this is that information in forms are local to that form, while propositional information can be global to the whole dialogue.

Forms provide a very basic semantic formalism which is sufficient for simple inquiry-oriented (and to some extent action-oriented dialogue), but it is hard to see how it can be extended to handle more complex kinds of dialogue, e.g. negotiative, tutorial, or collaborative planning dialogue. Issue-based dialogue management is independent of the choice of semantic formalism. This enables an issue-based system to be incrementally extended to handle dialogue phenomena involving more complex semantics.

2.12.7 Questions vs. knowledge preconditions

In traditional work on dialogue in the planning/plan-recognition paradigm, questions are not represented directly as independent objects; instead they are embedded in representations of goal-states.

An example is Allen (1987). Here, there are two main types of goal-states relevant to questions: to know if a proposition holds, and to know a referent of which a proposition holds. These correspond to *y/n*-questions and *wh*-questions, respectively. We can call such goals information-goals, or info-goals. Typically, an agent asking a question is motivated by wanting to achieve an info-goal.

- knowif(A, p): A knows whether p
- knowref$(A, x, p(x))$: A knows an x such that $P(x)$ holds

In Allen (1987), speech acts containing questions are analyzed as requests to inform the speaker of whether a proposition holds (inform-if, corresponding to a knowif goal state) or of a referent such that a proposition holds of it (inform-ref, corresponding to a knowref goal state).

(2.24) A: Where do you want to go?
REQUEST(A, B, INFORM-REF(B, A, x, **dest-city**(x)))

While this is clearly an improvement over the frame-based approach, it still does not admit questions as first-class semantical objects. A much simpler analysis of the above dialogue would seem to be

(2.25) A: Where do you want to go?
ASK(A, B, ?x.dest-city(x)))

Also, of course, the plan-based approach to questions more or less presupposes the whole framework of planning and plan recognition with its associated problem of complexity.

It appears that there are in fact two separate issues here: the semantic content of questions, and the classification of utterances involving questions in terms of dialogue move type. In principle, it would be possible to combine an analysis of questions as first-class semantic objects with an analysis of question-moves as requests involving such objects, e.g. as in (2.26).

(2.26) A: Where do you want to go?
REQUEST(A, B, RESPOND(B, ?x.dest-city(x)))

Embedding this an analysis of utterances conveying questions in a framework such as that described in this chapter would allow the use of answerhood conditions in the same way we have done. However, on our analysis ask-moves have the effect of updating QUD in a specific way. When we introduce request-moves in Chapter 5, we will see that while there are important similarities between request and ask moves, requests do not have the same update effects on the information state as questions do. Since on our approach update effects of utterances are determined by dialogue move type, this is an argument for keeping these two move types separate.

Although we will not deal much with issues involved in planning and plan recognition, we want to stress that there is no reason why the issue-based approach could not be fruitfully combined with mechanisms for planning and plan recognition. In fact, in Chapter 4 we will implement a very basic form of plan recognition.

2.13 Summary

In this chapter, we have laid the groundwork for further explorations of issue-based dialogue management and its implementation in the IBiS system. As a starting point we used Ginzburg's concept of Questions Under Discussion , and we explored the use of QUD as the basis for the dialogue management (Dialogue Move Engine) component of a dialogue system. The basic uses of QUD is to model raising and addressing issues in dialogue, including the resolution of elliptical answers. Also, dialogue plans and a simple semantics were introduced and implemented.

Chapter 3

Grounding issues

3.1 Introduction

In the previous chapter, we assumed "perfect communication" in the sense that all utterances were assumed to be correctly perceived and understood, and fully accepted. Of course, these assumptions are unrealistic both in human-human and human-computer conversation. A useful dialogue system needs to be able to deal with cases of miscommunication and rejections.

We will not attempt to give a complete computational theory about the grounding process in human-human dialogue. Rather, we will provide a basic issue-based account, influenced by Ginzburg, which tries to cover the main phenomena that a dialogue system needs to be able to handle. For instance, the fact that speech recognition is much harder for machines than for humans may motivate different grounding strategies for handling system utterances than for handling user utterances.

First, we provide some dialogue examples where various kinds of feedback are used. We then review and discuss some relevant background, and discuss general types and features of feedback as it appears in human-human dialogue. Next, we discuss the concept of grounding from an information update point of view, and introduce the concepts of optimistic, cautious and pessimistic grounding strategies. This is followed by the main section of this chapter, where we relate grounding and feedback to dialogue systems, discuss the implementation of issue-based grounding and feedback in IBiS2, and provide dialogue examples showing the system's behaviour and how it relates to internal updates. We then review additional implementation issues, and provide a final discussion.

3.1.1 Dialogue examples

The human-human dialogue excerpt[1] in (3.1) shows two common kinds of feedback. J's "mm" shows that J (thinks that he) understood P's previous utterance; P's "pardon" shows that P was not able to hear J's previous utterance. The example also includes a hesitation sound ("um") from J. (P is a customer and J a travel agent.)

(3.1) P : öm (.) flyg ti paris
um (.) flight to paris
J : mm (.) ska du ha en returbiljett
mm (.) do you want a return ticket
P : va sa du
pardon
J : ska du ha en tur å retur
do you want a round trip

The feedback in (3.1) consisted of conventionalized feedback words ("mm", "pardon"). However, feedback may also be more explicit and repeat the central content of the previous utterance, as K's second feedback utterance in (3.2).

(3.2) B : ja ska va framme i $[_1$ göteborg $]_1$ e e ungefär vi nietiden om de finns nå tidit $[_2$ morgonflyg $]_2$
I need to be in Gothenburg er er around nine if there is an early morning flight
K : $[_1$ m $]_1$
m
K : $[_2$ vi $]_2$ nietiden m vi ska se
Around nine m let's see

The function of an utterance answering a question is not primarily to give feedback, but rather to provide task-related information. However, an answer also shows that the previous question was understood and integrated. Example (3.3) shows that feedback is sometimes given in reaction to a question before the question is answered.

(3.3) J : sen måste du ha e sån där intenationellt studentkort också ha du de
then you need one of those international student cards do you have that
P : mm nä
mm no

[1]Except where noted, the human-human dialogues in this chapter have been collected by the University of Lund as part of the SDS project. We quote transcriptions done in Göteborg as part of the same project. The (.) symbol indicates a pause.

However, often human-human communication proceeds without explicit feedback, as in the excerpt from a telephone dialogue in (3.4).

(3.4) K : vart ska du åka
where are you going
B : e jag ska åka till göteborg
er i'm going to gothenburg
K : vilket datum
what date
B : den sjätte juni
the sixth of june
K : vill du åka me flyg eller me tåg
do you want to fly or take the train

By contrast, in current commercial systems it is common to check the system's interpretation of recognized user input asking a *y/n*-question, as seen in the (invented) dialogue in (3.5). This behaviour is much less common in human-human dialogue, but may occur in noisy environments or when communicating over an unreliable communication channel (e.g. when using a mobile phone on a train).

(3.5) S : What day do you wish to travel?
U : Friday
S : Friday the 5th of April, is that correct?

A similar but not identical behaviour does occur in human-human dialogue, as in (3.6), taken from the London-Lund corpus and used by Clark in Clark and Schaefer (1989a). In this example, B gives and explicit report of how A's utterance was interpreted, whereupon A confirms that the interpretation is correct by saying "yes".

(3.6) A : F . six two
B : F six two
A : yes
B : thanks very much

The dialogue in (DIALOGUE 3.1) gives an impression of what feedback capabilities we aim to add to IBiS in this chapter.

(DIALOGUE 3.1)

S> `Welcome to the travel agency!`

```
S> hello
U> price information please
S> You asked about price.  How do you want to travel?
U> pardon
S> How do you want to travel?
U> by plane
S> By plane, is that correct?
U> yes
S> Okay.  What city do you want to go to?
U> lododo
S> Pardon?
U> london
S> Okay, to london.  What city do you want to go from?
U> lyons
S> Sorry, Lyons is not in the database.  So, What city do you want to go
from?
U> paris
...

U> What about connecting flights?
S> You asked about connecting flights.  Sorry, I cannot answer that.
```

3.2 Background

In this section, we provide a brief overview and of some previous accounts of grounding and feedback.

3.2.1 Clark: Adding to the common ground

Clark and Schaefer (1989a) describe grounding as the process of adding to the common ground (see Section 2.2.2). They also formulate the *grounding criterion*:

> Grounding criterion: The contributor and the partners mutually believe that the partners have understood what the contributor meant, to a criterion sufficient for current purposes. (Clark and Schaefer, 1989a, p. 148)

To achieve this, each grounding process goes through two phases:

- **Presentation phase:** A presents utterance u for B to consider. He does so on the assumption that, if B gives evidence e or stronger, he can believe that B understands what A means by u.
- **Acceptance phase.** B accepts utterance u by giving evidence $e\prime$ that he believes he understands what A means by u. He does so on the assumption that, once A registers evidence $e\prime$, he will also believe that B understands. (Clark and Schaefer, 1989a, p. 151)

Clark (1996) argues that utterances involve actions on (at least) four different levels:

(3.7)

Level	**Speaker A's actions**	**Addressee B's actions**
4	A is proposing a joint project w to B	B is considering A's proposal of w
3	A is signalling that p for B	B is recognizing that p
2	A is presenting signal s to B	B is identifying s
1	A is executing behaviour t for B	B is attending to t

Examples of joint projects are adjacency pairs, e.g. one DP asking a question and the other answering it. According to Clark, these four levels of action constitute an *action ladder*, and as such it is subject to the principle of *downward evidence*: "In a ladder of actions, evidence that one level is complete is also evidence that all levels below it are complete". For example, if H understands u, H must also have perceived u and H and S must have established contact; however, H may not accept u.

In Clark and Schaefer (1989a), it is unclear whether grounding includes the proposal / consideration level in addition to understanding[2]. However, in Clark (1996), grounding is redefined to include all levels of action, i.e. attention, identification, recognition and consideration.

> *To ground a thing (...) is to establish it as part of common ground well enough for current purposes.* (...) On this hypothesis, grounding should occur at all levels of communication. (Clark, 1996, p.221, italics in original)

We will adopt this general use of the term grounding to include all four action levels. Also, we assume that the acceptance phase (potentially) concerns all four action levels, rather than only understanding[3].

[2]The definition suggests only understanding is involved, but some examples indicate that utterances which are rejected because of being inappropriate are not grounded.

[3]The term "acceptance phase" is a bit unfortunate, since "acceptance" is used by e.g. Ginzburg to designate the proposal-consideration action level.

Clark lists five ways to signal that a contribution has been successfully interpreted and accepted, ordered from weakest to strongest:

- Continued attention
- Relevant next contribution
- Acknowledgement: "uh-huh", nodding, etc.
- Demonstration: reformulation, collaborative completion
- Display: verbatim display of presentation

The presentation and acceptance phases both focus on externally observable communicative behaviour. However, corresponding to presentations by a speaker on each level of action there is also an "internal" action carried out by the addressee.

Clark views the proposal-consideration process in terms of negotiation, where an utterance such as an assertion or a question is seen as a proposal for a joint project, followed by a response to this proposal. Clark follows Goffman (1976) and Stenström (1984) in distinguishing four main types of responses to proposals of joint projects:

1. full compliance, e.g. answering a question [acceptance]
2. alteration of project, where *H* alters the proposed project to something he is willing to comply with; Clark asserts that alterations may be cooperative (in which case the altered project is still relevant to the original one) or uncooperative [alteration]
3. declination of project, where *H* is unable or unwilling to comply with the project as proposed. Declinations are often performed by offering a reason or justification for declining the proposal. Clark gives the response "I don't know" to a question as an example of declination. [rejection]
4. withdrawal from project, where *H* withdraws from considering the proposal, e.g. by deliberately ignoring a question and changing the topic [withdrawal]

3.2.2 Ginzburg: QUD-based utterance processing protocols

Ginzburg offers an issue-based model of grounding on the understanding and acceptance levels by positing two kinds of grounding-related questions: meaning-questions and acceptance questions[4]. If A produces utterance u, B is faced with a meaning-question, roughly

[4]The latter term is ours. It refers to Ginzburg's MAX-QUD questions discussed below.

"What does u mean?". If B cannot find an answer to this question, B should produce an utterance identical or related to the meaning-question, e.g. "What do you mean?". If B manages to find an answer to this question, he proceeds to consider the acceptance-question, roughly "Should u be accepted?".

Ginzburg's utterance processing protocol (pt. 1) Ginzburg formulates his theory in terms of an utterance processing protocol. Assuming the other DP A has uttered u, this is roughly what happens in the first part of the protocol:

B is faced with the *content-question* $q_{content}(u)$, which we formalize as $\mathbf{?}x\mathbf{.content(}u, x\mathbf{)}$, paraphrasable roughly as "What does u mean (given the current context)?". To answer this question, B must be able to provide a contextual interpretation c of u. This involves, among other things, finding referents for NPs. If B is not able to answer $q_{content}(u)$, B places $q_{content}(u)$ on QUD and produces a $q_{content}(u)$-specific utterance, e.g. a request for clarification. Once an answer to $q_{content}(u)$ has been found, B can be said to have an understanding of u (which may, of course, be a misunderstanding).

Ginzburg notes that utterances behave differently with regard to acceptance depending on whether they have propositions or questions as content. A proposition p can be accepted in two ways: as a fact or as a topic (issue) of discussion. In the latter case, the question under discussion is, roughly, whether p should be accepted as a fact (at least for the purposes of current discussion) or not. Accepting a proposition entails accepting it also as an issue for discussion (although the "discussion" in this case will consist only of the acceptance of p as a fact). The exchanges in (3.8) show some examples of reactions to assertions (note that these examples are not Ginzburg's).

(3.8) a. A: The train leaves at 10 a.m. [answer/assert p]
B: OK, thanks. [accept p]

b. A: The train leaves at 10 a.m. [answer/assert p]
B: No it doesn't! [reject p, accept $?p$ for discussion]

c. A: The train leaves at 10 a.m. [answer/assert p]
B: I'd prefer not to discuss this right now [reject $?p$ for discussion]

d. A: The train leaves at 10 a.m. [answer/assert p]
B: Nice weather, isn't it [ignore p]

Questions, by contrast, can only be accepted as issues for discussion. However, accepting q does not necessarily result in answering q. On this account, answering q should be viewed as one possible way of displaying internal acceptance of q; however, contrary to Clark we also allow the possibility of displaying acceptance of q without answering q.

(3.9) a. A: Where do you want to go [ask q]
B: Paris [answer q, implicitly accept q]

b. A: Where do you want to go [ask q]
B: Hmmm, good question... Do you have any recommendations? [explicitly accept q]

c. A: Where do you want to go [ask q]
B: That's none of your business [explicitly reject q because of unwillingness]

d. A: Where do you want to go [ask q]
B: I don't know [explicitly reject q because of inability]

e. A: Where do you want to go [ask q]
B: I'd like to travel in April [ignore q, answer other question]

f. A: Where do you want to go [ask q]
B: Do you have a student discount? [ignore q, ask other question]

Ginzburg's utterance processing protocol, pt. 2 As we saw above in Section 3.2.2, according to Ginzburg's utterance processing protocol, for a DP B to understand an utterance u amounts to finding an answer to the content-question q_c.

Once B is able to find an answer c which resolves $\mathbf{?}x\mathbf{.content(}u, x\mathbf{)}$, B is faced with the question $q_{accept}(c)$ of whether or not to accept c for discussion, formalized by Ginzburg as **?MAX-QUD(**c**)** ("Whether c should become QUD-maximal"[5]). At this point, the protocol is different for questions and propositions ("facts"). If c is a question and B answers $q_{accept}(c)$ negatively (rejects c for discussion), B pushes c on QUD and produces an c-specific utterance (e.g. "I don't want to discuss that"). If $q_{accept}(c)$ is answered positively and B accepts c, c will be added to QUD and B will produce a c-specific utterance, e.g. an answer to the question c.

If c instead is a proposition and if B answers $q_{accept}(c)$ negatively B should push $q_{accept}(c)$ on QUD and produce a $q_{accept}(c)$-specific utterance. But if $q_{accept}(c)$ is answered positively, B must now consider the question whether c, i.e. $?c$. If the answer is negative (i.e. B does not accept c), the corresponding y/n-question $?c$ is pushed on QUD. This amounts to accepting $?c$ for discussion, which is not the same as accepting c. If B answers $q_{accept}(c)$ positively, B should add c to her FACTS.

[5]This means that the arguably more intuitive interpretation of ?MAX-QUD(c) as "whether c *is* maximal on QUD" is wrong.

For clarity, we reproduce the full protocol in a more schematic way:

try to find an answer resolving $q_{content}(u)$ = **?x.content(u, x)**

- no answer found → push $q_{content}(u)$ on QUD, produce $q_{content}(u)$-specific utterance
- answer c found →
  - c is a question → consider $q_{accept}(c)$ = ?MAX-QUD(c)
    - decide on "no" → push $q_{accept}(c)$ on QUD, produce $q_{accept}(c)$-specific utterance [reject c]
    - decide on "yes" → push c on QUD, produce c-specific utterance [accept c]
  - c is a proposition → consider $q_{accept}(?c)$
    - no → push $q_{accept}(?c)$ on QUD, produce $q_{accept}(?c)$-specific utterance [reject ?c as topic for discussion]
    - yes → consider ?c [accept ?c as topic for discussion]
      - no → push ?c on QUD, produce ?c-specific utterance [reject c as fact]
      - yes → add c to FACTS [accept c as fact]

Note that there are a number of decisions that need to be made by B, and for each of these decisions there is the possibility of rejecting u on the corresponding level. For a question, there is only one way of rejecting it (once the content question has been resolved): to reject it as a question under discussion. This amounts to refusing to discuss the question. For a proposition p, there are two different ways of rejecting it. Firstly, one may reject the issue "whether p" completely; this amounts to refusing to discuss whether p is true or not. Alternatively, one may accept "whether p" for discussion but reject p as a fact.

3.2.3 Allwood: Interactive Communication Management

Allwood (1995) uses the concept of "Interactive Communication Management" to designate all communication dealing with the management of dialogue interaction. This includes feedback but also *sequencing* and *turn management*. Sequencing "concerns the mechanisms, whereby a dialogue is structured into sequences, subactivities, topics etc. ...".

Here, we will use the term ICM as a general term for coordination of the common ground, which in an information state update approach comes to mean explicit signals enabling coordination of updates to the common ground. While feedback is associated with specific utterances, ICM in general does not need to concern any specific utterance.

As will be seen below, we will also be making use of various other parts of Allwood's "activity-based pragmatics" (Allwood, 1995), including Allwood's action level terminology, the concept of Own Communication Management (OCM), and various distinctions concerning ICM.

3.3 Preliminary discussion

In the previous section we have seen examples of different ways of accounting for grounding and feedback. We feel that they all offer useful insights, and that they together can serve as a basis for our further explorations.

Therefore, in this section we will discuss the accounts presented in Section 3.2, relate them to each other, and establish some basic principles and terminological conventions.

3.3.1 Levels of action in dialogue

Both Allwood (1995) and Clark (1996) distinguish four levels of action involved in communication (S is the speaker of utterance u, H is the hearer/addressee). They use slightly different terminologies; here we use Allwood's terminology and add Clark's (and, for the reaction level, also Ginzburg's) corresponding terms in parenthesis. The definitions are mainly derived from Allwood.

- Reaction (acceptance, consideration): whether H has integrated (the content of) u
- Understanding (recognition): whether H understands u
- Perception (identification): whether H perceives u
- Contact (attention): whether H and S have contact, i.e. if they have established a channel of communication

These levels of action are involved in all dialogue, and to the extent that contact, perception, understanding and acceptance can be said to be negotiated, all human-human dialogue has an element of negotiation built in. Note that the above list of levels is formulated in terms of the hearer's perspective.

Given that grounding is concerned with all levels, it follows that four aspects of an utterance u in a dialogue between H and S can in principle be represented in the common ground, one for each action level:

- whether u has been integrated (taken up, accepted)
- whether u has been understood
- whether u has been perceived
- whether S and H have contact

Also, grounding-related feedback may concern any (and possibly several) of these levels.

The level referred to as reaction/acceptance/consideration in the list above is defined differently by different authors. Allwood calls it "reaction (to main evocative intention)", Ginzburg talks about "acceptance", and Clark uses the term "consideration (of joint project)". Perhaps it could be argued that these different definitions are not concerned with the exact same phenomena. Since we want to use the distinction rather than debate it, we choose to emphasize the similarities rather than the differences.

3.3.2 Reaction level feedback

Once an utterance has been understood (or is believed to be understood), in the sense that the hearer has interpreted the utterance to have a meaning and purpose which is relevant in the activity (as perceived by the hearer), the hearer must decide what to do with the utterance. Should he e.g. try to answer the question that was asked, or refuse? Should he choose to commit to an asserted proposition, or raise objections?

The reaction process which follows the understanding of a move M can be analytically divided into three substeps:

- consideration: whether or not to accept and integrate M (and consequently (try to) act on the evocative intention)
- integration: updating the common ground according to M
- feedback: signalling the results of consideration of M

The division of the reaction phase into consideration and feedback is also made in Allwood (1995), using the terms "evaluation" and "report" (respectively), and (though perhaps not so explicitly) in Clark (1996). However, the integration step is not (at least not explicitly) included in either of these accounts.

In the consideration phase, the DP investigates whether he can and wants to accept the proposed joint project or not. If not, he needs to decide whether to alter, decline, or ignore the proposal.

We will use the term *integration* for the silent (internal) consequence of deciding to accept (comply with) a proposed joint project, modelled as the process of updating one's view of the common ground with the full effects of a performed move. By "the full effects" we mean e.g. taking a proposition to be true (at least for the purposes of the conversation) or taking a question as being under discussion. In relation to Clark's use of "uptake", we would say that uptake signals integration. (Of course, uptake may be more than merely a signal that a previous utterance has been integrated, e.g. in the case of answering a question.)

In the feedback phase, the results of the consideration process (acceptance or rejection of issue or proposition) are signalled. The possibility of silent acceptance means that the feedback phase is optional.

Issue and fact acceptance Extending Clark's terminology, we can call the acceptance of a proposition or question as a topic for discussion *issue acceptance*, and the acceptance of a proposition as a fact *fact acceptance*. (We will also use the term *proposition acceptance* for the latter). The former kind of acceptance is available both for questions and propositions, while the latter is available only for propositions. Correspondingly, we can make a distinction between issue rejection and fact rejection in the case of propositions.

Reasons for utterance rejection If a question is asked and the addressee DP decides not to accept it (explicitly or implicitly), this may be explained in at least two ways:

- unwillingness: DP does not want to discuss the issue, e.g. because DP believes other information is more important at the moment
- inability: DP is not able to discuss the issue, e.g. because of confidentiality or lack of knowledge

Regarding the update effects of declining a question, there seems to be an important difference between being unable to answer a question (as e.g. in the case where the response is "I don't know"), and being unwilling to answer it. In the former case, it is not clear that the question is actually rejected as a topic for discussion. The addressee of the question may think that he might eventually come up with an answer (as a result of new information or inference); in this case "I don't know" can be interpreted as "I don't know right now, but

I'll keep the question in mind". In this case, the question might not have to be explicitly raised again before being responded to.

In the case where the rejection displays unwillingness to answer the question (e.g. "No comment", "I will not answer that", "That's none of your business"), it is much clearer that the question is actually rejected as a topic for discussion.

There is also a difference between questions and propositions regarding the reasons for rejection. As Ginzburg notes, asserted propositions may be rejected as issues for discussion, but even if accepted as issues they may be rejected as facts. So for propositions, the consideration phase is more complex than for questions, potentially involving two decisions (e.g. rejecting the asserted proposition as a fact, but accepting it as an issue).

Rejecting a proposition as an issue can be explained by the same kinds of reasons as for any issue. Rejecting a proposition as a fact may be caused e.g. by the addressee having a conflicting belief, or not trusting the speaker. It may also be explained by a belief that accepting the proposition will not serve the goals of the DPs, as e.g. if a customer in a travel agency asserts that the destination city of her flight is Kuala Lumpur, when in fact the agency only serves destinations in Europe. Of course, the proposition that the customer *wants* to travel to Kuala Lumpur can hardly be rejected by the clerk; however, the proposition that Kuala Lumpur is the destination city of a trip that the clerk will provide information about can be rejected. This kind of example is especially relevant for database search systems, where information about the user's desires and intentions is not stored as such.

3.3.3 Levels of understanding

Concerning the levels of action described in Section 3.2.1, we can make further distinctions between different levels of understanding, corresponding to three levels of meaning. These sublevels give a finer grading to the level of understanding. (A similar distinction is also used by Ginzburg (forth)).

- domain-dependent and discourse-dependent meaning (roughly, "content" in the terminology of Barwise and Perry, 1983 and Kaplan, 1979)
  - referential meaning , e.g. referents of pronouns, temporal expressions
  - pragmatic: the relevance of u in the current context
- discourse-independent (but possibly domain-dependent) meaning (roughly corresponding to "meaning" in the terminology of Barwise and Perry, 1983 and Kaplan, 1979), e.g. static word meanings

By "discourse-independent" we mean "independent of the dynamic dialogue context" (modelled in IBiS by the information state proper). However, discourse-independent meaning may still be dependent on static aspects of the activity/domain. It is obvious that these levels of meaning are intertwined and do not have perfectly clear boundaries. Nevertheless, we believe they are useful as analytical approximations.

Since dialogue systems usually operate in limited domains, we will assume that we do not have to deal with ambiguities which are resolved by static knowledge related to the domain. For example, a dialogue system for accessing bank accounts does not have to know that "bank" may also refer to the bank of a river; it is simply very unlikely (though of course not impossible) that the word will be used with this meaning in the activity. It can be argued whether this is always a good strategy, but for now we accept this as a reasonable simplification.

3.3.4 Some comments on Ginzburg's protocol

The first thing to note about Ginzburg's grounding protocol is that it does not specify exactly what kind of feedback should be produced. The notion of question-specificity (see Section 2.8.2) is a minimal requirement that needs to be supplemented with additional heuristics to decide on exactly what feedback to provide. Also, it does not specify how a DP decides when a satisfactory interpretation has been found, or how to resolve the content- and acceptance questions. These are all things we need to be specific about when implementing a dialogue system. (Of course, to the extent they are domain-dependent, we would not expect to find them in a general theory of dialogue. Whether they are domain-dependent or not is, on our view, an open question.)

Second, Ginzburg seems to assume a certain degree of freedom concerning the sharedness of QUD. According to the grounding protocol, DPs are free to add a grounding-related question q to QUD without informing the other DP(s), provided this is followed by an utterance specific to the added question. In fact, the mechanism of question accommodation that will be presented in Chapter 4 provides an explanation of how DPs can understand answers to unasked questions. However, it is not clear that this should allow the speaker to modify QUD *before* uttering the q-specific utterance. It seems inconsistent to say that a DP that assumes QUD is shared can modify QUD without having given any indication of this to the other DP; how would the other DP be able to know about this modification before the q-specific utterance has been made? So it appears that Ginzburg has a notion of QUD as not necessarily entirely shared, and this is slightly different from the notion of QUD we are using. (See also Section 4.8.5.)

Third, Ginzburg only deals with understanding and acceptance; contact and perception are left out. So the protocol above does not deal explicitly with cases where a DP is unsure which words were uttered (however, it deals with perception indirectly since understanding is based on perception).

Relation between Clark's and Ginzburg's accounts

It seems possible to draw some parallels between the two accounts reviewed above. Clark's "recognition of meaning" would presumably be modelled by Ginzburg as finding an answer to the content-question. Similarly, Clark's "consideration of proposal" is modelled as consideration of the acceptance question.

Clark talks about joint projects in "track 2" (meta-communication, as opposed to "track 1", for task-level communication) as involving speakers (often implicitly) raising various issues related to grounding, e.g. "Do you understand this?", and the responder answering these issues (often implicitly). This fits well with the issue-based approach proposed by Ginzburg.

On Clark's account, there is no asymmetry between questions and propositions concerning acceptance. The question-related counterpart of accepting a proposition as a fact, according to Clark, is answering the question. However, it can be argued that answering the question is merely an external behaviour caused by (and acting as positive feedback concerning) the actual acceptance of the question as an issue. Clark's question-related counterpart of rejecting a proposition as a fact (*declination*) is answering e.g. "I don't know" (Clark, 1996, p. 204), and thereby signalling lack of ability or willingness to "comply with the project as proposed". Presumably, "No comment" or "I refuse to answer that question" would also count as rejections on the same level, which indicates that they are really issue-rejections. The "withdrawal" that Clark talks about, where the addressee deliberately ignores a proposal, seems to be equally applicable to both assertions and questions. (In fact, Lewin (2000) views cases where a DP answers a different question than the one that was asked, thereby withdrawing from the proposed question, as rejections.)

3.4 Feedback and related behaviour in human-human dialogue

By feedback we mean behaviour whose primary function is to deal with grounding of utterances in dialogue[6]. This distinguishes feedback from behaviour whose primary function is related to the domain-level task at hand, e.g. getting price information. Non-feedback be-

[6]Since this thesis is not concerned with multimodal dialogue, we will only discuss verbal feedback.

haviour in this sense includes asking and answering task-level questions, giving instructions, etc. (cf. the "Core Speech Acts" of Poesio and Traum, 1998). Answering a domain-level question (e.g. saying "Paris" in response to "What city do you want to go to?") certainly involves aspects of grounding and acceptance, since it shows that the question was understood and accepted. However, the primary function of a domain-level answer is to resolve the question, not to show that it was understood and accepted.

A single utterance may include both feedback and domain-level information. Clark talks about communication of these two types of information as belonging to different "tracks": domain-level information is on track 1 while feedback, and grounding-related communication in general, is on track 2.

In this section we will attempt to give an overview of various aspects of feedback. We will return to sequencing ICM in Section 3.6.9.

3.4.1 Classifying explicit feedback

To get an overview of the range of explicit feedback behaviour that exists in human-human dialogue, we will classify feedback according to four criteria. We will assume that DP S has just uttered or is uttering u to DP H, when the feedback utterance f (uttered by H to S) occurs.

- level of action / basic communicative function (contact, perception, understanding, reaction / acceptance)
- polarity (positive / negative): whether f indicates contact / perception / understanding / acceptance or lack thereof
- eliciting / non-eliciting: whether f is intended to evoke a response (e.g. a reformulation or a reason to accept some content)
- form of f: single word, repetition etc.
- content of f: object-level or meta-level

The action level criterion has been explained above; the others will be explained presently. The criteria of basic communicative function, polarity, eliciting/non-eliciting, and surface form are all derived from Allwood *et al.* (1992) and Allwood (1995).

3.4.2 Positive, negative, and neutral feedback

Positive feedback indicates one or several of contact, perception, understanding, and integration, while negative feedback indicates lack thereof.

While there are clear cases of positive ("uhuh", "ok") and negative ("pardon?", "I don't understand") feedback, there are also some cases which are not so clear. For example, are check-questions (e.g. "To Paris?" in response to "I want to go to Paris") positive or negative? If positive feedback shows understanding, and negative feedback lack of understanding, then check-questions are somewhere in between; they indicate understanding but also that the lack of confidence in that understanding.

Here we will assume a third category of *neutral* feedback for check-questions and similar feedback types. If negative feedback indicates a lack of understanding, neutral feedback indicates lack of confidence in one's understanding.

Negative feedback can be caused by failure to integrate U on any of the levels of action in dialogue:

- lack of contact - H did not notice that S said something
- lack of perception - H did not hear what S said
- lack of understanding on a semantic/pragmatic level - H recognized all the words, but could not extract a content
  - context-independent meaning, e.g. word meanings
  - context-dependent meaning, e.g. referents
  - pragmatic meaning, i.e. the relevance of S's utterance in relation to the context
- rejection of content

For negative feedback, detecting the level with which the feedback is concerned is important for being able to respond appropriately. Here are some possibilities for the different levels:

- contact: try to establish contact ("Hey there")
- perception: speak louder, articulate
- understanding
  - meaning: reformulate

  - pragmatic meaning: reformulate, or explain how the utterance is relevant
- rejection: abandon or argue for the acceptance of the content

3.4.3 Eliciting and non-eliciting feedback

We will use the term "eliciting feedback", borrowed from Allwood *et al.* (1992), to refer to feedback utterances intended to elicit a response, or more specifically utterances u' such that u' is intended to make S respond to u' because H is not sure about how to interpret S's utterance u. Check-questions (both *y/n*- and alternative-questions) are seen as eliciting feedback in this sense. Eliciting feedback can also occur after S's utterance u is finished.

3.4.4 Form of feedback

As with all utterances, feedback utterances can have various syntactic forms:

- assertion
  - declarative ("I heard you say 'go to Paris'.", "You want to go to Paris.")
- imperative ("Please repeat.")
- interrogative
  - *y/n*-question ("Did you say 'Paris'?", "Do you want to go to Paris?")
  - *wh*-question ("What did you say?", "What do you mean?", "Where do you want to go?")
  - alternative-question ("Did you say 'Paris' or 'Ferris'?", "Do you want to go to Paris, France or Paris, Texas?")
- ellipsis ("Paris?", "to Paris.")
- conventional ("Pardon?")

Apart from showing the speaker that one has understood, feedback in the form of an explicit declarative report, repetition or reformulation has the additional function of making sure that the understanding is actually correct, by providing a chance for correction. A *y/n*-question has a similar function, but it indicates less confidence in the interpretation (i.e. is more neutral) and has a stronger eliciting element than an assertion; a question requires an answer, while an assertion can often be assumed to be accepted in the absence of protest.

A related dimension of classification is how the form of the feedback utterance relates to the previous utterance. One way of giving positive feedback is to simply repeat verbatim the previous utterance (e.g. "To Paris." in response to "To Paris."). A similar strategy is to provide a reformulation (e.g. "Your destination city is Paris, the capital of France."). The latter is perhaps a stronger signal of understanding then the former, since a verbatim repetition does not in principle require that the utterance was understood.

3.4.5 Meta-level and object-level feedback

A final distinction can be made depending on whether the feedback explicitly talks about what the speaker said or meant, in which case the feedback can be said to be *meta-level* feedback, or if instead it talks about the subject matter of the dialogue, in which case we talk about *object-level* feedback.

- Meta-level
  - perception ("Did you say 'Paris'?")
  - understanding ("Did you mean that you want to go to Paris?")
- Object-level ("Do you want to go to Paris?")

This distinction does not necessarily apply to all kinds of feedback. For example, for conventional phrases like "Pardon?" and elliptical phrases like "Paris?" it is not clear if they refer to what the speaker said or meant, or about the subject matter of the dialogue, or neither.

3.4.6 Fragment feedback / clarification ellipsis

Often, feedback does not concern a complete utterance, but only a part of it; this is the case e.g. with failure to identify a referent to an NP. We can refer to this kind of feedback as *fragment feedback* (exemplified in (3.10)) and contrast it with *complete feedback* which concerns a whole utterance (as in (3.11)).

(3.10) A: I met Jim yesterday.
B: Who? [negative partial understanding ICM]
Who did you say you met? [partial negative understanding ICM + partial positive understanding ICM]
Jim? [partial interrogative understanding ICM]
Jim Jones? [intermediate partial understanding]
Jim Jones or Jim Lewis? [partial intermediate understanding]
No, it was Bob that you met [partial acceptance, partial rejection[1]]

(3.11) A: I met Jim yesterday.
B: Pardon? [negative complete perception]
B: What do you mean? [negative complete understanding]
B: Liar! [(probably) complete rejection]

It is also possible to give negative partial feedback to one part of an utterance and simultaneously give positive partial feedback to some other part, as when B says "Who did you say you met?"; here, B gives positive feedback that B understood that A met someone, but negative feedback concerning who A met. Cooper and Ginzburg (2001) discuss negative partial feedback using the term "clarification ellipsis" and give an account in the QUD framework.

3.4.7 Own Communication Management

A further aspect related to feedback and ICM is what Allwood refers to as Own Communication Management, which involves hesitation sounds, such as "um", "er" etc., (which also have the effect of keeping the turn), and self-corrections (initiated either by the speaker or by the hearer). It should also be noted that some feedback behaviour also has an OCM aspect; for example, one can explicitly accept a question to "buy time" for coming up with an answer.

3.4.8 Repair and request for repair

One type of behaviour very closely related to feedback, and also to Own Communication Management (see below), is what Traum refers to as "other-initiated repair".

[1]That is, acceptance of the proposition "*A* met someone", but rejection of the proposition "the person that *A* met was Jim".

- other-initiated other-repair: repair by A ("You mean Paris.")
- other-initiated self-repair: request for repair by A ("Do you mean Paris?")

This overlaps with what Clark calls *alteration*, i.e. the case where A accepts an altered version of the proposed joint project.

3.4.9 Request for feedback

Above, we have analyzed feedback produced by the addressee A in response to an utterance u produced by a speaker S. An additional type of feedback behaviours which are produced by the speaker of u are requests for feedback, e.g. "Do you understand?", "Got that?".

- understanding ("Got that?", "Do you understand?")
- acceptance ("OK?", "Do you agree?")

3.5 Update strategies for grounding

After having reviewed grounding-related interactive communication management in human-human dialogue, we will now explore update strategies related to grounding. In this section, we introduce the concepts of optimism, caution, and pessimism regarding grounding update strategies.

3.5.1 Optimistic and pessimistic strategies

According to Clark and Schaefer (1989a), many models of dialogue make a tacit idealization (1) that DPs assume that the content of each utterance is automatically added to the common ground. Some models make the weaker idealization (2) that DPs assume that the content of each utterance is automatically added to the common ground unless there is evidence to the contrary. Clark and Schaefer argue against these idealizations and propose to replace them with "systematic procedures" for establishing mutual belief regarding the addressee's understanding of utterances.

Following Clark, more recent computational models of grounding (e.g. Traum (1994) and Ginzburg (forth)) take it for granted that utterances are not taken to be grounded until some form of feedback has been produced. This feedback need not be explicit: for example, a relevant answer to a question shows that the DP producing the answer has understood and accepted the question posed in the previous utterance. That is, DPs do not make assumptions about grounding until there is some evidence.

However, as noted by Traum this cannot apply to all utterances. If it did, each confirmation of understanding would again have to be confirmed and so on ad infinitum. In our terminology, this means that it is necessary to assume optimism on some level. In Traum's model, acknowledgements are optimistically assumed to be grounded (that is, they do not need to be acknowledged themselves before being added to the common ground) whereas any other conversational acts must receive some acknowledgement before being grounded.

While we believe that Clark is correct in criticizing tacit idealizations about DPs assumptions regarding grounding, we also believe that these tacit assumptions, if made in an explicit and systematic fashion, are not necessarily incorrect or more idealizing than Clark's alternative. Furthermore, they deserve to be explored both theoretically and for practical use in dialogue systems. We should also keep open the possibility that DPs may make different assumptions regarding grounding depending on various factors of the context.

If issues of tacitness are put aside, it seems that what we are dealing with are different accounts of when an utterance is to be regarded as grounded. The need for such an assumption arises partly because the communicative behaviour itself does not completely determine whether an utterance is grounded. At some point, a DP must simply assume that an utterance has been grounded, and we believe that the main difference between them can be described in terms of *optimism* and *pessimism*.

The first type of "tacit idealization" described by Clark ((1) above) can thus be restated as an optimistic grounding assumption; a DP adhering to this assumption will assume, for any utterance u, that u is grounded as soon as it has been uttered, with no regard to feedback. The second type of "idealization" ((2) above) can be restated as a pessimistic grounding assumption, since it requires some way of determining the absence of negative feedback before grounding is assumed. Clark's suggested assumption is also pessimistic, since DPs adhering to it will require positive evidence before assuming that an utterance is grounded.

3.5.2 Grounding updates and action levels

From an information state update perspective, it seems sensible to regard an utterance as grounded when it has been added to the common ground. Each action level connected to an utterance can be associated with a certain type of update. To assume grounding on the perception level can be seen as updating the common ground with the assumed surface form of the utterance; to assume grounding on the understanding level is to update the common ground with a semantic representation of the utterance. Finally, to assume an utterance has been grounded on the acceptance level is to update the common ground with the intended effects of the utterance (e.g. pushing a question on QUD). Thus, the grounding assumption can be divided into four independent assumptions, one for each of these levels; we will concentrate on the understanding and integration levels.

The independence of these assumptions means e.g. that it is possible to make an optimistic assumption about understanding but a pessimistic one about acceptance. This would mean assuming that an utterance was understood as soon as it was uttered, but requiring positive evidence before it is assumed to be accepted.

3.5.3 The cautious strategy

Clark seems to assume that once an utterance has been grounded, there is no turning back; the grounding assumption cannot be undone. That is, the moment information about an utterance is added to the common ground there is no way (short of general strategies for belief revision) of understanding negative feedback and react to it by modifying or removing the grounded material.

However, we believe that there is a difference between assuming an utterance as grounded (added to the common ground) and giving up the possibility of modifying or correcting the grounded material. This opens up a new kind of grounding strategy not included in Clark's account: the cautious strategy.

For a DP using a cautious strategy, it is possible to assume an utterance as being grounded, while still being able to understand and react appropriately to negative feedback. This requires (1) that negative feedback, which is often underspecified in the sense that it does not explicitly identify which part of an utterance it concerns, can be correctly interpreted, and (2) that the DP can revise the common ground in a way which undoes all effects of the erroneous assumption that the utterance was grounded. A simple example is shown in (3.12).

(3.12) A : Do I need a visa?
A optimistically assumes that "does A need a visa?" is now under discussion.
B : Pardon?
A correctly interprets B's utterance as negative feedback (probably on the perception level) regarding the previous utterance, and retracts the assumption that "does A need a visa?" is on QUD.

On this view, the updates associated with grounding involves two steps: adding the material to the common ground, and consequently, removing the possibility of (easily) undoing the updates from the first step.

One advantage of the cautious strategy is that inferences resulting from an utterance can be drawn immediately, without having to wait for feedback, while not requiring costly strategies for general belief revision in cases where the grounding assumption turns out to be premature.

We leave open the question of which strategy is the most cognitively plausible; in fact, the most reasonable assumption is probably that an intelligent combination of different strategies is the most realistic model. But we do believe that the cautious strategy deserves the same attention as the pessimistic strategy advocated by Clark. Moreover, we do not want to preclude the possibility that even the optimistic strategy might come in handy sometimes. As always, questions of cognitive plausibility are best resolved by empirical experimentation rather than by rhetoric. What we want to do here, apart from implementing useful grounding mechanisms for a dialogue system, is to show that the cautious approach is at least a possible alternative.

To repeat, we will use the qualification “optimistic”, “cautious” and “pessimistic” for grounding update strategies, with the following meanings:

- optimistic grounding update: DGB[7] is updated immediately after u was produced (and, in the case of utterances produced by other DP, understood and accepted); no backtracking mechanism available

- cautious grounding update: DGB is updated immediately after u was produced (and, in the case of utterances produced by other DP, understood and accepted); however, backtracking is available

- pessimistic grounding update: DGB is updated when positive evidence of grounding have been acquired

3.6 Feedback and grounding strategies for IBiS

In the previous sections, we discussed grounding-related ICM (and in particular feedback) and grounding strategies in human-human dialogue. In this section, we discuss feedback and grounding from the perspective of dialogue systems in general, and IBiS in particular.

[7]Dialogue Gameboard, i.e. the SHARED part of the information state in IBiS. See Section 2.2.3.

Most (if not all) dialogue systems today have an asymmetrical treatment of grounding, i.e. the grounding of system utterances is handled very differently from the grounding of user utterances. Typically, the system will provide fairly elaborate feedback on user input, usually in the form of questions such as "Did you say you want to go to Paris?". The user must then answer these questions affirmatively before the system will go on. The reason for this, of course, is the low quality of speech recognition.

3.6.1 Grounding strategies for dialogue systems

In this section, we discuss grounding update strategies from the viewpoint of their usefulness in dialogue systems. The two main factors determining the usefulness of an update strategy in a dialogue system is usability (including efficiency of dialogue interaction) and the efficiency of internal processing. On this view, pessimism has the disadvantage that it makes dialogue less efficient since each utterance must be explicitly grounded and accepted; for user utterances this is often achieved by asking check questions to which the user must reply before communication can proceed.

However, the cautiously optimistic approach has the disadvantage that revision is necessary when grounding or acceptance fails, which happens if the user responds negatively to the feedback. The solution presented solves the revision problem by keeping relevant parts of previous information states around.

A common method is to use the recognition score of a user utterance to determine the feedback behaviour from the system, given that the system has understood the utterance sufficiently. We believe that the best solution for an experimental speech-to-speech dialogue system is to switch between grounding update strategies depending on the reliability of communication (which depends on noisiness of environment, previous ratio of successful vs. unsuccessful communication, etc). We link pessimistic and optimistic grounding update to interrogative and positive feedback, respectively. Interrogative feedback from the system raises a question regarding the meaning of a previous utterance, which would not make sense if the system had already assumed that a certain answer to the meaning question had been grounded; in effect, this would amount to raising a question whose answer is (already) assumed to be part of the common ground. Similarly, giving positive feedback corresponds naturally to the case where some interpretation is deemed to be already grounded.

A more sophisticated method for determining what grounding and feedback strategy to use would also take into consideration the degree of relevance of a certain interpretation in the current dialogue context. If a recognition hypothesis which does not have the highest score is nevertheless more relevant than the hypothesis with the highest score, this could result in choosing the former hypothesis. This has not been implemented in IBiS, and is an area for future research.[8]

3.6.2 "Implicit" and "explicit" verification in dialogue systems

In the literature concerning practical dialogue systems (e.g. San-Segundo *et al.*, 2001), grounding is often reduced to verification of the system's recognition of user utterances. Two common ways of handling verification are described as "explicit" and "implicit" verification, exemplified in (3.13) (example from San-Segundo *et al.*, 2001).

(3.13) a. I understood you want to depart from Madrid. Is that correct? [explicit]

b. You leave from Madrid. Where are you arriving at? [implicit]

Actually, both "explicit" and "implicit" feedback contain a verbatim repetition or a reformulation of the original utterance, and in this sense they are both explicit. The actual base for the distinction is what we have here referred to as polarity: "explicit" verification is neutral (and eliciting and interrogative) whereas "implicit" verification is positive.

Given that verification is a rather marginal phenomena in human-human dialogue, it is perhaps surprising that it is often the only aspect of feedback covered in dialogue systems literature. Firstly, because it is usually not necessary for humans to verify what they (think they) have heard; that is, it is a rather uncommon grounding procedure in human-human dialogue. Second, because it only involves part of the full spectrum of feedback behaviour, excluding e.g. acceptance-related feedback behaviour.

3.6.3 Issue-based grounding in IBiS

In this section we outline a (partially) issue-based account of grounding in terms of information state updates, inspired by Ginzburg's account of content questions and acceptance-questions. However, we make significant departures from Ginzburg's account, for various reasons.

A basic idea of the account used in IBiS2 is that meta-issues (the content and acceptance questions) do not always have to be represented explicitly. However, in certain cases it is useful to represent grounding issues explicitly.

[8]For example, one could assess the degree of relevance of a certain `answer`-move by checking how many accommodation steps (see Chapter 4) would be necessary before integrating the question.

Content questions in IBiS

We regard explicit interrogative understanding feedback as explicitly raising content questions, which may be responded to explicitly or implicitly. We also refer to these as *understanding questions*. Explicit interrogative feedback is very relevant for dialogue systems, where poor speech recognition often makes it necessary for the system to explicitly verify each recognized user utterance, giving the user a chance to correct any misunderstandings.

Interrogative feedback can in principle be *wh*-questions ("What do you mean?"), *y/n*-questions ("Do you mean Paris?", "Paris, is that correct?"), or alternative questions ("Do you mean to Paris or from Paris?"). However, we have chosen negative feedback ("I don't understand") rather than *y/n*-questions to indicate lack of understanding. Clarification questions are not used in IBiS2; however, they will be introduced in Chapter 4. This leaves us with *y/n* understanding questions, which concern one specific interpretation of a previous utterance. These are represented in IBiS2 as **?und(**DP*C**)** where DP is a DP and C is a proposition, and can be paraphrased as "Did DP mean C?" or "Is C a correct understanding of DPs utterance?". In the case where the understanding question concerns a question (raised by an ask-move), the proposition C is **issue(**Q**)** where Q is a question. In this case, the paraphrase can be further specified as "Did DP mean to raise Q?".

To allow this, we have extended the IBiS semantic presented in Chapter 2 to include two new kinds of propositions.

- **und(**DP, P**)** : Proposition where P : Proposition and DP : Participant[9]
- **issue(**Q**)** : Proposition where Q : Question[10]

Implicit understanding-questions Actually, the use of the term "implicit" in the context of grounding (or verification) can be used to describe not the grounding behaviour itself but, rather, the status of the grounding issue. What is often referred to as implicit verification can arguably be seen as implicitly raising a grounding question, which may or may not be responded to.

This idea will not be implemented until Chapter 4, since it requires some additional mechanisms which will be needed anyway for the kind of behaviours we introduce there. Specif-

[9]Note that this definition allows P to itself be a proposition of the form und(DP, P'); however, we allow this to pass in the interest of brevity.

[10]This definition allows **issue(**Q**)** as a proposition even when it is not embedded in a proposition **und(**DP**, issue(**Q**))**. In IBiS, the proposition **issue(**Q**)** only appears inside understanding questions or as an argument to an ICM move (see Section 3.6.5). However, in Chapter 4 we will use the corresponding question **?issue(**Q**)**. A suitable paraphrase of this question would be "Should Q become an issue?" or "Should Q be opened for discussion"; this is similar to Ginzburg's MAX-QUD question.

ically, we need a distinction between a global and a local QUD (see also Cooper *et al.* (2000) and Cooper and Larsson (2002)), where the former contains explicitly raised or addressed (but as yet unresolved) issues, and the latter contains questions which can be used for resolving short answers.

To give a short preview, the basic idea is that explicit positive feedback *implicitly* raises an understanding-issue, i.e. when the implicit feedback is integrated, the understanding question is pushed on local QUD. This allows the user to discard the system's interpretation simply by providing a negative answer to the grounding question, or confirm it by giving a positive answer. Since the question is added only to the local QUD, and not to the global one, it will eventually disappear if it is not answered. This allows dialogues to proceed more efficiently, since the user does not have to give explicit confirmations all the time. Again, this will be explained in detail in Chapter 4.

Acceptance questions

In Ginzburg's protocol, a DP who has perceived and interpreted an utterance is faced with the acceptance-question; whether to accept the content of the utterance or not. If the content is not accepted, the DP should push the integrate question (push it on QUD) and address it.

One way of dealing with acceptance would be to follow Ginzburg's account and explicitly represent an acceptance-question which is pushed on QUD in cases where a user utterance is understood but cannot be integrated, and subsequently produce an utterance addressing the acceptance question. We regard feedback-moves on the reaction level (compliance and declination moves, in Clark's terminology) as addressing acceptance questions. Above, we have argued against pushing anything on QUD in this case since it is a shared structure, and the user in this case has no chance of doing the corresponding update on her own QUD until the utterance has been produced. So an alternative strategy would be to first produce the utterance addressing the acceptance question and subsequently assume that the user will accommodate it and push it on QUD; at this time, the system can do the same.

However, it appears that it is only useful to represent the acceptance question explicitly on QUD in cases where it can give rise to a discussion where DPs argue for and against the acceptance of a question as a topic for discussion, or for a proposition as a fact or as a topic for discussion. For a dialogue system unable to perform such argumentation dialogues, it appears pointless to represent the acceptance question explicitly. Since an acceptance or rejection move cannot be challenged, the move will provide a definite answer to the integration question which would thus be immediately removed from QUD once the rejection had been grounded.

For these reasons, we will not represent acceptance issues explicitly in IBiS. In a system capable of negotiation and/or argumentation, however, it would be necessary to do so, and to regard feedback-moves on the acceptance level as relevant answers to this question (see Section 5.8.2 for further discussion).

Temporary storage

To enable cautious grounding we need some way of revising the dialogue gameboard when optimistic grounding assumptions turn out to be premature, without having to deal with the problems of generalized belief revision Gärdenfors (1988). A straightforward way of doing this is to keep around relevant parts of previous dialogue gameboard states, and copy the contents of these back to the DGB when necessary. This strategy will be used for system utterances in IBiS2, and also for user utterances in IBiS3.

3.6.4 Enhancing the information state to handle feedback

In this section, we show how the IBiS information state needs to be modified to handle grounding and feedback. The new information state type is shown in Figure 3.1.

$$\begin{bmatrix} \textsc{private} & : & \begin{bmatrix} \textsc{agenda} & : & \text{OpenQueue(Action)} \\ \textsc{plan} & : & \text{OpenStack(PlanConstruct)} \\ \textsc{bel} & : & \text{Set(Prop)} \\ \textsc{tmp} & : & \begin{bmatrix} \textsc{com} & : & \text{set(Prop)} \\ \textsc{qud} & : & \text{OpenStack(Question)} \\ \textsc{agenda} & : & \text{Stack(Action)} \\ \textsc{plan} & : & \text{Stack(PlanConstruct)} \end{bmatrix} \\ \textsc{nim} & : & \text{OpenQueue(Move)} \end{bmatrix} \\ \textsc{shared} & : & \begin{bmatrix} \textsc{com} & : & \text{Set(Prop)} \\ \textsc{qud} & : & \text{OpenStack(Question)} \\ \textsc{pm} & : & \text{OpenQueue(Move)} \\ \textsc{lu} & : & \begin{bmatrix} \textsc{speaker} & : & \text{Participant} \\ \textsc{move} & : & \text{Set(Move)} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 3.1: IBiS2 Information State type

Temporary store

To enable the system to backtrack if an optimistic assumption turns out to be mistaken, relevant parts of the information state is kept in /PRIVATE/TMP. The QUD and COM fields may change when integrating an ask or answer move, respectively. The plan may also be modified, e.g. if a raise action is selected. Finally, if any actions are on the agenda when selection starts (which means they were put there during by the update module), these may have been removed during the move selection process.

Non-integrated moves

Since several moves can be performed per turn, IBiS needs some way of keeping track of which moves have been interpreted. This is done by putting all moves in LATEST_MOVES in a queue structure called NIM, for Non-Integrated Moves. This structure is private, since it is an internal matter for the system how many moves have been integrated so far. Once a move is assumed to be grounded on the understanding level the move is added to the /SHARED/LU/MOVES set. Since the move has now been understood on the pragmatic level, the content of the move will be a question or a full proposition (for short answers, the proposition resulting from combining it with a question on QUD).

Previous moves

To be able to detect irrelevant followups, IBiS needs to know what moves were performed (and grounded) in the previous utterance. These are stored in the /SHARED/PM field.

Timeout

To be able to decide when the user has given up her turn, we have added a TIS variable TIMEOUT of type Real, whose value is the time (in seconds) after which the system will assume that the turn has been given up if no speech has been detected. This variable will be further discussed in Section 3.6.6.

3.6.5 Feedback and sequencing dialogue moves

In this section, we first show how feedback dialogue moves in IBiS2 are represented. We then review the full range of feedback moves, starting with system-generated feedback and then moving on to user feedback.

The general notation for ICM dialogue moves used in IBiS is the following:

(3.14) $\textsf{icm}{:}L{*}P\{{:}Args\}$

where L is an action level, P is a polarity, and $Args$ are arguments.

- L: action level
  - con: contact (“Are you there?”)
  - per: perception (“I didn’t hear anything from you”, “I heard you say ’to Paris” ’)
  - sem: semantic understanding (“I don’t understand”, “To Paris.”)
  - und: pragmatic understanding (“I don’t quite understand”, “You want to know about price.”)
  - acc: acceptance/reaction (“Sorry, I can’t answer questions about connecting flights”, “Okay.”)
- P: polarity
  - neg: negative
  - int: interrogative
  - pos: positive
- $Args$: arguments

Note that the “neutral” polarity has been replaced by the label “int”; we have made a simplifying assumption that neutral feedback is always eliciting and interrogative.[11]

The arguments are different aspects of the utterance or move which is being grounded, depending action level:

[11]Note, however, that if we had included feedback forms like “What did you say?”, this would still be regarded as negative feedback. The “int” label only refers to check-questions, which are usually *y/n*-questions. This is arguably not an optimal labelling convention.

- for per-level: $String$, the recognized string
- for sem-level: $Move$, a move interpreted from the utterance
- for und-level: $DP * P$, where
  - DP : Participant is the DP who performed the utterance
  - C : Proposition is the propositional content of the utterance
- for acc-level: C : Proposition, the content of the utterance

For example, the ICM move icm:und*pos:usr*dest_city(paris) provides positive feedback regarding a user utterance that has been understood as meaning that the user wants to go to Paris.

In addition, sequencing ICM moves for indicating reraising of issues and loading a plan are included:

- icm:reraise: indicate reraising implicitly ("So, ...")
- icm:reraise:Q: reraising an issue Q explicitly ("Returning to the issue of Price.")
- icm:loadplan ("Let's see.")

System feedback to user utterances in IBiS2

In this section and the following section, we review surface forms related to feedback and other ICM behaviour that will be implemented in IBiS2.

For user utterances, IBiS2 will be able to produce e.g. the following kinds of feedback utterances (for the examples, assume that the user just said "I want to go to Paris"):

- contact
  - negative; icm:con*neg ("I didn't hear anything from you")
- perception
  - negative; icm:per*neg realized as fb-phrase ("Pardon?", "I didn't hear what you said.")

  - positive; icm:per*pos:*String* realized as metalevel verbatim repetition ("I heard 'to paris' ")

- understanding (semantic)
  - negative; icm:sem*neg realized as fb-phrase ("I don't understand.")
  - positive; icm:sem*pos:*Content* realized as repetition/reformulation of content (object-level) ("Paris.")
- understanding (pragmatic)
  - negative; icm:und*neg realized as fb-phrase ("I don't quite understand.")
  - positive; icm:und*pos:*DP*Content* realized as repetition/reformulation of content (object-level) ("To Paris.")
  - interrogative; icm:und*int:*DP*Content* realized as ask about interpretation ("To Paris, is that correct?")
- integration
  - negative
    - proposition-rejection; icm:acc*neg:*Content* realized as explanation ("Sorry, Paris is not a valid destination city")
  - positive; icm:acc*pos realized as fb-word ("Okay")

In addition, IBiS2 will be able to perform issue-rejection using the move icm:acc*neg:issue(Q), where Q : Question as illustrated in (DIALOGUE 3.2).

(DIALOGUE 3.2)

```
U> What about connecting flights?
S> Sorry, I cannot answer questions about connecting flights.
```

We are not claiming that humans always make these distinctions between action explicitly or even consciously, nor that the link between surface form and feedback type is a simple one-to-one correspondence; for example, "mm" may be used as positive feedback on the perception, understanding, and acceptance levels. Feedback is, simply, often ambiguous. However, since IBiS is making all these distinctions internally we might as well try to produce feedback which is not so ambiguous. Of course, there is also a tradeoff in relation to brevity; extremely explicit feedback (e.g. "I understood that you referred to Paris, but I don't see how that is relevant right now.") could be irritating and might decrease the efficiency of the dialogue. We feel that the current choices of surface forms are fairly reasonable, but testing and evaluation on real users would be needed to find the best ways to formulate feedback on different levels. This is an area for future research.

A general strategy used by IBiS in ICM selection is that if negative or interrogative feedback on some level is provided, the system should also provide positive feedback on the level below. For example, if the system produces negative feedback on the pragmatic understanding level, it should also produce positive feedback on the semantic understanding level.

(3.15) S> `Paris. I don't understand.`

In some systems, positive or interrogative feedback to user utterances is not given immediately; instead, the system repeats all the information it has received just before making a database query and asks the user if it is correct. It is also possible to combine feedback after each utterance with a final confirmation. In IBiS2, we have not implemented final confirmations. It can be argued that final confirmations are more important in action-oriented dialogue (see Chapter 5), whereas they are not so important in inquiry-oriented dialogue since they never result in any actions other than database searches.

User feedback to system utterances in IBiS2

For system utterances, IBiS2 will react appropriately to the following types of user feedback:

- perception level
  - negative; fb-phrase ("Pardon?", "Excuse me?", "Sorry, I didn't hear you") interpreted as icm:per*neg
- reaction/acceptance level
  - positive; fb-phrase ("Okay.") interpreted as icm:acc*pos
  - negative; issue rejection fb-phrase ("I don't know", "Never mind", "It doesn't matter") interpreted as icm:acc*neg:issue

In addition, irrelevant followups to system ask-moves are regarded as implicit issue-rejections.

The coverage of user feedback behaviour is thus more limited than the coverage for system behaviour. The main motivation for this is that system utterances are less likely to be problematic for the user to interpret than vice versa.

Still, the available coverage allows some useful feedback-phrases, including negative perception feedback which is useful if the output from the system's speech synthesizer is of poor quality. Ideally, this would provide a slight reformulation by the system, but since generation is not a main topic here, this has not been implemented.

Understanding-level feedback has not been included but may be useful in cases where the user hears the system but cannot understand the meaning of the words uttered by the system. In this case, a reformulation by the system may again be useful.

3.6.6 Grounding of user utterances in IBiS2

In this section we show how optimistic and pessimistic grounding of user utterances has been integrated in IBiS2. First we show how grounding strategies are dynamically selected depending on recognition score, in the case where a move has been fully understood and accepted. Next, we show how to deal with system responses to interrogative feedback associated with pessimistic grounding. Finally, we show how the system deals with failure to perceive, understand, and integrate user utterances by giving negative feedback on the appropriate action level.

Dynamic selection of grounding strategies for user moves

For user utterances, IBiS2 uses optimistic or pessimistic grounding strategies based on the recognition score and the dialogue move type. This makes the corresponding integration rules more complex than the ones in IBiS1. For user "core" moves (in IBiS2, ask and answer), the integration strategy depends on the recognition score for the utterance in question. This choice is determined by two recognition thresholds, T_1 and T_2, where $T_1 > T_2$. If the recognition score is higher than T_2, an optimistic strategy is chosen; positive acceptance feedback ("OK") is selected, and if the score is lower than T_1 positive understanding feedback ("To Paris.") is also selected.

If the score is lower than T_2, the move is not integrated and in the selection phase a pessimistic strategy involving interrogative understanding feedback (e.g. "To Paris, is that correct?") is selected (see Section 3.6.6).

Of course, the idea of using recognition score for determining whether and how to confirm user utterances is not new (see e.g. San-Segundo *et al.*, 2001), and more sophisticated decision procedures are certainly possible. We use it here to show how IBiS2 enables flexible choice both of feedback type and of grounding update strategy.

In addition to being checked for relevance, contentful moves are checked for integratability (acceptability) and if these conditions are not fulfilled the move will not be integrated; instead, it will give rise to negative acceptance feedback as explained in Section 3.6.6.

Integration of user ask move The integration rule for user ask move implementing the optimistic grounding strategy is shown in (RULE 3.1).

```
(RULE 3.1)  RULE: integrateUsrAsk
            CLASS: integrate
                  $/SHARED/LU/SPEAKER==usr
                  fst($/PRIVATE/NIM, ask(Q))
            PRE:  $SCORE=Score
                  Score > 0.7
                  $DOMAIN :: plan(Q, _Plan)
                  1 pop(/PRIVATE/NIM)
                  2 push(/PRIVATE/AGENDA, icm:acc*pos)
                  3 add(/SHARED/LU/MOVES, ask(Q))
                  4 if_do(Score ≤ 0.9,
            EFF:       push(/PRIVATE/AGENDA, icm:und*pos:usr*issue(Q)))
                  5 if_do(in($/SHARED/QUD, Q) and not fst($/SHARED/QUD, Q),
                       push(/PRIVATE/AGENDA, icm:reraise:Q))
                  6 push(/SHARED/QUD, Q)
                  7 push(/PRIVATE/AGENDA, respond(Q))
```

The first two conditions picks out a user ask move on NIM. The third and fourth conditions check the recognition score of the utterance and if it is higher than 0.7 (T_2), the rule proceeds to check for acceptability. If the score is too low, the move should not be optimistically integrated; instead, a pessimistic grounding strategy should be applied and interrogative feedback selected (see below).

The fifth condition checks for acceptability, i.e. that the system is able to deal with this question, i.e. that there is a corresponding plan in the domain resource. If not, the integration rule will not trigger and the ask move will remain on NIM until the selection phase, where it will give rise to an issue rejection (see Section 3.6.6).

The first update pops the integrated move off NIM. In update 2, positive integration feedback is added to the agenda, to indicate that the system can integrate the ask-move. Update 3 adds the move to /SHARED/LU/MOVES, thereby reflecting the optimistic grounding assumption on the understanding level. In update 4, positive understanding feedback is selected unless the score is higher than 0.9 (T_1).

Update 5 checks if this question is already on QUD; if so, the system selects sequencing feedback to show that it has understood that the user is reraising an open issue. (If the question is already on top of QUD, however, it is not seen as a case of reraising.) See Section 3.6.9 for more cases of reraising. Update 6 pushes Q on QUD; not that if Q was already on QUD but not topmost, pushing it will be equivalent to raising it to the topmost position. This is a property of the OpenStack datatype (see Section A.2.1).

Update 7 pushes the action to respond to Q on the agenda. This can be regarded as a shortcut for reasoning about obligations and intentions; when accepting a user question, thus accepting the obligation to try to respond to it, the system will automatically intend to respond to it.

Default ICM move selection rule The role of the ICM move selection rules is to add moves to be generated to the NEXT_MOVES TIS variable based on the contents of the agenda. ICM which is added to the agenda by the update module will be moved to NEXT_MOVES by the default ICM selection rule (RULE 3.2).

(RULE 3.2) RULE: **selectIcmOther**
CLASS: select_icm
PRE: $\left\{\begin{array}{l}\text{in(\$/PRIVATE/AGENDA, icm:}A\text{)}\\ \text{not in(\$NEXT\_MOVES, }B\text{) and }B\text{=ask(}C\text{)}\end{array}\right.$
EFF: $\left\{\begin{array}{l}\text{push(NEXT\_MOVES, icm:}A\text{)}\\ \text{del(/PRIVATE/AGENDA, icm:}A\text{)}\end{array}\right.$

Dialogue example: integrating user ask-move The dialogue below shows how a user ask move with a score of 0.76 is successfully integrated, and positive understanding and acceptance feedback is produced.

(DIALOGUE 3.3)

S> `Welcome to the travel agency!`

U> `price information please` [0.76]

getLatestMoves
$\left\{\begin{array}{l}\text{set(/PRIVATE/NIM, oqueue([ask(?}A\text{.price(}A\text{))]))}\\ \text{set(/SHARED/LU/SPEAKER, usr)}\\ \text{clear(/SHARED/LU/MOVES)}\\ \text{set(/SHARED/PM, set([greet]))}\end{array}\right.$
integrateUsrAsk

$$\left\{\begin{array}{l}
\text{pop}(/\textsc{private}/\textsc{nim}) \\
\text{push}(/\textsc{private}/\textsc{agenda}, \mathsf{icm{:}acc{*}pos}) \\
\text{add}(/\textsc{shared}/\textsc{lu}/\textsc{moves}, \mathsf{ask}(?A.\mathsf{price}(A))) \\
\text{if\_do}(0.76 \leq 0.9, \text{push}(/\textsc{private}/\textsc{agenda}, \mathsf{icm{:}und{*}pos{:}usr{*}issue}(?A.\mathsf{price}(A)))) \\
\text{if\_do}(\text{in}(\$/\textsc{shared}/\textsc{qud}, ?A.\mathsf{price}(A)) \text{ and not fst}(\$/\textsc{shared}/\textsc{qud}, ?A.\mathsf{price}(A)), \\
\quad \text{push}(/\textsc{private}/\textsc{agenda}, \mathsf{icm{:}reraise{:}}?A.\mathsf{price}(A))) \\
\text{push}(/\textsc{shared}/\textsc{qud}, ?A.\mathsf{price}(A)) \\
\text{push}(/\textsc{private}/\textsc{agenda}, \mathsf{respond}(?A.\mathsf{price}(A)))
\end{array}\right.$$

findPlan

$$\left[\begin{array}{lll}
\textsc{private} & = & \left[\begin{array}{lll}
\textsc{agenda} & = & \left\langle\left\langle \begin{array}{l} \mathsf{icm{:}acc{*}pos} \\ \mathsf{icm{:}und{*}pos{:}usr{*}issue}(?A.\mathsf{price}(A)) \\ \mathsf{icm{:}loadplan} \end{array}\right\rangle\right\rangle \\
\textsc{nim} & = & \langle\langle\rangle\rangle
\end{array}\right] \\
\textsc{shared} & = & \left[\begin{array}{lll}
\textsc{com} & = & \{\} \\
\textsc{qud} & = & \langle\ ?A.\mathsf{price}(A)\ \rangle \\
\textsc{pm} & = & \{\ \mathsf{greet}\ \} \\
\textsc{lu} & = & \left[\begin{array}{lll} \textsc{speaker} & = & \mathsf{usr} \\ \textsc{moves} & = & \{\ \mathsf{ask}(?A.\mathsf{price}(A))\ \} \end{array}\right]
\end{array}\right]
\end{array}\right]$$

backupShared
selectFromPlan
selectIcmOther

$$\left\{\begin{array}{l}
\text{push}(\textsc{next\_moves}, \mathsf{icm{:}acc{*}pos}) \\
\text{del}(/\textsc{private}/\textsc{agenda}, \mathsf{icm{:}acc{*}pos})
\end{array}\right.$$

selectIcmOther

$$\left\{\begin{array}{l}
\text{push}(\textsc{next\_moves}, \mathsf{icm{:}und{*}pos{:}usr{*}issue}(?A.\mathsf{price}(A))) \\
\text{del}(/\textsc{private}/\textsc{agenda}, \mathsf{icm{:}und{*}pos{:}usr{*}issue}(?A.\mathsf{price}(A)))
\end{array}\right.$$

selectIcmOther
selectAsk

S> `Okay.  You want to know about price.  Lets see.  How do you want to travel?`

Interrogative understanding feedback for user ask move If a user ask move cannot be assumed to be understood because of a low recognition score, interrogative feedback on the understanding level is selected by (RULE 3.3).

(RULE 3.3) RULE: **selectIcmUndIntAsk**
CLASS: select_icm

$$\textsc{pre:}\ \left\{\begin{array}{l}
\$/\textsc{shared}/\textsc{lu}/\textsc{speaker}{=}{=}\mathsf{usr} \\
\text{fst}(\$/\textsc{private}/\textsc{nim}, \mathsf{ask}(Q)) \\
\$\textsc{score} \leq 0.7
\end{array}\right.$$

$$\textsc{eff:}\ \left\{\begin{array}{l}
\text{pop}(/\textsc{private}/\textsc{nim}) \\
\text{push}(\textsc{next\_moves}, \mathsf{icm{:}und{*}int{:}usr{*}issue}(Q))
\end{array}\right.$$

The conditions are straightforward. The first update removes the move from NIM, even though it has not been integrated. An alternative approach would be to keep this move on NIM and explicitly represent the grounding as concerning this move. However, this would require labelling all moves with unique move IDs; instead, we follow the general philosophy of IBiS of trying to keep our representation as simple as possible as long as it works. The interrogative feedback selected in the second update will, in a sense, take over the function of the original move; if the feedback is answered positively, the end result will be the same as if the ask move had been integrated immediately (see Section 3.6.6 for further explanation).

Integration of user answer move The integration rule for user answer moves, shown in (RULE 3.4) is similar to that for ask moves, except that answers are checked for relevance as well as reliability and acceptability.

```
(RULE 3.4)  RULE: integrateUsrAnswer
            CLASS: integrate
                   1 fst($/PRIVATE/NIM, answer($A$))
                   2 $/SHARED/LU/SPEAKER==usr
                   3 ! $SCORE=$Score$
                   4 $Score > 0.7$
            PRE:   5 fst($/SHARED/QUD, $Q$)
                   6 $DOMAIN :: relevant($A$, $Q$)
                   7 $DOMAIN :: combine($Q$, $A$, $P$)
                   8 $DATABASE :: validDBparameter($P$) or $P$=not($X$)
                   1 pop(/PRIVATE/NIM)
                   2 add(/SHARED/LU/MOVES, answer($P$))
                   3 push(/PRIVATE/AGENDA, icm:acc*pos)
            EFF:   4 if_do($Score \leq 0.9$ and $A \neq$ yes and $A \neq$ no,
                        push(/PRIVATE/AGENDA, icm:und*pos:usr*$P$))
                   5 add(/SHARED/COM, $P$)
```

Conditions 1-4 are similar to those for the **integrateUsrAsk** rule. The relevance of the content of the answer to a question on QUD is checked in condition 6.

The acceptability condition in the condition 8 makes sure that the propositional content resulting from combining the question topmost on QUD with the content of the answer-move is either

- a valid database parameter, or
- a negated proposition

Negated propositions can always be integrated (as long as they are relevant); for example, it is okay to say that you do not want to go to Paris, even if Paris is not in the database.

Updates 1-3 again correspond closely to those in **integrateUsrAsk**. Update 4 checks if the score was lower than or equal to 0.9; if so, a positive understanding feedback move is selected. If the score is higher than 0.9 or if the answer is **yes** or **no**, no understanding feedback is produced. The special special status of "yes" and "no" builds on the assumption that these are easily recognized; if this is not the case, their special status should be removed. Finally, update 5 adds the proposition resulting from combining the question on QUD with the content of the answer move to the shared commitments.

Interrogative understanding feedback for user ask move If a user ask move receives a low score (lower than T_2, which is here set to 0.7) and the question raised by the move is acceptable to the system, interrogative understanding feedback is selected by (RULE 3.5). (If the question is not acceptable it will instead be rejected; see Section 3.6.6).

(RULE 3.5) RULE: **selectIcmUndIntAnswer**
CLASS: select_icm

$$\text{PRE:} \begin{cases} \text{fst}(\$/\text{PRIVATE}/\text{NIM},\ \mathsf{answer}(A)) \\ \$/\text{SHARED}/\text{LU}/\text{SPEAKER}{=}{=}\mathsf{usr} \\ \$\text{SCORE} \leq 0.7 \\ \text{fst}(\$/\text{SHARED}/\text{QUD},\ B) \\ \$\text{DOMAIN} :: \text{relevant}(A,\ B) \\ \$\text{DOMAIN} :: \text{combine}(B,\ A,\ C) \end{cases}$$

$$\text{EFF:} \begin{cases} \text{pop}(/\text{PRIVATE}/\text{NIM}) \\ \text{push}(\text{NEXT\_MOVES},\ \mathsf{icm{:}und{*}int{:}usr{*}}C) \end{cases}$$

The conditions check that there is a user answer move on NIM whose content is relevant to and combines with a question on QUD, and that the recognition score was less than or equal to 0.7. If these conditions are true, the move is popped off NIM and interrogative understanding feedback is selected.

Integrating and responding to interrogative feedback

Integrating interrogative understanding feedback As explained in Section 3.6.3, Interrogative feedback raises understanding questions. This is reflected in (RULE 3.6).

(RULE 3.6) RULE: **integrateUndIntICM**
CLASS: integrate
PRE: { fst($/PRIVATE/NIM, icm:und*int:DP*C)
EFF: { pop(/PRIVATE/NIM)
add(/SHARED/LU/MOVES, icm:und*int:DP*C)
push(/SHARED/QUD, und(DP*C))

The condition simply checks that there is an icm:und*int:DP*C move on NIM, which is then popped off by the first update and added to /SHARED/LU/MOVES by the second update. The third update pushes the understanding question **?und(**DP*C**)** on QUD.

Integrating positive answer to understanding-question When the system raises an understanding question (e.g. by saying "To Paris, is that correct?"), the user can either say "yes" or "no". (The case where the user does not give a relevant answer to the interrogative feedback is treated in Section 3.6.8). In IBiS2, we do not represent propositions related to the understanding of utterances in the same way as other propositions (which are stored in /SHARED/COM). Therefore, special rules are needed for dealing with answers to understanding-questions.

The rule for integrating a negative answer to an understanding-question is shown in (RULE 3.7).

(RULE 3.7) RULE: **integrateNegIcmAnswer**
CLASS: integrate
PRE: { fst($/PRIVATE/NIM, answer(no))
fst($/SHARED/QUD, und($DP$*$C$))
EFF: { pop(/PRIVATE/NIM)
add(/SHARED/LU/MOVES, answer(und(DP*C)))
pop(/SHARED/QUD)
push(/PRIVATE/AGENDA, icm:und*pos:DP*not(C))

The conditions check that there's an answer(yes) move on NIM and an understanding-question on QUD. The first three updates establish the move as shared and pop the understanding-question off QUD. Finally, positive feedback is selected to indicate that the system has understood that the assumed interpretation C was incorrect.

Integrating positive answer to understanding question The rule for integrating a positive answer to an understanding-question is shown in (RULE 3.8).

(RULE 3.8) RULE: **integratePosIcmAnswer**
CLASS: integrate

$$\text{PRE: } \begin{cases} \text{fst}(\$/\text{PRIVATE}/\text{NIM}, \textsf{answer(yes)}) \\ \text{fst}(\$/\text{SHARED}/\text{QUD}, \textsf{und}(DP*Content)) \end{cases}$$

$$\text{EFF: } \begin{cases} \text{pop}(/\text{PRIVATE}/\text{NIM}) \\ \text{add}(/\text{SHARED}/\text{LU}/\text{MOVES}, \textsf{answer}(\textsf{und}(DP*Content))) \\ \text{pop}(/\text{SHARED}/\text{QUD}) \\ \text{if\_then\_else}(Content{=}\textsf{issue}(Q), [\\ \quad \text{push}(/\text{SHARED}/\text{QUD}, Q) \\ \quad \text{push}(/\text{PRIVATE}/\text{AGENDA}, \textsf{respond}(Q))], \\ \text{add}(/\text{SHARED}/\text{COM}, Content)) \end{cases}$$

The conditions and the first three updates are similar to those in the **integrateNegIcmAnswer** rule. The final (conditionalized) update integrates the content C. If the "original" move (the move which caused the interrogative feedback to be produced in the first place) was ask, C will be a proposition **issue**(Q). Consequently, integrating this propositions has the same effects as integrating an ask-move: pushing the question Q on QUD and pushing the action respond(Q) on the agenda. If the proposition is not of this type, it is simply added to /SHARED/COM.

Dialogue example: positive and negative response to interrogative feedback In the following dialogue, the system produces interrogative understanding feedback for two user utterances, one containing an ask move and the other containing an answer move. The first interrogative feedback is answered positively and the second negatively.

(DIALOGUE 3.4)

U> `price information please` [0.65]

getLatestMoves

$$\begin{bmatrix} \text{PRIVATE} & = & \begin{bmatrix} \text{AGENDA} & = & \langle\langle\rangle\rangle \\ \text{PLAN} & = & \langle\rangle \\ \text{NIM} & = & \langle\langle\ \textsf{ask}(?A.\textsf{price}(A))\ \rangle\rangle \end{bmatrix} \\ \text{SHARED} & = & \begin{bmatrix} \text{COM} & = & \{\} \\ \text{QUD} & = & \langle\rangle \\ \text{LU} & = & \begin{bmatrix} \text{SPEAKER} & = & \textsf{usr} \\ \text{MOVES} & = & \{\} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

backupShared
selectIcmUndIntAsk

$$\begin{cases} \text{pop}(/\text{PRIVATE}/\text{NIM}) \\ \text{push}(/\text{PRIVATE}/\text{AGENDA}, \textsf{icm:und*int:usr*issue}(?A.\textsf{price}(A))) \end{cases}$$

selectIcmOther

$$\left\{\begin{array}{l}\text{push(NEXT\_MOVES, icm:und*int:usr*issue(?}A\text{.price(}A\text{)))}\\ \text{del(/PRIVATE/AGENDA, icm:und*int:usr*issue(?}A\text{.price(}A\text{)))}\end{array}\right.$$

S> `You want to know about price , is that correct?`

getLatestMoves
integrateUndIntICM

$$\left\{\begin{array}{l}\text{pop(/PRIVATE/NIM)}\\ \text{add(/SHARED/LU/MOVES, icm:und*int:usr*issue(?}A\text{.price(}A\text{)))}\\ \text{push(/SHARED/QUD, und(usr*issue(?}A\text{.price(}A\text{))))}\end{array}\right.$$

$$\left[\begin{array}{lcl}\text{PRIVATE} & = & \left[\begin{array}{lcl}\text{AGENDA} & = & \langle\langle\rangle\rangle\\ \text{PLAN} & = & \langle\rangle\\ \text{NIM} & = & \langle\langle\rangle\rangle\end{array}\right]\\ \text{SHARED} & = & \left[\begin{array}{lcl}\text{COM} & = & \{\}\\ \text{QUD} & = & \langle\ \text{und(usr*issue(?}A\text{.price(}A\text{)))}\ \rangle\\ \text{LU} & = & \left[\begin{array}{lcl}\text{SPEAKER} & = & \text{sys}\\ \text{MOVES} & = & \{\ \text{icm:und*int:usr*issue(?}A\text{.price(}A\text{))}\ \}\end{array}\right]\end{array}\right]\end{array}\right]$$

U> `yes`

getLatestMoves
integratePosIcmAnswer

$$\left\{\begin{array}{l}\text{pop(/PRIVATE/NIM)}\\ \text{add(/SHARED/LU/MOVES, answer(und(usr*issue(?}A\text{.price(}A\text{)))))}\\ \text{pop(/SHARED/QUD)}\\ \text{if\_then\_else(issue(?}A\text{.price(}A\text{))=issue(}B\text{), [}\\ \quad\text{push(/SHARED/QUD, }B\text{)}\\ \quad\text{push(/PRIVATE/AGENDA, respond(}B\text{))],}\\ \text{add(/SHARED/COM, issue(?}A\text{.price(}A\text{))))}\end{array}\right.$$

findPlan

$$\left[\begin{array}{lcl}\text{PRIVATE} & = & \left[\begin{array}{lcl}\text{AGENDA} & = & \langle\langle\ \text{icm:loadplan}\ \rangle\rangle\\ \text{PLAN} & = & \left\langle\begin{array}{l}\text{findout(?}A\text{.how(}A\text{))}\\ \text{findout(?}B\text{.dest\_city(}B\text{))}\\ \text{findout(?}C\text{.dept\_city(}C\text{))}\\ \text{findout(?}D\text{.month(}D\text{))}\\ \text{findout(?}E\text{.dept\_day(}E\text{))}\\ \text{findout(?}F\text{.class(}F\text{))}\\ \text{consultDB(?}G\text{.price(}G\text{))}\end{array}\right\rangle\\ \text{NIM} & = & \langle\langle\rangle\rangle\end{array}\right]\\ \text{SHARED} & = & \left[\begin{array}{lcl}\text{COM} & = & \{\}\\ \text{QUD} & = & \langle\ ?H\text{.price(}H\text{)}\ \rangle\\ \text{LU} & = & \left[\begin{array}{lcl}\text{SPEAKER} & = & \text{usr}\\ \text{MOVES} & = & \{\ \text{answer(und(usr*issue(?}H\text{.price(}H\text{))))}\ \}\end{array}\right]\end{array}\right]\end{array}\right]$$

backupShared
selectFromPlan

selectIcmOther
selectAsk

S> `Lets see.  How do you want to travel?`

getLatestMoves
integrateOtherICM
integrateSysAsk

U> `by plane` [0.56] *(user actually said "by train")*

getLatestMoves
backupShared
selectIcmUndIntAnswer

$$\left\{\begin{array}{l}\text{pop(/PRIVATE/NIM)}\\ \text{push(/PRIVATE/AGENDA, icm:und*int:usr*how(plane))}\end{array}\right.$$

selectIcmOther

S> `by flight , is that correct?`

getLatestMoves
integrateUndIntICM

$$\left\{\begin{array}{l}\text{pop(/PRIVATE/NIM)}\\ \text{add(/SHARED/LU/MOVES, icm:und*int:usr*how(plane))}\\ \text{push(/SHARED/QUD, und(usr*how(plane)))}\end{array}\right.$$

$$\begin{bmatrix} \text{PRIVATE} & = & \begin{bmatrix} \text{AGENDA} & = & \langle\langle\rangle\rangle \\ \text{PLAN} & = & \left\langle\begin{array}{l}\text{findout}(?A.\text{how}(A))\\ \text{findout}(?B.\text{dest\_city}(B))\\ \text{findout}(?C.\text{dept\_city}(C))\\ \text{findout}(?D.\text{month}(D))\\ \text{findout}(?E.\text{dept\_day}(E))\\ \text{findout}(?F.\text{class}(F))\\ \text{consultDB}(?G.\text{price}(G))\end{array}\right\rangle \\ \text{NIM} & = & \langle\langle\rangle\rangle \end{bmatrix} \\ \text{SHARED} & = & \begin{bmatrix} \text{COM} & = & \{\} \\ \text{QUD} & = & \left\langle\begin{array}{l}\text{und(usr*how(plane))}\\ ?H.\text{how}(H)\\ ?I.\text{price}(I)\end{array}\right\rangle \\ \text{LU} & = & \begin{bmatrix}\text{SPEAKER} & = & \text{sys}\\ \text{MOVES} & = & \{\ \text{icm:und*int:usr*how(plane)}\ \}\end{bmatrix} \end{bmatrix} \end{bmatrix}$$

U> `no`

getLatestMoves
integrateNegIcmAnswer

$$\left\{\begin{array}{l}
\text{pop}(/\textsc{private}/\textsc{nim})\\
\text{add}(/\textsc{shared}/\textsc{lu}/\textsc{moves},\ \mathsf{answer(und(usr*how(plane)))})\\
\text{pop}(/\textsc{shared}/\textsc{qud})\\
\text{push}(/\textsc{private}/\textsc{agenda},\ \mathsf{icm:und*pos:usr*not(how(plane))})
\end{array}\right.$$

$$\left[\begin{array}{lcl}
\textsc{private} & = & \left[\begin{array}{lcl}
\textsc{agenda} & = & \langle\langle\ \mathsf{icm:und*pos:usr*not(how(plane))}\ \rangle\rangle\\
\textsc{plan} & = & \left\langle\begin{array}{l}
\mathsf{findout}(?A.\mathsf{how}(A))\\
\mathsf{findout}(?B.\mathsf{dest\_city}(B))\\
\mathsf{findout}(?C.\mathsf{dept\_city}(C))\\
\mathsf{findout}(?D.\mathsf{month}(D))\\
\mathsf{findout}(?E.\mathsf{dept\_day}(E))\\
\mathsf{findout}(?F.\mathsf{class}(F))\\
\mathsf{consultDB}(?G.\mathsf{price}(G))
\end{array}\right\rangle\\
\textsc{nim} & = & \langle\langle\rangle\rangle
\end{array}\right]\\
\textsc{shared} & = & \left[\begin{array}{lcl}
\textsc{com} & = & \{\}\\
\textsc{qud} & = & \left\langle\begin{array}{l}
?H.\mathsf{how}(H)\\
?I.\mathsf{price}(I)
\end{array}\right\rangle\\
\textsc{lu} & = & \left[\begin{array}{lcl}
\textsc{speaker} & = & \mathsf{usr}\\
\textsc{moves} & = & \{\ \mathsf{answer(und(usr*how(plane)))}\ \}
\end{array}\right]
\end{array}\right]
\end{array}\right]$$

backupShared
reraiseIssue
selectIcmOther

$$\left\{\begin{array}{l}
\text{push}(\textsc{next\_moves},\ \mathsf{icm:und*pos:usr*not(how(plane))})\\
\text{del}(/\textsc{private}/\textsc{agenda},\ \mathsf{icm:und*pos:usr*not(how(plane))})
\end{array}\right.$$

selectIcmOther
selectAsk

```
S> not by flight. So,  How do you want to travel?
```

Negative contact and perception level feedback

What happens if no system utterance is detected, or if the speech recognizer fails? Most speech recognizers can tell the difference between not hearing anything at all, and hearing something but not being able to come up with any hypothesis regarding what was said. We will use this distinction to enable IBiS to produce feedback on the contact and perception levels.

If IBiS does not receive any input within a certain time-frame (specified by the TIMEOUT TIS variable), it will produce feedback indicating that nothing was perceived, e.g. "I didn't hear anything from you.". We classify this as negative feedback on the contact level. It could perhaps be argued that the distinction between contact and perception level feedback is not very sharp, and that this kind of feedback actually concerns the perception level. However, it is possible that the reason that nothing was registered by the recognizer was a failure to establish a channel of communication from the user to the system, e.g. if a microphone is broken or not plugged in properly.

If something is detected by the speech recognizer but it was not able to come up with a good enough guess about what was said, the system will produce negative feedback on the perception level, e.g. "I didn't hear what you said.".

We have configured the input module to set the INPUT variable to 'TIMED_OUT' if nothing is detected, and to 'FAIL' if something unrecognizable was detected.

Negative system contact feedback If the speech recognizer does not get any input within a certain time frame (specified by the TIMEOUT TIS variable), the INPUT variable will be set to 'TIMED_OUT' by the input module. The rule for selection of negative contact feedback is shown in (RULE 3.9).

(RULE 3.9)

```
RULE: selectIcmConNeg
CLASS: select_icm
       { $INPUT= 'TIMED_OUT'
PRE:   { is_empty($NEXT_MOVES)
       { is_empty($/PRIVATE/AGENDA)
EFF:   { push(NEXT_MOVES, icm:con*neg)
```

Unless the system has something else to do, this will trigger negative contact ICM by the system, realised e.g. as "I didn't hear anything from you.". The purpose of this is primarily to indicate to the user that nothing was heard, but perhaps also to elicit some response from the user to show that she is still there. Admittedly, this is a rather undeveloped aspect of ICM in the current IBiS implementation, and alternative strategies could be explored. For example, the system could increase the timeout span successively instead of repeating negative contact ICM every five seconds. Other formulations with more focus on the eliciting function could also be considered, e.g. "Are you there?" or simply "Hello?".

The second and third condition check that nothing is on the agenda or in NEXT_MOVES. The motivation for this is that there is no reason to address contact explicitly in this case, since any utterance from the system implicitly tries to establish contact.

Default ICM integration rule Since contact is not explicitly represented in the information state proper, integration of negative system contact ICM moves have no specific effect on the information state, and are therefore integrated by the default ICM integration rule shown in (RULE 3.10). Unless an ICM move has a specific integration rule defined for it, it will be integrated by this rule.

(RULE 3.10) RULE: **integrateOtherICM**
CLASS: integrate
PRE: { fst($/PRIVATE/NIM, icm:$A$)
EFF: { pop(/PRIVATE/NIM)
add(/SHARED/LU/MOVES, icm:A)

The condition and updates in this rule are straightforward.

Negative system perception feedback If the speech recognizer gets some input from the user but is not able to reliably figure out what was said (the recognition score may be too low), the INPUT variable gets set to 'FAIL'. This will trigger negative perception ICM, e.g. "I didn't hear what you said".

(RULE 3.11) RULE: **selectIcmPerNeg**
CLASS: select_icm
PRE: { $INPUT='FAIL'
not in($NEXT_MOVES, icm:per*neg)
EFF: { push(NEXT_MOVES, icm:per*neg)

The purpose of the second condition is to prevent selecting negative perception feedback more than once in the selection phase. As with negative system contact feedback, negative system perception feedback is integrated by the **integrateOtherICM** rule.

Negative understanding level feedback

Negative feedback can concern either of the two sublevels of the understanding level: semantic and pragmatic understanding.

Negative system semantic understanding feedback If some input is recognized by the recognition module, the interpretation module will try to find an interpretation of the input. If this fails, the LATEST_MOVES gets set to failed which triggers selection of negative semantic understanding feedback (e.g. "I don't understand"). In addition, positive perception feedback (e.g. "I heard 'perish' ") is produced to indicate to the user what the system thought she said.

This will only occur if the recognition lexicon covers sentences not covered by the interpretation lexicon.

```
(RULE 3.12)  RULE: selectIcmSemNeg
             CLASS: select_icm
                    { $LATEST_MOVES=failed
             PRE:   { $INPUT=String
                    { not in($NEXT_MOVES, icm:sem*neg)
             EFF:   { push(NEXT_MOVES, icm:per*pos:String)
                    { push(NEXT_MOVES, icm:sem*neg)
```

The purpose of the third condition is to prevent negative semantic understanding feedback from being selected more than one time. Since only one string is recognized per turn, there is never any reason to apply the rule more than once; and if anything at all can be interpreted, the rule will not trigger at all even if some material was not used in interpretation. In a system with a wide-coverage recognizer and a more sophisticated interpretation module, one may consider producing negative semantic understanding feedback for any material which cannot be interpreted (e.g. “I understand that you want to go to Paris, but I don’t understand what you mean by ‘Londres’.”).

The first update in this rule selects positive perception ICM to show the user what the system heard. The second update selects negative semantic understanding ICM.

Negative system pragmatic understanding feedback The system will try to integrate the moves according to the rules above in Section 3.6.7. If this fails (if there are still moves which have not been integrated), the rule in (RULE 3.13) will be triggered and a icm:und*neg-move will be selected by the system. However, if the reason that the move was not integrated is that it had a low score or was not acceptable to the system, interrogative understanding feedback (Section 3.6.6) or negative acceptance feedback (Section 3.6.6), respectively, will instead be selected and the move will be popped off NIM before the rule in (RULE 3.13) is tried.

In IBiS, only ask-moves can be irrelevant. Other moves, including ask, do not have any relevance requirements. This means that answer moves are the only moves that can fail to be understood on the pragmatic level, given that they have been understood on the semantic level. Also, for an utterance to be completely irrelevant, no part of it must have been integrated. For these reasons, the rule in (RULE 3.13) will trigger only if no move in the latest utterance was integrated, and the utterance was interpreted as containing at least one answer-move.

```
(RULE 3.13)  RULE: selectIcmUndNeg
             CLASS: select_icm
                    ⎧ not in($NEXT_MOVES, icm:und*neg)
                    ⎪ in($LATEST_MOVES, answer(A))
                    ⎪ forall($LATEST_MOVES/ELEM=Move,
             PRE:   ⎨     $/PRIVATE/NIM/ELEM=Move)
                    ⎪ forall($LATEST_MOVES/ELEM=answer(A′),
                    ⎩     not fst($/SHARED/QUD, D) and $DOMAIN :: relevant(A′, Q))
                    ⎧ forall_do($LATEST_MOVES/ELEM=Move,
             EFF:   ⎨     push(NEXT_MOVES, icm:sem*pos:Move))
                    ⎩ push(NEXT_MOVES, icm:und*neg)
```

The first rule checks that negative pragmatic understanding feedback has not already been selected. The second condition checks that the latest utterance contained an answer move, and the third checks that none of the moves performed in the latest utterance has been integrated; all moves in LATEST_MOVES are still on NIM. Finally, the fourth condition checks that no answer is relevant to any question on QUD.

The first update selects positive feedback on the semantic understanding level for each move performed in the latest utterance, to show that the utterance was at least understood to some extent. The second update selects negative feedback and pushes it on NEXT_MOVES.

The system is thus able to make a distinction between utterances it cannot interpret (and thus not ground), and utterances that it can interpret and ground but not integrate. The rule in (3.15) triggers when integration fails because the system cannot see the relevance of the user utterance in the current dialogue context. Negative pragmatic understanding feedback is currently realized as "I don't quite understand"; the idea is to indicate that the utterance was almost fully understood, but not quite. Again, it can be argued what the best realization is.

Dialogue example: negative system contact, perception, and understanding feedback In this dialogue, we see examples of negative system feedback on the contact, perception, and understanding (both semantic and pragmatic) levels. (Since this dialogue uses the text-based input module, we have simulated timeout and recognition failure.)

(DIALOGUE 3.5)

```
S> Welcome to the travel agency!

U> 'TIMED_OUT'
```

getLatestMoves
integrateNoMove
{ pop(/PRIVATE/NIM)
selectIcmConNeg
{ push(NEXT_MOVES, icm:con*neg)

S> `I didnt hear anything from you.`

getLatestMoves
integrateOtherICM
{ pop(/PRIVATE/NIM)

U> `'FAIL'`

backupShared
selectIcmPerNeg
{ push(NEXT_MOVES, icm:per*neg)

S> `Sorry, I didnt hear what you said.`

getLatestMoves
integrateOtherICM
{ pop(/PRIVATE/NIM)

U> `jfdbhajhdgarbledfdasd`

backupShared
selectIcmSemNeg

S> `I heard you say jfdbhajhdgarbledfdasd. Sorry, I dont understand.`

getLatestMoves
integrateOtherICM
{ pop(/PRIVATE/NIM)
integrateOtherICM
{ pop(/PRIVATE/NIM)

U> `paris`

getLatestMoves
backupShared
selectIcmUndNeg

```
{ forall_do($/SHARED/LU/MOVES/ELEM=$A$, push(NEXT_MOVES, icm:sem*pos:$A$))
  push(NEXT_MOVES, icm:und*neg)
  forall_do(in($/SHARED/LU/MOVES, $E$) and $E$=answer($C$) and $LEXICON :: yn_answer($C$) and
      in($/PRIVATE/NIM, $E$),
      del(/PRIVATE/NIM, $E$))
```

S> `paris. I dont quite understand.`

getLatestMoves
integrateOtherICM
{ pop(/PRIVATE/NIM)
integrateOtherICM
{ pop(/PRIVATE/NIM)

Negative reaction level feedback

To be able to distinguish relevant but non-integratable utterances from utterances which are both relevant and integratable, a dialogue system in general needs to be able to distinguish between relevance and integratability (acceptability), i.e. it needs to understand the relevance of utterances that it cannot integrate.

System proposition-rejection In addition to issue-rejection, proposition-rejection is also arguably relevant to a dialogue system. A case in point is when the user supplies information which results in an invalid database query, i.e. a query which would yield no results. An example is given in (DIALOGUE 3.6) (understanding-feedback has been removed for readability).

(DIALOGUE 3.6)

U(1)> `Price information please`
S(1)> `OK. Where do you want to travel?`
U(2)> `to Paris`
S(2)> `OK. What city you want to travel from?`
U(3)> `Oslo`
S(3)> `Oslo.  Sorry, there are no flights matching your specification.`

However, this case is a bit more problematic - is S(3) really a rejection of U(3), or should it be regarded as a negative answer to the user's query in U(1)? We believe it makes more sense to do the latter. On this view, the issue of price will be regarded as (negatively) resolved after S(3). (Note that we are here assuming that Oslo is in fact a valid departure city, but there happen to be no flights from Oslo to Paris in the database.)

A variant of the dialogue in (DIALOGUE 3.6) that is perhaps a better case of rejection is where the user supplies a destination which is not available in the database. In this case, it seems to make sense to say that it is indeed the utterance containing the information about the destination that is rejected.

(DIALOGUE 3.7)

U(1)> `Price information please`
S(1)> `OK. Where do you want to travel?`
U(2)> `to Paris`
S(2)> `OK. What city you want to travel from?`
U(3)> `Kuala Lumpur`
S(3)> `Sorry, Kuala Lumpur is not in the database.  So, What city do you want to travel from?`[12]

In this case, the issue of price is still unresolved, as is the issue of destination city. To handle a dialogue like that in (DIALOGUE 3.7), a system again needs to be able to recognize relevant information that it cannot deal with, and distinguish it from such information that it can deal with. One way of doing this is to encode relevant information in the domain knowledge resource that is not necessarily in the database. If a user utterance that contains a relevant answer or assertion is perceived and understood, the system should perform a database search to check if it is able to deal with that information; if not, the user's utterance should be rejected.

Of course, it is a well-known problem that bigger vocabularies make speech recognition harder, and consequently there's a tradeoff between recognizing and dealing correctly with non-acceptable information, and getting the acceptable information right. Possibly, one could use collected dialogues in a domain to decide how much non-acceptable information the system should be able to recognize and understand.

In IBiS, we have implemented the ability to reject user answers by checking whether they provide valid database parameters. This requires an additional database resource condition "validDBparameter(P)" which is true if P is a valid parameter in the database. For example, if a travel agency database contains flights within Europe, any destination outside Europe is an invalid database parameter and should be rejected by the system.

[12]Optionally, one might want a system to be more helpful and offer a suitable alternative destination.

(RULE 3.14) RULE: **rejectProp**
CLASS: select_action

$$\text{PRE: } \begin{cases} \text{in(\$/PRIVATE/NIM, answer}(A)) \\ \text{\$/SHARED/LU/SPEAKER=usr} \\ \text{fst(\$/SHARED/QUD, } Q) \\ \text{\$DOMAIN :: relevant}(A, Q) \\ \text{\$DOMAIN :: combine}(Q, A, P) \\ \text{not \$DATABASE :: validDBparameter}(P) \end{cases}$$

$$\text{EFF: } \begin{cases} \text{del(/PRIVATE/NIM, answer}(A)) \\ \text{push(/PRIVATE/AGENDA, icm:und*pos:usr*}P) \\ \text{push(/PRIVATE/AGENDA, icm:acc*neg:}P) \end{cases}$$

The first five conditions are identical to those for the rule for integrating user answers, **integrateUsrAnswer** (Section 3.6.6). The final condition checks that the proposition P, resulting from combining a question on QUD with the content of the answer move, is not a valid database parameter. The updates remove the move from NIM and selects positive understanding feedback to show what the system understood, and negative acceptance feedback.

Of course, it is not optimally efficient that the same sequence of conditions is checked by several different rules; an alternative approach would be to let one rule determine e.g. how an answer move is relevant, combine it with a question on QUD, and store the result in a datastructure containing pragmatically interpreted material. This datastructure could then be inspected by both integration and rejection rules. (See also Section 6.5.1.)

Dialogue example: system proposition rejection In the following dialogue, we illustrate system rejection of the proposition that the means of transport to search for will be train. A motivation is also given by the system, i.e. that “train” is not available as a means of transport in the database.

(DIALOGUE 3.8)

S> `Okay.  I need some information.  How do you want to travel?`

getLatestMoves
integrateOtherICM
integrateOtherICM
integrateSysAsk

U> `train please`

getLatestMoves
backupShared
rejectProp

$\left\{\begin{array}{l}\text{del(/PRIVATE/NIM, answer(train))}\\ \text{push(/PRIVATE/AGENDA, icm:und*pos:usr*how(train))}\\ \text{push(/PRIVATE/AGENDA, icm:acc*neg:how(train))}\end{array}\right.$

selectIcmOther

$\left\{\begin{array}{l}\text{push(NEXT\_MOVES, icm:und*pos:usr*how(train))}\\ \text{del(/PRIVATE/AGENDA, icm:und*pos:usr*how(train))}\end{array}\right.$

selectIcmOther

$\left\{\begin{array}{l}\text{push(NEXT\_MOVES, icm:acc*neg:how(train))}\\ \text{del(/PRIVATE/AGENDA, icm:acc*neg:how(train))}\end{array}\right.$

```
S> by train. Sorry,  by train  is not in the database.
```

getLatestMoves
integrateOtherICM
integrateOtherICM

System issue-rejection For example, the system might know some questions which are relevant in a certain activity, but not be able to answer them. This is not usually the case with existing dialogue systems. For example, the Swedish railway information system (based on the Philips dialog system (Aust *et al.*, 1994) cannot answer questions about the availability of a cafeteria on a train. If this question is asked, the system will try to interpret it as an answer to something it just asked about (as illustrated in the made-up dialogue (3.16)). But one could imagine a system that would have a store of potentially relevant questions which it cannot handle, enabling it to respond to such questions in a more appropriate way, e.g. by saying "Sorry, I cannot answer that question". This would constitute a rejection (an issue-rejection, to be precise) of a question whose meaning has been understood. An (made-up) example is shown in (3.17).

(3.16) U : Is there a cafeteria on the train?
S : You want to travel to Siberia, is that correct?

(3.17) U : Is there a cafeteria on the train?
S : Sorry, I cannot answer questions about cafeteria availability.

Issue rejection has been implemented in IBiS2 for the travel agency domain; in the travel agency domain, the system will recognize and understand, but reject, questions about connecting flights. A possible extension of this would be to make the system more helpful and make it explain why it cannot answer the question; this has not yet been done in IBiS.

In case the system has interpreted a user utterance as an ask-move with content q, but the system does not have a plan for dealing with q, the system must reject q and indicate this to the user using appropriate feedback. This rule allows the system to respond intelligently to user questions even if it cannot answer them (given that they can be recognized and interpreted).

(RULE 3.15) RULE: **rejectIssue**
CLASS: select_action

$$\text{PRE: } \begin{cases} \text{in}(\$/\text{PRIVATE}/\text{NIM},\ \mathsf{ask}(Q)) \\ \$/\text{SHARED}/\text{LU}/\text{SPEAKER}{=}\mathsf{usr} \\ \text{not } \$\text{DOMAIN} :: \text{plan}(Q,\ \_Plan) \end{cases}$$

$$\text{EFF: } \begin{cases} \text{del}(/\text{PRIVATE}/\text{NIM},\ \mathsf{ask}(Q)) \\ \text{push}(/\text{PRIVATE}/\text{AGENDA},\ \mathsf{icm{:}und{*}pos{:}usr{*}issue}(Q)) \\ \text{push}(/\text{PRIVATE}/\text{AGENDA},\ \mathsf{icm{:}acc{*}neg{:}issue}(Q)) \end{cases}$$

The rule is similar to the **rejectProp** rule. The third condition checks that there is no plan for dealing with the question Q.

Dialogue example: system issue rejection In the following dialogue, the user's request for information about connecting flights is rejected on the grounds that the system does not know how to address that issue.

(DIALOGUE 3.9)

S> `Okay. The price is 123 crowns.`

U> `what about connecting flights`

getLatestMoves
backupShared
rejectIssue

$$\begin{cases} \text{del}(/\text{PRIVATE}/\text{NIM},\ \mathsf{ask}(?A.\mathsf{con\_flight}(A))) \\ \text{push}(/\text{PRIVATE}/\text{AGENDA},\ ) \\ \text{push}(/\text{PRIVATE}/\text{AGENDA},\ \mathsf{icm{:}acc{*}neg{:}issue}(?A.\mathsf{con\_flight}(A))) \end{cases}$$

selectIcmOther

$$\begin{cases} \text{push}(\text{NEXT\_MOVES},\ \mathsf{icm{:}und{*}pos{:}usr{*}issue}(?A.\mathsf{con\_flight}(A))) \\ \text{del}(/\text{PRIVATE}/\text{AGENDA},\ \mathsf{icm{:}und{*}pos{:}usr{*}issue}(?A.\mathsf{con\_flight}(A))) \end{cases}$$

selectIcmOther

$$\begin{cases} \text{push}(\text{NEXT\_MOVES},\ \mathsf{icm{:}acc{*}neg{:}issue}(?A.\mathsf{con\_flight}(A))) \\ \text{del}(/\text{PRIVATE}/\text{AGENDA},\ \mathsf{icm{:}acc{*}neg{:}issue}(?A.\mathsf{con\_flight}(A))) \end{cases}$$

S> `You asked about connecting flights. Sorry, I cannot answer questions`

```
about connecting flights.
```

getLatestMoves
integrateOtherICM
integrateOtherICM

3.6.7 Grounding of system utterances in IBiS2

In this section, we show how a cautiously optimistic grounding strategy for system utterances has been implemented in IBiS2. We first present basic update rules reflecting the cautious strategy. We then present integration rules for the "core" system dialogue moves (ask and answer), and describe the rules for integrating user feedback to system moves.

Enabling cautious updates

IBiS2 uses a mix of various grounding strategies. For system utterances, a cautiously optimistic strategy is used.

Moving latest moves to nim The IBiS2 version of the update rule **getLatestMoves** is shown in (RULE 3.16).

(RULE 3.16) RULE: **getLatestMoves**
CLASS: grounding

$$\text{PRE:}\left\{\begin{array}{l} \$\text{LATEST\_MOVES}=Moves \\ \$\text{LATEST\_SPEAKER}=DP \\ \$/\text{SHARED}/\text{LU}/\text{MOVES}=PrevMoves \end{array}\right.$$

$$\text{EFF:}\left\{\begin{array}{l} \text{set}(/\text{PRIVATE}/\text{NIM},\ Moves) \\ \text{set}(/\text{SHARED}/\text{LU}/\text{SPEAKER},\ DP) \\ \text{clear}(/\text{SHARED}/\text{LU}/\text{MOVES}) \\ \text{set}(/\text{SHARED}/\text{PM},\ PrevMoves) \end{array}\right.$$

The rule loads information regarding the latest utterance performed into NIM and copies the previously grounded moves (in /SHARED/LU/MOVES) to the /SHARED/PM field. Note that this rule has changed significantly compared to IBiS1; no optimistic assumption about understanding of the latest utterance is made here. Instead of putting the latest moves in /SHARED/LU/MOVES, which would be to assume that they have been mutually understood, IBiS2 clears /SHARED/LU/MOVES so that moves can be added when they are actually integrated; only then are they assumed to be understood.

Saving previous state before integration Before selecting, producing, and integrating a new system utterance, the rule in (RULE 3.17) copies relevant parts of the IS to the TMP field. This makes it possible to backtrack to a previous state, should the optimistic grounding assumptions concerning a system move turn out to be mistaken. This means that any optimistic updates associated with integration of system moves are now cautiously optimistic.

(RULE 3.17) RULE: **backupShared**
CLASS: none
PRE: {

$$\text{EFF:}\left\{\begin{array}{l}\text{/PRIVATE/TMP/QUD} := \text{\$/SHARED/QUD}\\ \text{/PRIVATE/TMP/COM} := \text{\$/SHARED/COM}\\ \text{/PRIVATE/TMP/AGENDA} := \text{\$/PRIVATE/AGENDA}\\ \text{/PRIVATE/TMP/PLAN} := \text{\$/PRIVATE/PLAN}\end{array}\right.$$

There are no conditions on this rule. It is executed at the start of the selection algorithm described in Section 3.7, and is thus only called before system utterances.

Cautiously optimistic integration of system moves

For system ask and answer moves, the integration rules are similar to those in IBiS1; however, rather than picking out moves from /SHARED/LU/MOVES, IBiS2 picks moves from /PRIVATE/NIM and adds them to /SHARED/LU/MOVES, thereby assuming grounding on the understanding level, only in connection with integration. Since optimistic grounding is assumed for system moves, it would be okay to handle them the same way we did in IBiS1; however, user moves are no longer (always) optimistically grounded, and we have chosen to give a uniform treatment to all moves. Since in IBiS system moves are always successfully integrated, however, there is no real difference between the way they are handled in IBiS1 and IBiS2.

(RULE 3.18) RULE: **integrateSysAsk**
CLASS: integrate

$$\text{PRE: } \begin{cases} \$/\text{SHARED}/\text{LU}/\text{SPEAKER}{==}\mathsf{sys} \\ \text{fst}(\$/\text{PRIVATE}/\text{NIM},\ \mathsf{ask}(A)) \end{cases}$$

$$\text{EFF: } \begin{cases} \text{pop}(/\text{PRIVATE}/\text{NIM}) \\ \text{add}(/\text{SHARED}/\text{LU}/\text{MOVES},\ \mathsf{ask}(A)) \\ \text{push}(/\text{SHARED}/\text{QUD},\ A) \end{cases}$$

(RULE 3.19) RULE: **integrateSysAnswer**
CLASS: integrate

$$\text{PRE: } \begin{cases} \text{fst}(\$/\text{PRIVATE}/\text{NIM},\ \mathsf{answer}(A)) \\ \$/\text{SHARED}/\text{LU}/\text{SPEAKER}{==}\mathsf{sys} \\ \$\text{DOMAIN} :: \text{proposition}(A) \\ \text{fst}(\$/\text{SHARED}/\text{QUD},\ B) \\ \$\text{DOMAIN} :: \text{relevant}(A,\ B) \end{cases}$$

$$\text{EFF: } \begin{cases} \text{pop}(/\text{PRIVATE}/\text{NIM}) \\ \text{add}(/\text{SHARED}/\text{LU}/\text{MOVES},\ \mathsf{answer}(A)) \\ \text{add}(/\text{SHARED}/\text{COM},\ A) \end{cases}$$

One complication is that in IBiS2, several moves may be performed in a single utterance. To keep track of which utterances have been integrated, the /PRIVATE/NIM stack of non-integrated moves is popped for each move that gets integrated. Note also that each integrated (and thus understood) move is added to /SHARED/LU/MOVES (whereas in IBiS1 this was done at the start of the update cycle).

The cautiously optimistic acceptance assumptions built into these rules can be retracted on integration of negative user perception feedback, as explained in Section 3.6.6, or on negative user integration feedback, as show in Section 3.6.7. Dialogue examples involving the rules shown above will be given in these sections.

User feedback to system utterances

In this section we review user feedback to system utterances and how these affect the optimistic grounding assumptions.

Negative user perception feedback If the system makes an utterance, it will assume it is grounded and accepted. If the user indicates that she did not understand the utterance, the rule in (RULE 3.20) makes it possible to retract the effects of the system's latest move, thus cancelling the assumptions of grounding and acceptance.

(RULE 3.20) RULE: **integrateUsrPerNegICM**
CLASS: integrate
PRE: $\left\{ \begin{array}{l} \$/\text{SHARED/LU/SPEAKER}{=}{=}\text{usr} \\ \text{fst}(\$/\text{PRIVATE/NIM}, \text{icm:per*neg}) \end{array} \right.$
EFF: $\left\{ \begin{array}{l} \text{pop}(/\text{PRIVATE/NIM}) \\ /\text{SHARED/QUD} := \$/\text{PRIVATE/TMP/QUD} \\ /\text{SHARED/COM} := \$/\text{PRIVATE/TMP/COM} \\ /\text{PRIVATE/AGENDA} := \$/\text{PRIVATE/TMP/AGENDA} \\ /\text{PRIVATE/PLAN} := \$/\text{PRIVATE/TMP/PLAN} \end{array} \right.$

The four last updates revert the COM, QUD, PLAN and AGENDA fields to the values stored in /PRIVATE/TMP.

Dialogue example: negative user perception feedback This dialogue shows how IBiS2 is able to react to negative user perception feedback (e.g. "pardon") by retracting the optimistic grounding assumption by backtracking relevant parts of SHARED to the state in /PRIVATE/TMP/SYS, stored before the system utterance was generated. Also, the plan and agenda are backtracked to enable the system to continue the dialogue properly.

(DIALOGUE 3.10)

S> `Okay. You asked about price. I need some information. How do you want to travel?`

getLatestMoves
integrateOtherICM
integrateOtherICM
integrateOtherICM
integrateSysAsk

$$
\left[\begin{array}{ll}
\textsc{pr.} = & \left[\begin{array}{lll}
\textsc{agenda} & = & \langle\langle\rangle\rangle \\
\textsc{plan} & = & \left\langle\begin{array}{l}
\mathsf{findout}(?A.\mathsf{how}(A)) \\
\mathsf{findout}(?B.\mathsf{dest\_city}(B)) \\
\mathsf{findout}(?C.\mathsf{dept\_city}(C)) \\
\mathsf{findout}(?D.\mathsf{month}(D)) \\
\mathsf{findout}(?E.\mathsf{dept\_day}(E)) \\
\mathsf{findout}(?F.\mathsf{class}(F)) \\
\mathsf{consultDB}(?G.\mathsf{price}(G))
\end{array}\right\rangle \\
\textsc{bel} & = & \{\} \\
\textsc{tmp} & = & \left[\begin{array}{lll}
\textsc{com} & = & \{\} \\
\textsc{qud} & = & \langle\ ?H.\mathsf{price}(H)\ \rangle \\
\textsc{agenda} & = & \left\langle\left\langle\begin{array}{l}
\mathsf{icm{:}acc{*}pos} \\
\mathsf{icm{:}und{*}pos{:}usr{*}issue}(?H.\mathsf{price}(H)) \\
\mathsf{icm{:}loadplan}
\end{array}\right\rangle\right\rangle \\
\textsc{plan} & = & \left\langle\begin{array}{l}
\mathsf{findout}(?A.\mathsf{how}(A)) \\
\mathsf{findout}(?B.\mathsf{dest\_city}(B)) \\
\mathsf{findout}(?C.\mathsf{dept\_city}(C)) \\
\mathsf{findout}(?D.\mathsf{month}(D)) \\
\mathsf{findout}(?E.\mathsf{dept\_day}(E)) \\
\mathsf{findout}(?F.\mathsf{class}(F)) \\
\mathsf{consultDB}(?G.\mathsf{price}(G))
\end{array}\right\rangle
\end{array}\right] \\
\textsc{nim} & = & \langle\langle\rangle\rangle
\end{array}\right] \\
\textsc{sh.} = & \left[\begin{array}{lll}
\textsc{com} & = & \{\} \\
\textsc{qud} & = & \left\langle\begin{array}{l}
?I.\mathsf{how}(I) \\
?H.\mathsf{price}(H)
\end{array}\right\rangle \\
\textsc{lu} & = & \left[\begin{array}{lll}
\textsc{speaker} & = & \mathsf{sys} \\
\textsc{moves} & = & \langle\langle\ \mathsf{icm{:}acc{*}pos},\ \ldots\ \rangle\rangle
\end{array}\right] \\
\textsc{pm} & = & \langle\langle\ \mathsf{ask}(?H.\mathsf{price}(H))\ \rangle\rangle
\end{array}\right]
\end{array}\right]
$$

U> `pardon`

getLatestMoves
integrateUsrPerNegICM

$$
\left\{\begin{array}{l}
\text{pop}(/\textsc{private}/\textsc{nim}) \\
/\textsc{shared}/\textsc{qud} := \$/\textsc{private}/\textsc{tmp}/\textsc{qud} \\
/\textsc{shared}/\textsc{com} := \$/\textsc{private}/\textsc{tmp}/\textsc{com} \\
/\textsc{private}/\textsc{agenda} := \$/\textsc{private}/\textsc{tmp}/\textsc{agenda} \\
/\textsc{private}/\textsc{plan} := \$/\textsc{private}/\textsc{tmp}/\textsc{plan}
\end{array}\right.
$$

$$\left[\begin{array}{lll}
\textsc{private} & = & \left[\begin{array}{lll}
\textsc{agenda} & = & \left\langle\!\left\langle \begin{array}{l} \mathsf{icm{:}acc{*}pos} \\ \mathsf{icm{:}und{*}pos{:}usr{*}issue}(?A.\mathsf{price}(A)) \\ \mathsf{icm{:}loadplan} \end{array}\right\rangle\!\right\rangle \\
\textsc{plan} & = & \left\langle \begin{array}{l} \mathsf{findout}(?B.\mathsf{how}(B)) \\ \mathsf{findout}(?C.\mathsf{dest\_city}(C)) \\ \mathsf{findout}(?D.\mathsf{dept\_city}(D)) \\ \mathsf{findout}(?E.\mathsf{month}(E)) \\ \mathsf{findout}(?F.\mathsf{dept\_day}(F)) \\ \mathsf{findout}(?G.\mathsf{class}(G)) \\ \mathsf{consultDB}(?H.\mathsf{price}(H)) \end{array}\right\rangle \\
\textsc{bel} & = & \{\} \\
\textsc{tmp} & = & \ldots \\
\textsc{nim} & = & \langle\langle\rangle\rangle
\end{array}\right] \\
\textsc{shared} & = & \left[\begin{array}{lll}
\textsc{com} & = & \{\} \\
\textsc{qud} & = & \langle\ ?A.\mathsf{price}(A)\ \rangle \\
\textsc{lu} & = & \left[\begin{array}{lll} \textsc{speaker} & = & \mathsf{usr} \\ \textsc{moves} & = & \mathsf{oqueue}([\mathsf{icm{:}per{*}neg}]) \end{array}\right] \\
\textsc{pm} & = & \ldots
\end{array}\right]
\end{array}\right]$$

backupShared
selectFromPlan
selectIcmOther
selectIcmOther
selectIcmOther
selectAsk

S> `Okay. You asked about price. I need some information. How do you want to travel?`

Explicit user issue rejection The rule in (RULE 3.21) allows the user to reject a system question (by indicating inability to answer, i.e. by uttering "I don't know" or similar). If this is done, the optimistic grounding update is retracted by restoring the shared parts stored in NIM, i.e. QUD and COM, to their previous states.

$$\text{(RULE 3.21)}\quad \begin{array}{ll}
\textsc{rule:} & \textbf{integrateUsrAccNegICM} \\
\textsc{class:} & \mathsf{integrate} \\
\textsc{pre:} & \left\{\begin{array}{l} \$/\textsc{shared}/\textsc{lu}/\textsc{speaker}{==}\mathsf{usr} \\ \mathrm{fst}(\$/\textsc{private}/\textsc{nim},\ \mathsf{icm{:}acc{*}neg{:}issue}) \\ \mathrm{in}(\$/\textsc{shared}/\textsc{pm},\ \mathsf{ask}(Q)) \end{array}\right. \\
\textsc{eff:} & \left\{\begin{array}{l} \mathrm{pop}(/\textsc{private}/\textsc{nim}) \\ \mathrm{add}(/\textsc{shared}/\textsc{lu}/\textsc{moves},\ \mathsf{icm{:}acc{*}neg{:}issue}) \\ /\textsc{shared}/\textsc{qud} := \$/\textsc{private}/\textsc{tmp}/\textsc{qud} \\ /\textsc{shared}/\textsc{com} := \$/\textsc{private}/\textsc{tmp}/\textsc{com} \end{array}\right.
\end{array}$$

The third condition checks that the previous utterance contained an ask move. The final two updates retract the optimistic grounding assumption on the integration / acceptance / reaction level.

Of course, if a question is rejected by the user this may result in a failed database query (unless the alternative database access method described in Section 2.12.4 is used). But how should a system react if the user rejects a system question? In some frame-based dialogue systems for database search (e.g. Chu-Carroll, 2000), fields in the frame can be labelled as obligatory or optional. In IBiS, this corresponds roughly to the distinction between the raise and findout actions; the former has succeeded as soon as the system asks the question, whereas the latter requires the question to be resolved. So if a question which was raised by a raise action was rejected, it will not be asked again. Questions raised by findout actions, however, will currently be raised again by IBiS2 immediately after a user rejection, since the action is still on top of the plan. This is perhaps not very cooperative, and alternative strategies need to be explored. For example, the findout action could be moved further down in the plan so that it will not be asked immediately again, or it may be raised again only if the database search fails.

Dialogue example: explicit user issue rejection In the following dialogue example, the user rejects the system question regarding how to travel. In this example, the plan has been altered so that findout($?x.\mathbf{class}(x)$) has been replaced by raise($?x.\mathbf{class}(x)$), thereby making the class-question optional. Also, the alternative database access method described in Section 2.12.4 is used.

(DIALOGUE 3.11)

S> `What class did you have in mind?`

getLatestMoves
integrateSysAsk

$$\left\{ \begin{array}{l} \text{pop(/PRIVATE/NIM)} \\ \text{push(/SHARED/QUD, } ?A.\mathsf{class}(A)) \end{array} \right.$$

$$\left[\begin{array}{lll}
\textsc{private} & = & \left[\begin{array}{lll}
\textsc{agenda} & = & \langle\langle\rangle\rangle \\
\textsc{plan} & = & \left\langle\begin{array}{l}\mathsf{raise}(?A.\mathsf{class}(A)) \\ \mathsf{consultDB}(?B.\mathsf{price}(B))\end{array}\right\rangle \\
\textsc{bel} & = & \{\} \\
\textsc{tmp} & = & \left[\begin{array}{lll}
\textsc{com} & = & \left\{\begin{array}{l}\mathsf{month(april)} \\ \mathsf{dept\_city(london)} \\ \mathsf{dest\_city(paris)} \\ \mathsf{how(plane)}\end{array}\right\} \\
\textsc{qud} & = & \left\langle\ ?C.\mathsf{price}(C)\ \right\rangle \\
\textsc{agenda} & = & \langle\langle\rangle\rangle \\
\textsc{plan} & = & \left\langle\begin{array}{l}\mathsf{raise}(?A.\mathsf{class}(A)) \\ \mathsf{consultDB}(?B.\mathsf{price}(B))\end{array}\right\rangle
\end{array}\right] \\
\textsc{nim} & = & \langle\langle\rangle\rangle
\end{array}\right] \\
\textsc{shared} & = & \left[\begin{array}{lll}
\textsc{com} & = & \left\{\begin{array}{l}\mathsf{month(april)} \\ \mathsf{dept\_city(london)} \\ \mathsf{dest\_city(paris)} \\ \mathsf{how(plane)}\end{array}\right\} \\
\textsc{qud} & = & \left\langle\begin{array}{l}?D.\mathsf{class}(D) \\ ?C.\mathsf{price}(C)\end{array}\right\rangle \\
\textsc{lu} & = & \left[\begin{array}{lll}\textsc{speaker} & = & \mathsf{sys} \\ \textsc{moves} & = & \langle\langle\ \mathsf{ask}(?D.\mathsf{class}(D))\ \rangle\rangle\end{array}\right] \\
\textsc{pm} & = & \langle\langle\ \mathsf{icm{:}acc{*}neg{:}issue}\ \rangle\rangle
\end{array}\right]
\end{array}\right]$$

U> `it doesnt matter`

getLatestMoves
integrateUsrAccNegICM

$$\left\{\begin{array}{l}
\mathrm{pop}(/\textsc{private}/\textsc{nim}) \\
\mathrm{add}(/\textsc{shared}/\textsc{lu}/\textsc{moves},\ \mathsf{icm{:}acc{*}neg{:}issue}) \\
/\textsc{shared}/\textsc{qud} := \$/\textsc{private}/\textsc{tmp}/\textsc{qud} \\
/\textsc{shared}/\textsc{com} := \$/\textsc{private}/\textsc{tmp}/\textsc{com}
\end{array}\right.$$

exec_consultDB

$$
\left[\begin{array}{lll}
\textsc{pr.} & = & \left[\begin{array}{lll}
\textsc{bel} & = & \left\{\begin{array}{l}
\mathsf{db\_entry(}\left\{\begin{array}{l}\mathsf{month(april)}\\ \mathsf{dept\_city(london)}\\ \mathsf{dest\_city(paris)}\\ \mathsf{how(plane)}\end{array}\right\}\mathsf{,\ \{class(economy)\},\ price(123))}\\
\mathsf{db\_entry(}\left\{\begin{array}{l}\mathsf{month(april)}\\ \mathsf{dept\_city(london)}\\ \mathsf{dest\_city(paris)}\\ \mathsf{how(plane)}\end{array}\right\}\mathsf{,\ \{class(business)\},\ price(1234))}
\end{array}\right\}\\
\textsc{tmp} & = & \left[\begin{array}{lll}
\textsc{com} & = & \left\{\begin{array}{l}\mathsf{month(april)}\\ \mathsf{dept\_city(london)}\\ \mathsf{dest\_city(paris)}\\ \mathsf{how(plane)}\end{array}\right\}\\
\textsc{qud} & = & \left\langle\ ?B.\mathsf{price}(B)\ \right\rangle\\
\textsc{agenda} & = & \langle\langle\rangle\rangle\\
\textsc{plan} & = & \left\langle\begin{array}{l}\mathsf{raise}(?C.\mathsf{class}(C))\\ \mathsf{consultDB}(?D.\mathsf{price}(D))\end{array}\right\rangle
\end{array}\right]\\
\textsc{nim} & = & \langle\langle\rangle\rangle
\end{array}\right]\\
\textsc{sh.} & = & \left[\begin{array}{lll}
\textsc{com} & = & \left\{\begin{array}{l}\mathsf{month(april)}\\ \mathsf{dept\_city(london)}\\ \mathsf{dest\_city(paris)}\\ \mathsf{how(plane)}\end{array}\right\}\\
\textsc{qud} & = & \left\langle\ ?B.\mathsf{price}(B)\ \right\rangle\\
\textsc{lu} & = & \left[\begin{array}{lll}\textsc{speaker} & = & \mathsf{usr}\\ \textsc{moves} & = & \langle\langle\ \mathsf{icm{:}acc{*}neg{:}issue}\ \rangle\rangle\end{array}\right]\\
\textsc{pm} & = & \langle\langle\ \mathsf{ask}(?C.\mathsf{class}(C))\ \rangle\rangle
\end{array}\right]
\end{array}\right]
$$

backupShared
selectRespond
selectAnswer

S> `The price is 123 crowns.  cheap.  The price is 1234 crowns.`
`business class.`

3.6.8 Evidence requirements and implicit grounding

In this section, we discuss evidence requirements for grounding and how these have been implemented in the form of update rules for implicit grounding.

In IBiS2 we use a cautiously optimistic grounding strategy for system utterances. This assumption can be retracted if negative evidence concerning grounding is found. So, what counts as negative and positive evidence? Recall Clark's ranking of different forms of positive evidence, ranging from weakest to strongest:

- Continued attention
- Relevant next contribution
- Acknowledgement: "uh-huh", nodding, etc.
- Demonstration: reformulation, collaborative completion
- Display: verbatim display of presentation

Regarding the attention level, we will not have much to say[13]. The levels of acknowledgement, demonstration and display are presumably what we would regard as explicit feedback, although we have been mainly concerned with the acknowledgement level.

Evidence and relevance

The remaining level in Clark's typology of evidence of understanding is "relevant next contribution". Two questions arise here. First, what counts as a relevant followup? Second, if no relevant followup is produced, should this count as negative evidence of grounding, and if so, on what action level?

A property of dialogue systems sometimes discussed in the literature (The DISC consortium, 1999, Bohlin *et al.*, 1999) is the ability of a system to understand and integrate information different from that which was requested by the system. How does this relate to relevance and grounding? One way to formulate the problem is this: if the system just asked q, and the user's response u did not contain an answer relevant to q or feedback concerning the user's utterance, what should be assumed about the grounding status of q? This is, of course, also a problem that human DPs must resolve; however, Clark does not (to our knowledge) directly discuss this case.

(3.18) a. A: What city do you want to go to? [ask q]
B: I'd like to travel in April [answer other question]

b. A: What city do you want to go to? [ask q]
B: Do you have a student discount? [ask other question]

[13]Clark includes "continued attention" as the weakest form of positive evidence of grounding. However, in principle continued attention from an addressee A after an utterance u is consistent with a complete lack of perception on A's side; A may not even have perceived u but is still waiting for the next utterance. While this example may not be very relevant for human-human communication, it is not a completely unlikely scenario if A is a dialogue system. Also, contact level feedback appears related to this.

Regarding cases where a question is ignored (i.e. neither addressed by a relevant answer, explicitly accepted, nor explicitly rejected), it is not obvious whether the question was accepted or not. The reason is that there are several possible explanations for this behaviour: one complies silently with the question but thinks that other information is more important right now (in which case the question was integrated by the hearer, and will be answered eventually), or one misheard or did not hear the question at all (in which case it was not understood, and thus neither accepted or rejected), or one does not think that the question is appropriate (in which case the question was implicitly rejected).

One possible strategy for finding negative evidence is to look for signs of misunderstanding, and to try to come up with a plausible explanation for how this misunderstanding came about. This is, however, a fairly difficult task even for humans and not one we intend to explore here.

Cases where a question is not followed by a relevant answer or relevant ICM, can be regarded as implicit rejections of that question. However, if the followup is relevant in some other way to the question asked, this should not be regarded as rejection. One type of relevant followup can be defined using Ginzburg's notion of question dependence:

(3.19) An ask-move with content q is a relevant followup to an ask-move with content q' if q' depends on q.

In Section 2.8.2, we defined a domain-dependent notion of question dependence related to terms of plans, where q' depends on q if there is a plan for dealing with q' which includes an action findout(q).

Consequently, in IBiS2 we have chosen the following requirements on an utterance u to count as an irrelevant followup to an utterance raising a question q:

- u contains no ICM
- the previous move raised a question q
- u contains no answer to q
- u contains no ask-move raising a question q' such that q depends on q'

Concerning our second question, are irrelevant followups to be regarded as negative grounding evidence? Or could it be the case that a DP understood and accepted an utterance u but opted to change the subject temporarily, planning to respond to u eventually?

If the irrelevant followup is interpreted as negative grounding evidence, how do we know what action level is concerned? Did the user implicitly reject the issue by ignoring it, or did she simply not perceive or understand it? We suspect that the choice between these two interpretations might depend on quite subtle issues concerning timing. For example, if the user's followup overlaps with the system's question it is possible that the user has not even heard the system's question.

In IBiS2 we have chosen to consider irrelevant followups to system ask moves as implicit rejections. However, this choice is not obvious and is a further topic for future research.

Implicit user rejection of issue

If an irrelevant followup is detected, this is interpreted as an implicit issue rejection and consequently the optimistic assumption that the question q' was integrated by the user is assumed to be mistaken. Therefore, the optimistic assumption is retracted by reverting the previous shared state for the relevant parts of SHARED.

(RULE 3.22)
```
RULE:  irrelevantFollowup
CLASS: none
PRE:   1 $/PRIVATE/NIM=Moves
       2 $/SHARED/LU/SPEAKER==usr
       3 not A/ELEM=icm:_
       4 in($/SHARED/PM, PrevMove)
       5 PrevMove=ask(Q) or
           ( PrevMove=icm:und*int:DP*C and Q=und(DP*C) )
       6 not Moves/ELEM=ask(Q') and $DOMAIN :: depends(Q, Q')
       7 not A/ELEM=answer(A) and $DOMAIN :: relevant(A, Q)
EFF:   /SHARED/QUD := $/PRIVATE/TMP/QUD
       /SHARED/COM := $/PRIVATE/TMP/COM
```

(Since this rule is called "by name" from the update algorithm, there is no need for including it in a rule class.) Condition 3 checks that no ICM was included in the latest move. Condition 4 and 5 tries to find a question Q raised by the previous move, possibly an understanding-question. Note here that we do not check QUD; in IBiS2, questions remain on QUD only for one turn but it may be the case that we want questions to remain on QUD over several turns. What we are interested here is thus not which questions are on QUD but which questions were raised by the previous utterance, and this is the reason for looking in PM rather than QUD. Conditions 6 and 7 check that no move performed in the latest utterance is relevant to Q, neither by answering it nor by asking a question on which Q depends. The updates are similar to those for integration of negative acceptance feedback (Section 3.6.7).

As is the case for explicit rejections, questions raised by findout actions will be asked again, but questions raised by raise actions will not. ICM-related questions (interrogative understanding feedback) are not repeated since they are not in the plan but only on the agenda.

A dialogue involving implicit user rejection of an issue will be shown later in (DIALOGUE 3.12).

3.6.9 Sequencing ICM: reraising issues and loading plans

In this section, we review sequencing-related ICM and show how this has been implemented in IBiS2.

We believe it is good practice to try to keep the user informed about what's going on inside the system, at least to a degree that facilitates a natural dialogue where system utterances "feel natural". One way of doing this is to produce ICM phrases indicating significant updates to the information state which are not directly related to specific user utterances. Using Allwood's (1995) terminology, we refer to these instances of ICM as "sequencing ICM".

For IBiS2, we will implement two types of sequencing ICM. First, when loading a plan IBiS2 will indicate this. Second, IBiS2 will produce ICM to indicate when an issue is being reraised (in contrast to being raised for the first time).

Loading plans

IBiS2 will indicate when a plan is being loaded, thus preparing the user to answer questions. This is currently generated as "Let's see."

The rule for finding an appropriate plan to deal with a respond-action on the agenda is similar to that in IBiS1. The difference is that the IBiS2 rule produces ICM to indicate that it has loaded a plan, formalized as icm:loadplan and generated e.g. as "Let's see". Again, the choice of output form is provisory.

(RULE 3.23) RULE: **findPlan**
CLASS: load_plan

$$\text{PRE: }\left\{\begin{array}{l}\text{in}(\$/\text{PRIVATE}/\text{AGENDA},\ \mathsf{respond}(Q))\\ \$\text{DOMAIN} :: \text{plan}(Q,\ Plan)\\ \text{not in}(\$/\text{PRIVATE}/\text{BEL},\ P)\text{ and }\$\text{DOMAIN} :: \text{resolves}(P,\ Q)\end{array}\right.$$

$$\text{EFF: }\left\{\begin{array}{l}\text{del}(/\text{PRIVATE}/\text{AGENDA},\ \mathsf{respond}(Q))\\ \text{set}(/\text{PRIVATE}/\text{PLAN},\ Plan)\\ \text{push}(/\text{PRIVATE}/\text{AGENDA},\ \mathsf{icm{:}loadplan})\end{array}\right.$$

This rule is identical to that in IBiS1 (Section 2.8.6), expect for the final update which pushes the icm:loadplan move on the agenda.

Reraising issues

System reraising of issue associated with plan If the user raises a question Q and then raises Q' before Q has been resolved, the system should return to dealing with Q once Q' is resolved; this was described in Section 3.6.9. The **recoverPlan** rule in IBiS2, shown in (3.20), is almost identical to the one in IBiS1, except that ICM is produced to indicate that an issue ($q1$) is being reraised. This ICM is formalized as icm:reraise:q where q is the question being reraised, and expressed e.g. as "Returning to the issue of price".

(RULE 3.24) RULE: **recoverPlan**
CLASS: load_plan

$$\text{PRE: }\left\{\begin{array}{l}\text{in}(\$/\text{SHARED}/\text{QUD},\ Q)\\ \text{is\_empty}(\$/\text{PRIVATE}/\text{AGENDA})\\ \text{is\_empty}(\$/\text{PRIVATE}/\text{PLAN})\\ \$\text{DOMAIN} :: \text{plan}(Q,\ Plan)\\ \text{not in}(\$/\text{PRIVATE}/\text{BEL},\ P)\text{ and }\$\text{DOMAIN} :: \text{resolves}(P,\ Q)\end{array}\right.$$

$$\text{EFF: }\left\{\begin{array}{l}\text{set}(/\text{PRIVATE}/\text{PLAN},\ Plan)\\ \text{push}(/\text{PRIVATE}/\text{AGENDA},\ \mathsf{icm{:}reraise{:}}Q)\\ \text{push}(/\text{PRIVATE}/\text{AGENDA},\ \mathsf{icm{:}loadplan})\end{array}\right.$$

Issue reraising by user In the case where the user reraises an open issue, an icm:reraise:Q move is selected by the **integrateUsrAsk** described in Section 3.6.6.

System reraising of issue not associated with plan The IBiS1 **reraiseIssue** rule described in Section 2.12.3 reraises any questions on QUD which are not associated with any plan (i.e. which have been raised previously by the system). In this case it is again helpful to indicate that the system is aware that the issue is being reraised. However, since the issue will be reraised, the sequencing ICM does not need to indicate which question is being reraised.

(RULE 3.25) RULE: **reraiseIssue**
CLASS: select_action
PRE: $\left\{\begin{array}{l}\text{fst}(\$/\text{SHARED}/\text{ISSUES},\ Q)\\ \text{not } \$\text{DOMAIN} :: \text{plan}(Q,\ P)\end{array}\right.$
EFF: $\left\{\begin{array}{l}\text{push}(/\text{PRIVATE}/\text{AGENDA},\ \mathsf{icm{:}reraise})\\ \text{push}(/\text{PRIVATE}/\text{AGENDA},\ \mathsf{raise}(Q))\end{array}\right.$

The conditions of this rule checks that there is a question Q on ISSUES for which the system has no plan, i.e. one that the system needs to ask the user.

The first update adds an icm:reraise (without an argument) to signal that it is reraising a question; this is currently generated as prefixing "so" to the next ask move, which is an ordinary raising of the question (placed on the agenda by the second update in the rule). In a more sophisticated implementation one could consider abbreviating the original raising of the question to make an appropriate reraising, e.g. "So, from what city?" But we do not have a general method for doing that yet, even in these simple dialogues. To do it properly would probably require more detailed syntactic and semantic analysis of contributions than we currently have in IBiS.

Dialogue example: user and system reraising (DIALOGUE 3.12) is an example of a dialogue where the user reraises an issue which was previously raised but not resolved.

Also subsequently the system reraises an issue which is relevant for addressing the issue that the user reraised. In order to address the issue of price the system needs to get information from the user concerning where she wishes to travel from.

(DIALOGUE 3.12)

```
S> Welcome to the travel agency!
U> price information please
S> Okay.  Lets see.  How do you want to travel?
U> a flight
S> Okay.  What city do you want to go to?
U> paris
S> Okay.  What city do you want to go from?
```

U> `right... actually, do i need a visa`
user accepts departure city issue, then raises visa issue
S> `Okay. Lets see. What country are you from?`
U> `um... lets talk about price again`
user implicitly rejects (avoids) citizenship issue, reraises price issue
S> `Okay. Returning to the issue of price. Lets see. So, What city do you want to go from?`
system indicates returning to price issue, then reraises departure city issue
U> `london`
S> `Okay. What month do you want to leave?`
U> `april`
S> `Okay. What day do you want to leave?`
U> `fifth`
S> `Okay. What class did you have in mind?`
U> `cheap`
S> `Okay. The price is 123 crowns. Returning to the issue of visa. Let's see. What country are you from?`
system answers price issue, thereby resolving it, then reraises the visa issue, indicates plan loading, and reraises citizenship issue

The first case of reraising here, where it is the user who reraises the price issue, is handled by the conditional update of the **integrateUsrAnswer** (Section 3.6.6) which selects an icm:reraise:Q move to signal that the system is aware that issue was already open by saying "You reraised the issue of price". In the same utterance, the system reraises the issue of where the user wants to travel from, requires a selection rule for the system. When reraising an issue, IBiS2 produces ICM to indicate awareness that the issue has been raised before. This ICM is formalized as icm:reraise and can be realized e.g. by the discourse particle "So". Note that this would not have happened if the user had not accepted this question (by saying "right") when it was first raised. Since the system does not regard the departure city question as dependent on the visa issue, raising the visa issue in response to asking for departure city would have been regarded as an implicit rejection (Section 3.6.8).

Once the price issue has been resolved, the system reraises the visa issue which is still unresolved; this is done by the **recoverPlan** rule as described in Section 3.6.9.

3.7 Further implementation issues

In this section we describe parts of the implementation of IBiS2 which have not been discussed earlier in this chapter, and which are not directly reused from IBiS1.

3.7.1 Update module

The IBiS2 update algorithm is shown in (3.20).

(3.20)
```
1 if not LATEST_MOVES == failed
2 then ⟨ getLatestMove,
3          try irrelevantFollowup,
4          repeat integrate,
5          try load_plan,
6          repeat manage_plan
7          try downdate_qud ⟩
8 else try unclearFollowup
```

Line 1 checks that the interpretation of the latest utterance was successful (of course, in the case of system utterances this is always true). If not, the **unclearFollowup** rule described in Section 3.6.8 is tried. If interpretation was successful, the latest moves are incorporated in the information state proper by the **getLatestMoves** rule (see Section 3.6.7). Before integration starts, the **irrelevantFollowup** rule described in Section 3.6.8 is tried to catch cases where a system question has been ignored by the user. After this, the integration rule class is repeatedly applied until the system has tried to integrate all moves in /PRIVATE/NIM. If the user asked a question, the appropriate plan will be loaded by line 5. Any loaded plan is executed by applying the exec_plan rule class until no more execution is possible at the current stage of the dialogue.

3.7.2 Selection module

As in IBiS1, action selection rules add actions to the agenda. However, while in IBiS1 only one action was selected per turn, in IBiS2 several actions may be selected per turn. For example, the selectRespond in IBiS2, shown in (RULE 3.26), does not require the agenda to be empty, but only that the respond action has not already been selected, and thus it allows several moves to be selected per turn.

(RULE 3.26) RULE: **selectRespond**
CLASS: select_action

$$\text{PRE:} \left\{\begin{array}{l} \text{is\_empty}(\$/\text{PRIVATE}/\text{PLAN}) \\ \text{fst}(\$/\text{SHARED}/\text{QUD},\ A) \\ \text{in}(\$/\text{PRIVATE}/\text{BEL},\ B) \\ \text{not in}(\$/\text{SHARED}/\text{COM},\ B) \\ \$\text{DOMAIN} :: \text{resolves}(B,\ A) \\ \text{not in}(\$/\text{PRIVATE}/\text{AGENDA},\ \mathsf{respond}(A)) \end{array}\right.$$

$$\text{EFF:} \left\{\begin{array}{l} \text{push}(/\text{PRIVATE}/\text{AGENDA},\ \mathsf{respond}(A)) \end{array}\right.$$

Similarly, the move selection rules in IBiS2 are repeatedly applied, popping actions off the AGENDA queue and pushing the corresponding moves on NEXT_MOVES. As an example, the **selectAnswer** rule is shown in (RULE 3.27).

(RULE 3.27) RULE: **selectAnswer**
CLASS: select_move

$$\text{PRE:} \left\{\begin{array}{l} \text{fst}(\$/\text{PRIVATE}/\text{AGENDA},\ \mathsf{respond}(A)) \\ \text{in}(\$/\text{PRIVATE}/\text{BEL},\ B) \\ \text{not in}(\$/\text{SHARED}/\text{COM},\ B) \\ \$\text{DOMAIN} :: \text{resolves}(B,\ A) \end{array}\right.$$

$$\text{EFF:} \left\{\begin{array}{l} \text{push}(\text{NEXT\_MOVES},\ \mathsf{answer}(B)) \\ \text{pop}(/\text{PRIVATE}/\text{AGENDA}) \end{array}\right.$$

The selection algorithm for IBiS2 is shown in (3.21).

```
(3.21) ⟨ backupShared,
       if not in($/PRIVATE/AGENDA, A) and q_raising_action(A)
       then try select_action,
       repeat ( select_icm orelse select_move ) ⟩
```

The select_action rule class selects actions and places them on the AGENDA, whereas the select_move and select_icm rule classes selects AGENDA items and places them on NEXT_MOVES. Before selection, the **backupShared** (Section 3.6.7) is applied to copy relevant parts of the information state to /PRIVATE/NIM.

The basic strategy for selection in IBiS is that only one question should be raised by the system in each utterance. The IBiS2 selection algorithm first checks if some question-raising action is already on the agenda; if not, it tries to select a new action. Then, it selects moves and ICM repeatedly until nothing more can be selected.

The "q_raising_action(A)" condition uses a macro condition (see Section A.4.2) whose definition is shown in (3.22). What this says is, basically, that interrogative ICM, raise and, findout actions raise questions.

(3.22) q_raising_action($Move$) if
$Move$ = icm:und*int:X or $Move$ = raise(X) or $Move$ = findout(X)

3.8 Discussion

3.8.1 Some grounding-related phenomena not handled by IBiS2

In this section we mention some areas which have not been accounted for in the issue-based approach presented here. We do not by any means claim that this list is complete.

Perhaps the most significant omission in IBiS2 is a treatment of semantic ambiguity, e.g. ambiguous words. A possible direction of research in this area is to handle semantic ambiguity on a pragmatic level. Specifically, the relevance of an ambiguous move in the current dialogue context may be sufficient to resolve the semantic ambiguity, or at least reduce the number of possible semantic interpretations. In any case, we see no reason that mechanisms similar to those for dealing with pragmatic ambiguity could be used for semantic ambiguity.

Another area that remains unexplored from the point of view of issue-based dialogue management is semantic vagueness. For instance, one might want a system to understand vague answers (e.g. "I want to go to southern France", "I want to travel around the 10th of April"), and perhaps also to ask less specific questions which leave more room for the user to choose how to specify e.g. parameters for database search (e.g. "Where do you want to travel?" rather than "What city do you want to go to?").

On the pragmatic understanding level, we have concentrated on ellipsis resolution and relevance, however we are still lacking a treatment of referent resolution. One reason for this is of course that IBiS2 does not represent referents. This is a fairly well-researched area, and we hope to be able to include some existing account of referent resolution when this becomes necessary.

Overlapping user feedback and and barge-in

Most dialogue systems do not handle feedback from the user in any form, and most (if not all) existing systems which handle barge-in will stop talking if they perceive any sound from the speaker. This means that even positive feedback (e.g. "uhuh") from the user will cause the system to stop speaking. This problem is aggravated in noisy environments, where noises may be misinterpreted as speech from the user and cause a system to stop speaking. What is needed is clearly that the system makes a distinction between different kinds of feedback from the user; positive feedback should usually not cause the system to stop speaking.

Mechanisms for handling overlapping user feedback has been explored within the GoDiS framework (Berman, 2001), but are not included here. However, the inclusion of positive user feedback in IBiS provides a basis for further explorations in this area.

3.8.2 Towards an issue-based account of grounding and action levels

We have hinted that a full-coverage account of grounding should include grounding on all four action-levels. Ginzburg's content- and acceptance-questions indicate how this could be accomplished in an issue-based theory of dialogue. For each action level, grounding issues can be raised and addressed; feedback moves on level L are regarded as addressing grounding issues on level L.

This would allow grounding to be handled by the same basic update mechanisms as questions and answers. A distinction can be made between short (elliptical, underspecified) answers (feedback utterances whose action level is not explicit) and full answers (feedback utterances whose action level is clear from the form and content of the utterance).

In IBiS, we strive for simplicity at the cost of completeness; however, the account given here can be seen as a first step towards a more complete issue-based account of grounding an action levels in dialogue. A sketch of a more complete account can be found in Section 6.5.1.

3.8.3 Comparison to Traum's computational theory of grounding

Traum (1994) provides a computational account of grounding based on a combination of finite automata and cognitive modelling. This model builds on Clark and Schaefer (1989b) but attempts to solve some computational problems inherent in that account.

Traum argues that Clark's account of the presentation and acceptance phases is problematic from a computational point of view. Firstly, it may be hard to tell if a speech signal is part of the presentation or acceptance phase. Second, it is hard to know when a presentation or acceptance is finished; often, this is only possible in hindsight, which may cause problems for a dialogue system engaged in real-time spoken dialogue. Third, it is unclear whether grounding acts (in our terminology, ICM dialogue moves) themselves need to be grounded.

Regarding the last point, we follow Traum in assuming that ICM moves do not need to be grounded. In fact, on our view this amounts to an optimistic grounding strategy where ICM moves are concerned.

We agree that in general the problem of deciding when a contribution ends is one that should be handled as a part of dialogue management, and that something like Traum's atomic grounding acts are needed for this. However, for the time being we make the simplifying assumption that contributions are already segmented before dialogue management starts; in the implementation, we rely on the speech recognizer's built-in algorithms for deciding when an utterance is finished.

Our account does not address the first point, i.e. the problem of jointly produced contributions, where DPs e.g. can repair each other's utterances. Traum proposes a recursive transition network (RTN) model of the grounding process which includes repairs, requests for repairs, acknowledgements and requests for acknowledgements (a simpler finite state model is also provided). Our account does not include repairs or requests for acknowledgements; however, Traum's acknowledgements correspond roughly to positive feedback and requests for repairs correspond (very) roughly to negative feedback.

It is important to note here that Traum (1994) uses the term "grounding" to refer exclusively to what we call "understanding-level grounding". It is notable that Tram focuses almost exclusively on positive feedback, whereas negative feedback is given a less detailed treatment. The grounding act most closely corresponding to negative feedback is request for repair; however, it is doubtful whether all negative feedback can be regarded as requests (e.g. "I don't understand"). To use Allwood's terminology, feedback has both an expressive dimension (expressing lack of perception, understanding, acceptance) and an evocative dimension (requesting a "repair" or repetition/reformulation). It appears that Traum has focused more on the evocative dimension whereas we have been more concerned with the expressive dimension. (We do feel that the evocative aspect of feedback is something that perhaps deserves more attention than we have given it so far; this is yet another area for future research.)

The dialogue acts "accept" and "reject" are regarded by Traum as "Core Speech Acts" on the same level as assertions, asking questions, giving instructions etc. The accept act is defined as "agreeing to a proposal" (p.58), which gives the impression that an acceptance act is a natural followup to some proposal act.

However, Traum's acceptance act also has similarities to what we refer to as positive reaction-level feedback. For example, an acceptance move may follow an assertion or an instruction, and the effects of the accept act is to change the status of the content of the assertion or instruction from being merely proposed to actually being shared. Regarding questions, it is unclear whether they need to be accepted before being shared. According to Traum, asking a question imposes an obligation on the addressee to address the question, i.e. to either answer it or to reject it. This seems (although it is far from clear) to indicate that questions on Traum's account are optimistically assumed to be grounded whereas assertions and instructions are not.

In Chapter 5, we extend the issue-based account of dialogue to negotiative dialogue, and argue that two kinds of acceptances need to be distinguished: acceptance as positive feedback on the reaction level, and acceptance of a proposed alternative solution to some problem (e.g. a certain domain plan as one among several ways to reach some goal).

3.9 Summary

After providing some dialogue examples where various kinds of feedback are used, we reviewed some relevant background, and discussed general types and features of feedback as it appears in human-human dialogue. Next, we discussed the concept of grounding from an information update point of view, and introduced the concepts of optimistic, cautious and pessimistic grounding strategies. We then related grounding and feedback to dialogue systems, and discussed the implementation of a partial-coverage model of feedback related to grounding in IBiS2. This allows the system to produce and respond to feedback concerning issues dealing with the grounding of utterances.

Chapter 4

Addressing unraised issues

4.1 Introduction

In the previous chapter, we discussed various mechanisms for handling grounding. One of the action levels to which grounding applies is that of pragmatic understanding, i.e. making sense of the meaning of an utterance in the current dialogue context. Some basic mechanisms for grounding on the understanding level were implemented in IBiS2. However, the kinds of dialogues handled by this system are still rather rigid and system-controlled.

The aim of the current chapter is to enable more flexible dialogue. After reviewing some shortcomings of IBiS2, we take a closer look at the notions underlying the QUD data structure, which results in dividing QUD into two substructures, one global and one local. Next, the notion of question accommodation is introduced to allow the system to be more flexible in the way utterances are interpreted relative to the dialogue context. Among other things, question accommodation allows the system to understand answers to questions which have not yet been asked, and to understand such answers even before any issue has been explicitly raised. In cases of ambiguity, clarification dialogues may be needed. Question accommodation combined with (very basic) belief revision abilities also allows IBiS to reaccommodate questions which have previously been resolved. Finally, a version of reaccommodation, where reaccommodation of one issue requires reaccommodation of a dependent issue as well, allows for successive modifications of database queries.

The division of QUD into a global and a local structure also enables a simple accommodation mechanism allowing the user to correct the system in cases where explicit positive feedback shows that the system has misunderstood a user utterance.

Apart from the initial and final sections, this chapter is structured around the various question accommodation mechanisms. For each type of accommodation, there is an informal description, a formalization consisting of one or more update rules, and dialogue examples.

4.2 Some limitations of IBiS2

Handling answers to unasked questions

The dialogue structure allowed by the IBiS2 system is rather rigid and system-controlled. The main part of the dialogue consists of the system asking questions which the user has to answer. The user is not allowed to give more information, or different information, than what the system has just asked for.

In general, we require that the content of each answer-move must match a question on QUD. In IBiS2, the only way questions can end up on QUD is by being explicitly asked. This forces a simple tree structure on dialogue. In real dialogue, however, people often perform utterances which can be seen as answers to questions, or addressing issues, which have not yet been raised.

Revising information

Once the user has supplied some information to IBiS2, this information cannot be changed. This is clearly undesirable, and solving this problem would provide several advantages:

- The user may change his mind during the specification of the database query
- After the user has been given e.g. price information for a specified trip, he can modify some of the information to produce a new query, without having to enter all information again

Correcting explicit positive feedback

An important factor influencing the choice of feedback and grounding strategies in a dialogue system is usability (including efficiency of dialogue interaction). A disadvantage of the confirmation-question approach is that the dialogue becomes slow and tiring for the user, which decreases the efficiency and usability of the system.

For this reason, in the previous chapter we added the capability of producing feedback on the understanding level in non-eliciting form, i.e. as a declarative or elliptical sentence (without question intonation). However, this solution is unsatisfactory since the system may be mistaken and there is no way to correct it. A very natural response to positive explicit feedback which indicates a misunderstanding is to protest, e.g. by saying "no!", possibly followed by a correction.

In this chapter, we use a special case of question accommodation to allow this, thus extending the issue-based account of grounding. If the user is satisfied with the system's interpretation, she does not have to do anything; the system will eventually continue (possibly after a short pause) with the next step in the dialogue plan. The user's silence is regarded as an implicit compliance with the system's feedback. There is also the option of giving an explicit positive response to the feedback (e.g. "yes" or "right"). Finally, if the user responds negatively to the system's feedback (e.g. by saying "no"), the system will understand that it misunderstood the user and act accordingly.

4.3 The nature(s) of QUD

Before extending the capabilities of IBiS, we will investigate the nature of QUD and make some distinctions between the different tasks that QUD can be used for. We will draw the conclusion that QUD needs to be divided into two substructures, one global and one local.

In this section, we present and compare some alternative notions of QUD.

4.3.1 Ginzburg's definition of QUD

In Ginzburg (1997), Ginzburg provides the following definition of QUD:

> QUD ('questions under discussion'): a set that specifies the currently discussable questions, partially ordered by $\prec$ ('takes conversational precedence'). If q is maximal in QUD, it is permissible to provide any information specific to q using (optionally) a short-answer. (Ginzburg, 1997, p. 63)

While the definition above merely states that QUD is a partially ordered set, the operations performed on QUD in Ginzburg's protocols suggest that in fact it is more like a partially ordered stack[1]. In IBiS1 and IBiS2 we made the simplification that QUD is simply a stack.

Ginzburg thus uses a single structure to do two jobs: (1) specifying the questions that are currently available for discussion ("open" questions), and (2) specifying the questions that can be addressed by a short answer (namely, those that are QUD-maximal).

Based on Ginzburg's QUD querying protocol (Section 2.8.2), Ginzburg's QUD can also be said to (3) represent questions which have been explicitly raised in the dialogue. While this is not explicitly stated, it appears that the only way a task-level question can enter QUD on Ginzburg's account is by being explicitly asked. (However, grounding-related questions may enter QUD without being raised as part of the internal reasoning of a DP; see Section 3.2.2).

Similarly, the QUD downdate protocol (Section 2.8.4) suggests that QUD also fulfills a further property (4) of containing as-yet unresolved questions. Openness and unresolvedness may not be identical properties; arguably, resolved questions may still to some extent be open for discussion, and a question could be discarded from the open issues without being resolved, e.g. if it becomes irrelevant (Larsson, 1998).

To summarize, given this basic characterization of QUD, we can say that questions on QUD are

1. open for discussion,
2. available for ellipsis resolution,
3. explicitly raised, and
4. not yet resolved.

In IBiS2, the implemented QUD essentially fits with Ginzburg's definition, except for the simplification that it is a plain stack rather than an open, partially ordered stack. This is sufficient for the relatively system-controlled, rigid dialogue handled by IBiS2. When the dialogue structure becomes more flexible, however, these various properties of the QUD listed above no longer appear to co-occur in all situations.

[1]A partially ordered stack would be a structure where elements can be pushed and popped, but which only has a partial ordering. For example, more than one element can be topmost on the stack.

4.3.2 Open questions not available for ellipsis resolution

Regarding QUD as a stack (or stack-like) structure suggests that when the topmost element (or set of elements) is popped off the stack, the element (or set of elements) that was previously next-to-maximal becomes maximal. This implies that questions can be answered elliptically at an arbitrary distance from when they were raised. However, it can be argued that in many cases a question which has been raised a few turns back is no longer available for ellipsis resolution (or at least significantly less available than it was right after the question was raised). For example, B's final utterance in the made-up dialogue in (4.1) is unlikely to occur and it would be rather confusing if it did, simply because it is not clear which question B is answering.

(4.1) A : Who's coming to the party?
B : That depends, is Jill coming
A : Jill Jennings?
B : Yes
A : By the way, did you hear about her brother? What's his name anyway?
B : Umm.. I'm not sure. Anyway, I'd rather not talk about it.
A : OK. So, No
B : So, Jim

If this argument is accepted, we see that a question may satisfy requirements (1) and (3) above, to be a currently open for discussion, explicitly raised question, while not satisfying property (2) of being available for ellipsis resolution.

4.3.3 Open but not explicitly raised questions

Studying recorded travel agency dialogues in light of the QUD approach indicates that it may be the case that a question which has not been (explicitly) raised is in fact discussable, and even available for ellipsis resolution, as the dialogue in (4.2) shows[2].

(4.2) A : When do you want to travel?
B : April, as cheap as possible

Thus, we can observe that a question may satisfy requirements (1) of being open for discussion and (2) of being available for ellipsis resolution, without satisfying property (3) of having been explicitly raised.

[2]This is a simplified version of the dialogue in example 4.6.

4.3.4 Global and local QUD

The observations above suggest that it is not ideal to model QUD using a single structure satisfying properties (1) to (4). The solution we propose is to divide QUD into a global and a local structure; the former satisfying property (1) of being open for discussion and (4) of being unresolved, and the latter satisfying property (2) of being available for ellipsis resolution. Property (3) of being explicitly raised is not satisfied by either structure. This enables more flexible ways of introducing questions into a dialogue. This division of labour also appears to allow the use of simpler data structures than partially ordered sets.

Definition of local QUD

For the local QUD, a set seems appropriate for modelling the questions currently available for ellipsis resolution. A stack-like structure would suggest e.g. the (made-up) dialogues (4.3) should be easily processed by DPs, but in fact it is very unclear what B means.

(4.3) A : Where are you going? Where is your wife going?
B : Paris. London.

Also, consider example (4.4):

(4.4) A : When are you leaving? When are you coming back?
B : ten thirty and eleven thirty

A simple stack structure also suggests a very unintuitive interpretation of B's answer, where 10:30 is the time when B is coming back and 11:30 is the time when B is leaving. It appears that among the constraints guiding ellipsis resolution in cases where multiple questions are available, the order in which the questions were asked is not very significant. Of course, Ginzburg realizes this and this appears to be the main reason for letting QUD be a partially ordered set where several internally unordered elements may be topmost on QUD, and thus available for ellipsis resolution.

In IBiS3 we define QUD to be an open stack of questions that can be addressed using short answers. The reason for using an open stack is that it has the set-like properties we want, but also retains a stack structure in case it should be useful for ellipsis resolution.

Definition of global QUD, or "Live Issues"

The global QUD contains all questions which have been raised in a dialogue (explicitly or implicitly) but not yet resolved. It thus contains a collection of current, or "live" issues. A suitable data structure appears to be an open stack, i.e. a stack where non-topmost elements can be accessed. This allows a non-rigid modelling of current issues and task-related dialogue structure.

4.3.5 Some other notions of what a QUD might be

In fact, there are some additional notions of what QUD might be, all of which in some sense contain questions that are under discussion, and all of which have potential uses in a theory of dialogue management and in a dialogue system.

- closed issues: questions that have been raised and resolved (see Section 3.6.9)
- raisable domain issues: all issues potentially relevant in regard to the domain
- potential grounding issues: all issues pertaining to grounding of (a) recent utterance(s)
- resolvable issues (for a DP): all issues that a DP knows some way of dealing with, either by answering directly or by entering a subdialogue

However, while all these may be useful, it may not be necessary to model them explicitly as separate structures in a dialogue system. For example, "raisable domain issues" and "resolvable issues" may be derived from the (static) domain knowledge.

Regarding "closed issues", we can to some extent derive them from the shared commitments by checking which issues are resolved by propositional information, as in (4.5).

(4.5) Q is a closed issue iff there is some $P \in$ /SHARED/COM such that P resolves Q and P does not resolve any other question (in the domain)

However, this only works as long as each proposition resolves a unique issue. If this is not true, a separate store of closed issues is needed e.g. for detecting reraisings of previously discussed issues.

4.4 Question Accommodation

In this section, we introduce the concept of accommodation and show how it can be extended to handle accommodation of questions in various ways. We also show how question accommodation can be implemented in IBiS.

4.4.1 Background: Accommodation

Lewis' notion of accommodation

David Lewis, in Lewis (1979), in discussing the concept of a conversational scoreboard, compares conversation to a baseball game:

> ...conversational score does tend to evolve in such a way as is required in order to make whatever occurs count as correct play (Lewis, 1979, p. 347)

He also provides a general scheme for rules of *accommodation* for conversational score:

> If at time t something is said that requires component s_n of conversational score to have a value in the range r if what is said is to be true, or otherwise acceptable; and if s_n does not have a value in the range r just before t; and if such-and-such further conditions hold; then at t the score-component s_n takes some value in the range r. (Lewis, 1979, p. 347)

This very general schema can be used for dealing with definite descriptions, presupposition projection (see e.g. van der Sandt, 1992), anaphora resolution, and many other pragmatic and semantic problems.

One motivation for thinking in terms of accommodation has to do with generality. We could associate expressions which introduce a presupposition as being ambiguous between a presuppositional reading and a similar reading where what is the presupposition is part of what is asserted. For example, an utterance of “The king of France is bald” can be understood either as an assertion of the proposition that there is a king of France and he is bald, or as an assertion of the proposition that he is bald with the presupposition that there is a king of France and that “he” refers to that individual. However, if we assume that accommodation takes place before the integration of the information expressed by the utterance then we can say that the utterance always has the same interpretation.

4.4.2 Accommodation, interpretation, and tacit moves

In an information update framework, accommodation is naturally implemented as an update rule which modifies the information state to include the information presupposed by an utterance, in such a way as to make the utterance felicitous, i.e. to make it possible to understand the relevance of and possibly integrate the move(s) associated with the utterance. The accommodation update acts as a replacement for a dialogue move, which would have prepared the (common) ground for the utterance actually performed. For this reason, accommodation updates may be referred to as a kind of *tacit move*. For example, the silent accommodation move which adds "there is a king of France" to allow the integration of "The king of France is bald" corresponds to a dialogue move asserting this proposition.

Thus, we can simplify our dialogue move analysis so that the updates to the information state normally associated with a dialogue move are actually carried out by tacit accommodation moves.

This fits well with the fact that very few (if any) effects of a dialogue move are guaranteed as a consequence of performing the move; rather, the actual resulting updates depend on reasoning by the addressed participant. Accommodation is one type of reasoning involved in understanding and integrating the effects of dialogue moves.

4.4.3 Extending the notion of accommodation

In this section, we extend the notion of accommodation introduced by Lewis to cover accommodation of Questions Under Discussion.

As defined by Lewis, accommodation is not limited to only propositions[3]. It states that any component of the scoreboard can be modified by accommodation. If we carry this over to the issue-based approach to dialogue management, it follows that in addition to the accommodation of propositions to the set of jointly committed propositions, questions can be accommodated to QUD.

Thus, question accommodation can be exploited to provide an explanation of the fact that questions can be addressed (even elliptically) without having been explicitly raised. This is very relevant for a dialogue system, since it allows the user more freedom regarding when and how to provide information to the system. In addition, the related concept of

[3]Of course, all information on the DGB is, in the end, propositional in nature; a DGB containing a question Q at the top of QUD could in principle be described by a set of propositions including "Q is topmost on QUD". This, however, would be impractical and inefficient compared to maintaining a proper stack-like structure.

question *reaccommodation* can be used to enable addressing resolved issues, which among other things provides a way of handling revision of jointly committed propositions in a principled manner.

Before proceeding to explore the exact formulation and formalization of question accommodation, we provide a rough characterization of the notions used:

- question/issue accommodation: adjustments of common ground required to understand an utterance addressing an issue which has not been raised, but which is
  - relevant to the current dialogue plan
  - relevant to some issue in the domain
- question/issue reaccommodation: adjustments of common ground required to understand an utterance addressing an issue which has been resolved and
  - does not influence any other resolved issue
  - influences another resolved issue
  - concerns grounding of a previous utterance

Utterances which are relevant to the current dialogue plan can also be regarded as being *indirectly relevant* to the goal of that plan. In inquiry-oriented dialogue we model goals as issues which allows an alternative formulation of accommodation as "adjustments of common ground required for understanding an utterance addressing an issue which has not been raised, but which is (directly or indirectly) relevant to some issue in the domain." In action-oriented dialogue (Chapter 5), utterances may also be indirectly relevant to some goal action.

4.5 Formalizing question accommodation

In this section we discuss the various types of question accommodation and show how they are formalized in IBiS3. We start by explaining and motivating some modifications of the information state type required to handle dialogues involving question accommodation.

4.5.1 Information state in IBiS3

The information state used in IBiS3 is shown in Figure 4.1.

$$
\left[\begin{array}{lll}
\textsc{private} & : & \left[\begin{array}{lll}
\textsc{agenda} & : & \text{OpenQueue(Action)} \\
\textsc{plan} & : & \text{OpenStack(PlanConstruct)} \\
\textsc{bel} & : & \text{Set(Prop)} \\
\textsc{tmp} & : & \left[\begin{array}{lll} \textsc{usr} & : & \mathit{Tmp} \\ \textsc{sys} & : & \mathit{Tmp} \end{array}\right] \\
\textsc{nim} & : & \text{OpenQueue(Pair(DP,Move))}
\end{array}\right] \\
\textsc{shared} & : & \left[\begin{array}{lll}
\textsc{com} & : & \text{Set(Prop)} \\
\textsc{issues} & : & \text{OpenStack(Question)} \\
\textsc{qud} & : & \text{OpenStack(Question)} \\
\textsc{pm} & : & \text{OpenQueue(Move)} \\
\textsc{lu} & : & \left[\begin{array}{lll} \textsc{speaker} & : & \text{Participant} \\ \textsc{moves} & : & \text{Set(Move)} \end{array}\right]
\end{array}\right]
\end{array}\right]
$$

$$
\mathit{Tmp} = \left[\begin{array}{lll}
\textsc{com} & : & \text{Set(Prop)} \\
\textsc{issues} & : & \text{OpenStack(Question)} \\
\textsc{qud} & : & \text{OpenStack(Question)} \\
\textsc{agenda} & : & \text{OpenQueue(Action)} \\
\textsc{plan} & : & \text{OpenStack(PlanConstruct)}
\end{array}\right]
$$

Figure 4.1: IBiS3 Information State type

The first change compared to the IBiS2 information state is the addition of the open stack /SHARED/ISSUES, which contains the open issues. The /SHARED/QUD field has not been modified in terms of data type, but is now used for modelling the local QUD.

The second change is the division of /SHARED/TMP into two subfields. The SYS subfield corresponds to the TMP field in IBiS2, and contains parts of the information state copied right before integrating the latest system utterance. As in IBiS2, system utterances are optimistically assumed to be grounded, and if the user gives negative feedback the TMP/SYS field is used to retract the optimistic assumption. In addition, the system sometimes makes an optimistic assumption regarding the grounding and understanding of a user utterance, and produces positive feedback (e.g. "OK. To Paris."). In IBiS3, we will use a type of question accommodation to enable retraction of the optimistic grounding assumption regarding user utterances, in cases where the user rejects the system's reported interpretation. For this, we also need to keep a copy of relevant parts of the information state as they were right before the user's utterance was interpreted and integrated; this is what the TMP/USR field contains.

Finally, the items on /PRIVATE/NIM are now pairs, where the first element is the DP who made the move, and the second is the move itself. In IBiS2, it can be assumed that all non-integrated moves were performed in the latest utterance. In IBiS3, question accommodation mechanisms allow less restricted dialogues, and there is no longer any guarantee that all non-integrated moves were made in the latest utterance. Moves may be stored in NIM for several turns before being integrated.

4.6 Varieties of question accommodation and reaccommodation

As shown by the dialogue in (4.6)[4], questions can be answered (even elliptically) without previously having been raised.

(4.6) J : vicken månad ska du åka
what month do you want to go
B : ja: typ den: ä: tredje fjärde april / nån gång där
well around 3rd 4th april / some time there
P : så billit som möjlit
as cheap as possible

But where does the accommodated question come from? In principle, we could imagine a huge number of possible questions associated with any answer, especially if it is elliptical or semantically underspecified. How is this search space constrained? The answer lies in the activity which is being performed; the question must be available as part of the knowledge associated with the activity - either static knowledge describing how the activity is typically performed, or dynamic knowledge of the current state of the activity.

In this section we first describe the three basic question accommodation mechanisms: global question accommodation (issue accommodation), local question accommodation (QUD accommodation) and dependent issue accommodation. We then discuss the need for clarification questions in cases where it is not clear which question is being addressed, before moving on to describing reaccommodation and dependent reaccommodation. For each type of accommodation we also describe the implementation and provide dialogue examples from the implemented system.

In general, accommodation is tried only after "normal" integration has failed. The coordination of the accommodation rules in relation to grounding (including integration) rules is handled by the update algorithm described in Section 4.7.2.

[4]This dialogue has been collected by the University of Lund as part of the SDS project. We quote the transcription done in Göteborg as part of the same project.

4.6.1 Issue accommodation: from dialogue plan to ISSUES

This type of accommodation occurs when a DP addresses an issue which is not yet open but which is part of the current plan[5]. In the dialogue in example 6, P's second utterance ("as cheap as possible") addresses the issue of which price class P is interested in. At this stage of the dialogue, this issue has not been raised, but presumably J was planning to raise it eventually.

Before IBiS can integrate an answer, it needs to find an open issue to which the answer is relevant (see the definition of the **integrateUsrAnswer** rule in Section 3.6.6). Thus, to handle a dialogue like that in example 6 some mechanism is needed for finding an appropriate issue in the current dialogue plan and moving it to the ISSUES stack. A schematic representation of issue accommodation is shown in Figure 4.2.

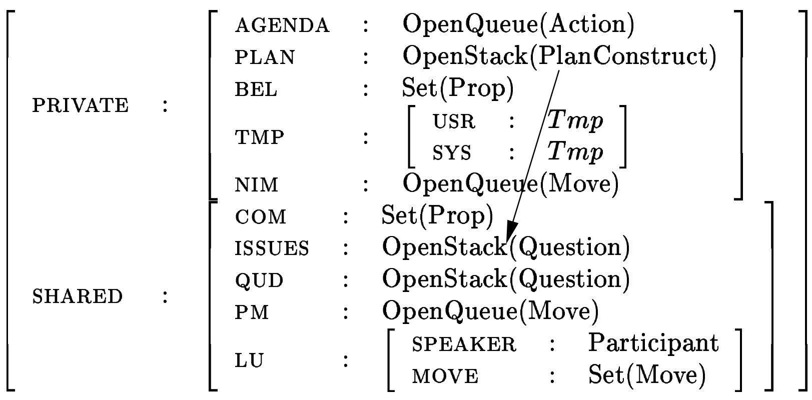


Figure 4.2: Issue accommodation

The issue accommodation update rule in (RULE 4.1) first checks whether a question which matches the answer occurs in the current dialogue plan (provided there is one). A question matches an answer if the answer is relevant to, or (in Ginzburg's terminology) about the question. If such a question can be found, it can be assumed that this is now an open issue. Accommodating this amounts to pushing the question on the ISSUES stack.

[5]Since the current plan is presumably being carried out in order to deal with some open issue, we may regard the utterance as indirectly relevant to some open issue (via the plan).

(RULE 4.1) RULE: **accommodatePlan2Issues**

$$
\begin{array}{ll}
\textsc{class:} & \mathsf{accommodate} \\
\textsc{pre:} & \left\{\begin{array}{l}
\$/\textsc{private/nim/elem/snd} = \mathsf{answer}(A) \\
\text{not } \$\textsc{lexicon} :: \text{yn\_answer}(A) \\
\text{in}(\$/\textsc{private/plan}, \mathsf{findout}(Q)) \\
\$\textsc{domain} :: \text{relevant}(A, Q) \\
\$\textsc{domain} :: \text{default\_question}(Q) \text{ or} \\
\quad \text{not (in}(\$/\textsc{private/plan}, \mathsf{findout}(Q')) \\
\qquad \text{and } Q \neq C \\
\qquad \text{and } \$\textsc{domain} :: \text{relevant}(A, Q') \text{)}
\end{array}\right. \\
\textsc{eff:} & \left\{ \text{push}(/\textsc{shared/issues}, B) \right.
\end{array}
$$

The first condition picks out a non-integrated answer move with content A. The second condition checks that A is not a *y/n* answer (e.g. **yes**, **no**, **maybe** etc.), and thus implements an assumption that such answers cannot trigger question accommodation, since they are too ambiguous[6]. The third and fourth conditions check if there is a findout action with content Q in the currently loaded plan, such that A is relevant to Q. The final condition checks that there is no other question in the plan that the answer is relevant to, or alternatively that Q has the status of a default question. If this condition does not hold, a clarification question should be raised by the system; this is described in Section 4.6.3. The "default question" option allows encoding of the fact that one issue may be significantly more salient in a certain domain. For example, in a travel agency setting the destination city may be regarded as more salient than the departure city question. If this is encoded as a default question, then if the user says simply "Paris" it is interpreted as answering the destination city question; no clarification is triggered[7]

Example dialogue: issue accommodation The dialogue in (DIALOGUE 4.1) illustrates accommodation of the question **?C.class(C)** from the plan to the stack of open issues.

(DIALOGUE 4.1)

[6]However, in general one cannot rule out the possibility that *y/n* answers can trigger accommodation in severely restricted domains. The assumption that this cannot happen can be regarded as a very simplified version of a constraint on the number of questions which an answer may be relevant without making question accommodation infeasible.

[7]The normal grounding mechanisms should of course enable correction of this assumption. In IBiS3 the choice of grounding strategy depends solely on the recognition score which means that a high-scoring answer may be interpreted as an answer to a default question and not receive any explicit feedback. This is one case which indicates a need for taking more factors into account when choosing feedback and grounding strategy.

S> `What month do you want to leave?`

$$\left[\begin{array}{lcl}
\text{PRIVATE} & = & \left[\begin{array}{lcl}
\text{AGENDA} & = & \langle\langle\rangle\rangle \\
\text{PLAN} & = & \left\langle\begin{array}{l}
\mathsf{findout}(?A.\mathsf{month}(A)) \\
\mathsf{findout}(?B.\mathsf{dept\_day}(B)) \\
\mathsf{findout}(?C.\mathsf{class}(C)) \\
\mathsf{consultDB}(?D.\mathsf{price}(D))
\end{array}\right\rangle \\
\text{BEL} & = & \{\} \\
\text{NIM} & = & \langle\langle\rangle\rangle
\end{array}\right] \\
\text{SHARED} & = & \left[\begin{array}{lcl}
\text{COM} & = & \left\{\begin{array}{l}
\mathsf{dest\_city}(\mathsf{paris}) \\
\mathsf{dept\_city}(\mathsf{london}) \\
\mathsf{how}(\mathsf{plane})
\end{array}\right\} \\
\text{ISSUES} & = & \left\langle\begin{array}{l}
?F.\mathsf{month}(F) \\
?E.\mathsf{price}(E)
\end{array}\right\rangle \\
\text{QUD} & = & \left\langle\ ?F.\mathsf{month}(F)\ \right\rangle \\
\text{PM} & = & \ldots \\
\text{LU} & = & \left[\begin{array}{lcl}
\text{SPEAKER} & = & \mathsf{sys} \\
\text{MOVES} & = & \mathsf{ask}(?F.\mathsf{month}(F)) \\
\text{SCORE} & = & 1
\end{array}\right]
\end{array}\right]
\end{array}\right]$$

U> `april as cheap as possible`

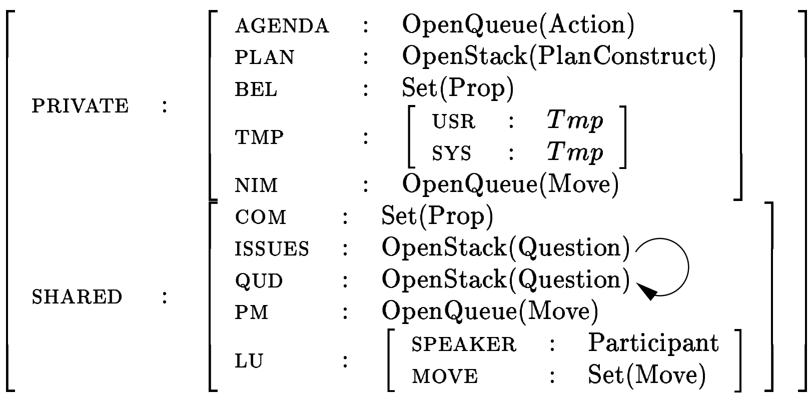
getLatestMoves
backupSharedUsr
integrateUsrShortAnswer
downdateISSUES
removeFindout
accommodatePlan2Issues
$\{$ push(/SHARED/ISSUES, $?A.\mathsf{class}(A)$)
integrateUsrFullAnswer
downdateISSUES
removeFindout
downdateQUD

$$\left[\begin{array}{lll}
\text{PRIVATE} & = & \left[\begin{array}{lll}
\text{AGENDA} & = & \langle\langle\ \mathsf{icm{:}acc{*}pos}\ \rangle\rangle \\
\text{PLAN} & = & \left\langle\begin{array}{l}\mathsf{findout}(?A.\mathsf{dept\_day}(A)) \\ \mathsf{consultDB}(?B.\mathsf{price}(B))\end{array}\right\rangle \\
\text{BEL} & = & \{\} \\
\text{NIM} & = & \langle\langle\rangle\rangle
\end{array}\right] \\
\text{SHARED} & = & \left[\begin{array}{lll}
\text{COM} & = & \left\{\begin{array}{l}\mathsf{class(economy)} \\ \mathsf{month(april)} \\ \mathsf{dest\_city(paris)} \\ \mathsf{dept\_city(london)} \\ \mathsf{how(plane)}\end{array}\right\} \\
\text{ISSUES} & = & \langle\ ?D.\mathsf{price}(D)\ \rangle \\
\text{QUD} & = & \langle\rangle \\
\text{PM} & = & \langle\langle\ \mathsf{icm{:}acc{*}pos,\ icm{:}loadplan,\ ask}(?C.\mathsf{month}(C))\ \rangle\rangle \\
\text{LU} & = & \left[\begin{array}{lll}
\text{SPEAKER} & = & \mathsf{usr} \\
\text{MOVES} & = & \langle\langle\ \mathsf{answer(april),\ answer(class(economy))}\ \rangle\rangle \\
\text{SCORE} & = & 1
\end{array}\right]
\end{array}\right]
\end{array}\right]$$

S> `Okay.  What day do you want to leave?`

4.6.2 Local question accommodation: from ISSUES to QUD

If a move with underspecified content is made which does not match any question on the QUD, the closest place to look for such a question is ISSUES, and if it can be found there it should be pushed on the local QUD to enable ellipsis resolution. As a side-effect, the question has now been brought into focus and should, if it is not topmost on the open issues stack, be raised to the top of open issues. A schematic overview of local question accommodation is shown in Figure 4.3.



$$\left[\begin{array}{lll}
\text{PRIVATE} & : & \left[\begin{array}{lll}
\text{AGENDA} & : & \text{OpenQueue(Action)} \\
\text{PLAN} & : & \text{OpenStack(PlanConstruct)} \\
\text{BEL} & : & \text{Set(Prop)} \\
\text{TMP} & : & \left[\begin{array}{lll}\text{USR} & : & \mathit{Tmp} \\ \text{SYS} & : & \mathit{Tmp}\end{array}\right] \\
\text{NIM} & : & \text{OpenQueue(Move)}
\end{array}\right] \\
\text{SHARED} & : & \left[\begin{array}{lll}
\text{COM} & : & \text{Set(Prop)} \\
\text{ISSUES} & : & \text{OpenStack(Question)} \\
\text{QUD} & : & \text{OpenStack(Question)} \\
\text{PM} & : & \text{OpenQueue(Move)} \\
\text{LU} & : & \left[\begin{array}{lll}\text{SPEAKER} & : & \text{Participant} \\ \text{MOVE} & : & \text{Set(Move)}\end{array}\right]
\end{array}\right]
\end{array}\right]$$

Figure 4.3: Local question accommodation

This type of accommodation can e.g. occur if a question which was raised previously has dropped off the local QUD but has not yet been resolved and remains on ISSUES. It should also be noted that several accommodation steps can be taken during the processing of a single utterance; for example, if an issue that is in the plan but has not yet been raised is answered elliptically.

(RULE 4.2) RULE: **accommodateIssues2QUD**
CLASS: accommodate

$$\text{PRE:} \left\{ \begin{array}{l} \$/\textsc{private}/\textsc{nim}/\textsc{elem}{=}\mathsf{usr\text{-}answer}(A) \\ \$\textsc{domain} :: \text{short\_answer}(A) \\ \text{not } \$\textsc{lexicon} :: \text{yn\_answer}(A) \\ \text{in}(\$/\textsc{shared}/\textsc{issues}, Q) \\ \text{not in}(\$/\textsc{shared}/\textsc{qud}, Q) \\ \$\textsc{domain} :: \text{relevant}(A, Q) \end{array} \right.$$

$$\text{EFF:} \left\{ \begin{array}{l} \text{push}(/\textsc{shared}/\textsc{qud}, Q) \\ \text{raise}(/\textsc{shared}/\textsc{issues}, Q) \end{array} \right.$$

The second condition in (RULE 4.2) checks that the content of the answer move picked out by condition 1 is semantically underspecified. The third condition imposes a constraint on local question accommodation, excluding short answers to *y/n*-questions (“yes”, “no”, “maybe” etc.). The remaining conditions check that the answer-content is relevant to an issue which is on ISSUES but not on QUD. The first operation pushes the accommodated question on QUD, and the final update raises the question to the top of the stack of open issues.

4.6.3 Issue clarification

In IBiS2, user answers are either pragmatically relevant to the question topmost on QUD, or not relevant at all. When we add mechanisms of accommodation to allow for answers to unraised questions, it becomes necessary to deal with cases where an answer may be potentially relevant to several different questions.

Semantically underspecified answers may (but need not) be pragmatically ambiguous, i.e. it is not clear what question they provide an answer to. This can be resolved by asking the speaker what question she intended to answer (or equivalently, which proposition she wanted to convey).

In this case, we can use the same strategy as for negative grounding, i.e. when a pragmatically ambiguous utterance is to be interpreted the system raises a question whose answer will be integrated instead of the ambiguous answer. For example, “Paris” may be relevant to either the destination city question or the departure city question. When the the clarification question “Do you mean from Paris or to Paris?” is raised it is expected that the user will answer this question, which means that the ambiguous answer no longer needs to be integrated and can be thrown away[8].

In this way we see how question accommodation, amended with a mechanism for resolving which question to accommodate, can be used to resolve pragmatic ambiguities in user input. The accommodation mechanism can thus be regarded as a refinement of the account of grounding on the understanding level put forward in Chapter 3. The rule which selects the issue clarification issue is shown in (RULE 4.3).

(RULE 4.3) RULE: **clarifyIssue**
CLASS: select_action

$$\text{PRE: } \begin{cases} \text{in(\$/PRIVATE/NIM, usr-answer}(A)) \\ \text{setof}(C, \text{in(\$/PRIVATE/PLAN, findout}(Q)) \text{ and} \\ \quad \text{\$DOMAIN :: relevant}(A, Q), QSet) \\ \text{\$\$arity}(QSet) > 1 \end{cases}$$

$$\text{EFF: } \begin{cases} \text{! setof}(?P, \text{in}(QSet, Q) \text{ and \$DOMAIN :: combine}(Q, A, P), AltQ) \\ \text{push(/PRIVATE/AGENDA, findout}(AltQ)) \\ \text{del(/PRIVATE/NIM, usr-answer}(A)) \end{cases}$$

The first condition picks out the answer-move from the NIM queue. The second and third conditions check that there is more than one question in the plan to which the answer is relevant, by constructing the set of such questions. The first operation constructs the alternative-question by applying each question in the set constructed in condition 2 to the answer to get a proposition and prefixing the question operator '?' to each proposition to get a *y/n*-question. The alternative question is this set of *y/n*-questions. The second operation pushes the action to raise the alternative question on the agenda, and the final operation removes the answer move from NIM; this is motivated above.

A sample dialogue with issue clarification is shown in (DIALOGUE 4.2).

(DIALOGUE 4.2)

```
S> Welcome to the travel agency!
U> price information please
S> Okay.  I need some information.  How do you want to travel?
U> flight um paris
S> OK, by flight. Do you mean from paris or to paris?
```

[8]IBiS3 only handles full answers to clarification questions, i.e. “To Paris.” or “From Paris.”. A slightly more advanced semantics would be required to handle cases where the user again gives an underspecified response which resolves the question, i.e. “To.” or “From.”.

The user's utterance of "paris" is interpreted as answer(**paris**), which is relevant to two questions in the plan: $?x.\mathbf{dest\_city}(x)$ and $?x.\mathbf{dept\_city}(x)$. Because of this, the issue accommodation rule in 1 will not fire and the answer is not integrated. This allows the **clarifyIssue** rule to fire in the selection phase. By combining each of these questions with the content of the answer (**paris**), and turning each resulting proposition into a *y/n*-question, the set $\{\mathbf{?dest\_city(paris),?dest\_city(paris)}\}$ is obtained. This set also works as an alternative-question, which is used as the content of the clarification question in the system's final utterance in (4.6).

Note that these clarification questions are dynamically put together by the system and thus do not need to be pre-programmed. This means that the application designer does not even need to realize that an ambiguity exists.

4.6.4 Dependent issue accommodation: from domain resource to ISSUES

Issue accommodation, introduced above, presupposes that there is a current plan in which to look for an appropriate question; this, in turn, presupposes that there is some issue under discussion which the plan is meant to deal with. But what if there is currently no plan?

In this case, it may be necessary to look in at the set of stored domain-specific dialogue plans (or come up with a new plan) to try to figure out which issue the latest utterance was addressing. An appropriate plan should contain a question matching some information provided in the latest utterance. If such a plan is found, it is possible that, in addition to the question answered by the latest utterance, a further issue should also be accommodated: the "goal-issue" which the plan in question is aimed at dealing with. Given our definition of dependence between questions in Section 2.8.2, the goal issue is dependent on the issue directly addressed, and hence we refer to this as dependent issue accommodation.

Dependent issue accommodation is thus the process of finding an appropriate background issue and a plan for dealing with that issue which makes the latest utterance relevant, given "normal" global issue accommodation. That is, dependent issue accommodation is always followed by global issue accommodation. Dependent issue accommodation applies when no issues are under discussion, and a previously unraised question is answered using a full or short answer (in the latter case, global issue accommodation must in turn be followed by local question accommodation). A schematic overview of dependent issue accommodation is shown in Figure 4.4, and the update rule is shown in (4.7).

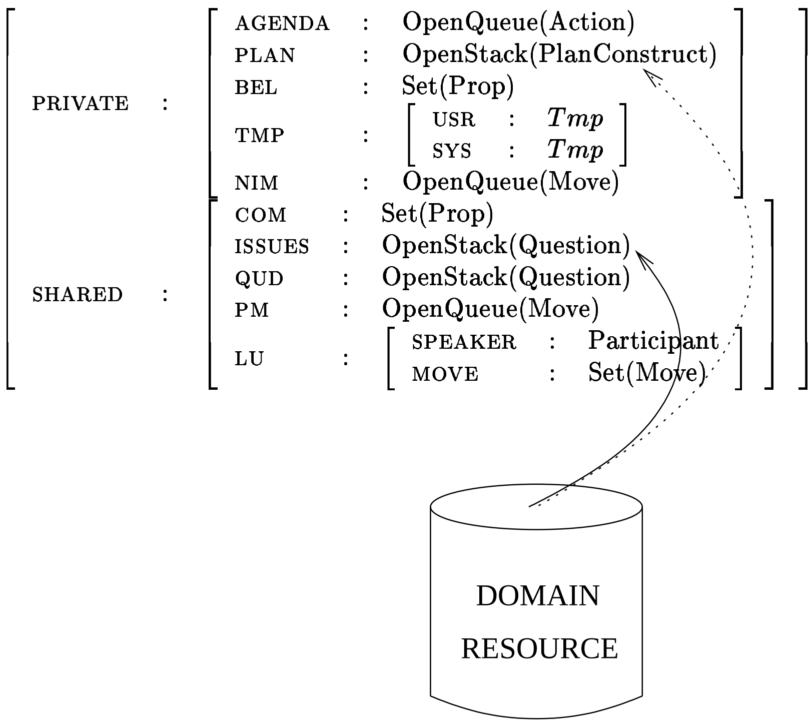


Figure 4.4: Dependent issue accommodation

```
(RULE 4.4)  RULE: accommodateDependentIssue
            CLASS: accommodate
                   setof(A, $/PRIVATE/NIM/ELEM/SND=answer(A), AnsSet)
                   $$arity(AnsSet) > 0
                   is_empty($/PRIVATE/PLAN)
                   $DOMAIN :: plan(DepQ, Plan)
                   forall(in(AnsSet, A), in(Plan, findout(Q)) and
            PRE:  {    $DOMAIN :: relevant(A, Q))
                   not ( $DOMAIN :: plan(DepQ′, Plan′) and DepQ′ ≠ DepQ and
                       forall(in(AnswerSet, A), in(Plan′, findout(Q)) and
                           $DOMAIN :: relevant(A, Q)) )
                   not in($/PRIVATE/AGENDA, icm:und*int:usr*issue(DepQ))
                   push(/SHARED/ISSUES, DepQ)
                   push(/PRIVATE/AGENDA, icm:accommodate:DepQ)
            EFF:  {push(/PRIVATE/AGENDA, icm:und*pos:usr*issue(DepQ))
                   set(/PRIVATE/PLAN, Plan)
                   push(/PRIVATE/AGENDA, icm:loadplan)
```

The first two conditions construct a set of all non-integrated answers and check that the arity of this set is larger than zero, i.e. that there is at least one non-integrated answer.

It should be noted that this formulation of the rule relies on the assumption that all unintegrated answers have been provided by the user. This is true for IBiS3, since all system answers are integrated immediately and never need accommodation. However, in a more complex system this may not always be true; in this case, the rule would need some slight modifications to only pick out user moves.

The third condition checks that the plan is empty. The consequence of this is that dependent issue accommodation is not available when some plan is being executed, so if an issue is being dealt with the only way to raise a new issue is to do so explicitly. We believe this is a reasonable restriction, but if desired it can be disabled by removing the condition. However, doing so may give problems in case speech recognition mistakenly recognizes an answer not matching the current plan; if this answer triggers dependent accommodation this may result in confusing utterances from the system.

The fourth and fifth conditions look for a plan in the domain resource to which all non-integrated answers are relevant. This can be regarded as a simple version of plan recognition: given an observed set of actions (user answers), try to find a plan and a goal (an issue) such that the actions fit the plan. Here, the user answers fit the plan by being relevant answers to questions in the plan (more precisely, questions such that the plan includes actions to resolve them).

The final condition checks that there is only one plan to which all the answers are relevant. If there are several such plans, the accommodation rule should not trigger; instead, a clarification question should be raised by the system (see Section 4.6.5).

The updates push the dependent issue on ISSUES, loads the plan, and pushes the appropriate ICM moves on the agenda: positive feedback concerning the accommodated issue ("You want to know about price.") and feedback indicating that a new plan has been loaded ("I need some information"). In addition, ICM indicating accommodation is produced (see Section 4.7.1).

(DIALOGUE 4.3)

S> `Welcome to the travel agency!`

U> `i want a flight`

getLatestMoves
backupSharedUsr
accommodateDependentIssue

$$\left\{\begin{array}{l}
\text{push}(/\textsc{shared}/\textsc{issues},\ ?C.\mathsf{price}(C)) \\
\text{push}(/\textsc{private}/\textsc{agenda},\ \mathsf{icm{:}accommodate{:}}?C.\mathsf{price}(C)) \\
\text{push}(/\textsc{private}/\textsc{agenda},\ \mathsf{icm{:}und{*}pos{:}usr{*}issue}(?C.\mathsf{price}(C))) \\
\text{set}(/\textsc{private}/\textsc{plan},\ \mathsf{stackset}([\mathsf{findout}(?D.\mathsf{how}(D)),\ \mathsf{findout}(?E.\mathsf{dest\_city}(E)),\ \ldots\ ])) \\
\text{push}(/\textsc{private}/\textsc{agenda},\ \mathsf{icm{:}loadplan})
\end{array}\right.$$

accommodatePlan2Issues

$$\left\{\ \text{push}(/\textsc{shared}/\textsc{issues},\ ?A.\mathsf{how}(A))\right.$$

accommodateIssues2QUD

$$\left\{\begin{array}{l}
\text{push}(/\textsc{shared}/\textsc{qud},\ ?A.\mathsf{how}(A)) \\
\text{raise}(/\textsc{shared}/\textsc{issues},\ ?A.\mathsf{how}(A))
\end{array}\right.$$

integrateUsrAnswer
downdateISSUES
removeFindout
downdateQUD

backupSharedSys
selectIcmOther
selectIcmOther

$$\left[\begin{array}{lll}
\textsc{private} & = & \left[\begin{array}{lll}
\textsc{agenda} & = & \langle\langle\ \mathsf{icm{:}loadplan},\ \mathsf{icm{:}acc{*}pos}\ \rangle\rangle \\
\textsc{plan} & = & \left\langle\begin{array}{l}
\mathsf{findout}(?A.\mathsf{dest\_city}(A)) \\
\mathsf{findout}(?B.\mathsf{dept\_city}(B)) \\
\mathsf{findout}(?C.\mathsf{month}(C)) \\
\mathsf{findout}(?D.\mathsf{dept\_day}(D)) \\
\mathsf{findout}(?E.\mathsf{class}(E)) \\
\mathsf{consultDB}(?F.\mathsf{price}(F))
\end{array}\right\rangle \\
\textsc{bel} & = & \ldots \\
\textsc{nim} & = & \ldots
\end{array}\right] \\
\textsc{shared} & = & \left[\begin{array}{lll}
\textsc{com} & = & \{\ \mathsf{how(plane)}\ \} \\
\textsc{issues} & = & \langle\ ?G.\mathsf{price}(G)\ \rangle \\
\textsc{qud} & = & \langle\rangle \\
\textsc{pm} & = & \langle\langle\ \mathsf{greet}\ \rangle\rangle \\
\textsc{lu} & = & \left[\begin{array}{lll}
\textsc{speaker} & = & \mathsf{usr} \\
\textsc{moves} & = & \langle\langle\ \mathsf{answer(plane)}\ \rangle\rangle \\
\textsc{score} & = & 1
\end{array}\right]
\end{array}\right]
\end{array}\right]$$

S> `Alright.  You want to know about price.`

U>

S> `I need some information.  Okay.  By flight.  What city do you want to go to?`

The current solution has an optimistic strategy for dependent accommodation: the issue is assumed to be under discussion and the system gives explicit positive feedback of this assumption. It may be argued that a pessimistic strategy is more appropriate for dependent accommodation; this can be achieved by replacing the list of updates in 4 with the update in (4.7).

(4.7) push(/PRIVATE/AGENDA, icm:und*int:usr*issue(D))

This will provide interrogative feedback from the system concerning whether the dependent issue should be opened, e.g. "You want to know about price, is that correct?". If the user gives a positive response to this feedback, the system will use the same update rules as usual for integrating the user's response to interrogative feedback.

(DIALOGUE 4.4)

S> `Welcome to the travel agency!`

U> `i want a flight`

getLatestMoves
backupSharedUsr
accommodateDependentIssue
{ push(/PRIVATE/AGENDA, icm:und*int:usr*issue(?C.price(C)))
downdateQUD
backupSharedSys
selectIcmUndNeg
selectIcmOther

S> `flight. I dont quite understand. You want to know about price , is that correct?`

getLatestMoves
integrateOtherICM
integrateOtherICM
integrateUndIntICM

U> `yes`

getLatestMoves
integratePosIcmAnswer
findPlan
accommodatePlan2Issues
accommodateIssues2QUD
integrateUsrAnswer
downdateQUD

$$\begin{bmatrix} \textsc{private} & = & \begin{bmatrix} \textsc{agenda} & = & \langle\langle\ \text{icm:loadplan, icm:und*int:usr*how(plane)}\ \rangle\rangle \\ \textsc{plan} & = & \left\langle \begin{array}{l} \text{findout}(?A.\text{how}(A)) \\ \text{findout}(?B.\text{dest\_city}(B)) \\ \text{findout}(?C.\text{dept\_city}(C)) \\ \text{findout}(?D.\text{month}(D)) \\ \text{findout}(?E.\text{dept\_day}(E)) \\ \text{findout}(?F.\text{class}(F)) \\ \text{consultDB}(?G.\text{price}(G)) \end{array} \right\rangle \\ \textsc{bel} & = & \{\} \\ \textsc{nim} & = & \langle\langle\langle\rangle\rangle\rangle \end{bmatrix} \\ \textsc{shared} & = & \begin{bmatrix} \textsc{com} & = & \{\} \\ \textsc{issues} & = & \left\langle \begin{array}{l} ?A.\text{how}(A) \\ ?H.\text{price}(H) \end{array} \right\rangle \\ \textsc{qud} & = & \langle\rangle \\ \textsc{pm} & = & \langle\langle\ \text{icm:sem*pos:answer(plane), \ldots}\ \rangle\rangle \\ \textsc{lu} & = & \begin{bmatrix} \textsc{speaker} & = & \text{usr} \\ \textsc{moves} & = & \text{oqueueanswer(yes)} \\ \textsc{score} & = & 1 \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

backupSharedSys
selectIcmOther
selectIcmOther

```
S> I need some information.  by flight , is that correct?
```

4.6.5 Dependent issue clarification

If no plan is loaded and one or several non-integrated answers are relevant to several plans, a clarification question should be raised by the system to find out which issue the user wants the system to deal with. This is done by the selection rule in (RULE 4.5).

(RULE 4.5) RULE: **clarifyDependentIssue**
CLASS: select_action

$$\text{PRE:} \left\{ \begin{array}{l} \text{in}(\$/\textsc{private}/\textsc{nim},\ \text{pair(usr, answer}(A))) \\ \text{setof}(Q',\ \$\textsc{domain} :: \text{plan}(Q',\ Plan) \text{ and} \\ \qquad \text{in}(Plan,\ \text{findout}(SomeQ)) \text{ and} \\ \qquad \$\textsc{domain} :: \text{relevant}(A,\ SomeQ), \\ QSet') \\ \text{remove\_unifiables}(QSet',\ QSet) \\ \$\$\text{arity}(QSet) > 1 \end{array} \right.$$

$$\text{EFF:} \left\{ \begin{array}{l} !\ \text{setof}(IssueQ,\ \text{in}(QSet,\ I) \text{ and } IssueQ{=}?\text{issue}(I),\ AltQ) \\ \text{push}(/\textsc{private}/\textsc{agenda},\ \text{findout}(AltQ)) \end{array} \right.$$

The first condition checks if there is at least one non-integrated user answer left after the system has attempted to integrate the latest user utterance. The second and third conditions constructs the set *QSet* of dependent issues that the non-integrated answer is indirectly relevant to (i.e. issues for which there is a plan containing an action to resolve a question to which the answer is relevant)[9]. The final condition checks that there is more than one such dependent issue.

The first update constructs an alternative-question by picking out each question I in *QSet* and adding **?issue**(I) to the set which constitutes the alternative-question. The final update pushes an action to resolve alternative-question on the agenda.

In the travel agency domain, an example of dependent issue clarification occurs if the user's first utterance is "to Paris", interpreted as answer(**dest_city(paris)**). This answer is relevant to the question **?**x.**dest_city**(x) which occurs in both the plan for addressing the price issue and that for addressing the visa issue. This blocks the dependent issue accommodation rule. In the dialogue in (DIALOGUE 4.5), the system instead raises a clarification question. Note that IBiS3 here makes use of the fact that an ask-move can supply an answer to a question concerning which issue to pursue.

(DIALOGUE 4.5)

S> `Welcome to the travel agency!`

U> `to paris`

getLatestMoves
backupSharedUsr
downdateQUD

backupSharedSys
clarifyDependentIssue
$\left\{ \begin{array}{l} !\ \text{setof}(E,\ \text{in}(\text{set}([\text{need\_visa},\ ?D.\text{price}(D)]),\ F)\ \text{and}\ E{=}\text{issue}(F),\ G) \\ \text{push}(/\text{PRIVATE}/\text{AGENDA},\ \text{findout}(G)) \end{array} \right.$
selectIcmUndNeg
selectAsk

S> `to paris.  I dont quite understand.  Do you mean to ask about visa or to ask about price?`

[9]The remove_unifiables condition is used to remove multiple occurrences of the same issue. Note that these occurrences are not identical, since they may differ in the identity of variables. One may of course argue whether sets should have this property, but in the current TrindiKit implementation they do.

```
getLatestMoves
integrateOtherICM
integrateOtherICM
integrateSysAsk


U> visa

getLatestMoves
backupSharedUsr
integrateUsrAsk
downdateISSUES
findPlan
accommodatePlan2Issues
integrateUsrAnswer
downdateQUD

S> Okay.  I need some information.  to paris , is that correct?
```

4.6.6 Question reaccommodation

In IBiS1 and IBiS2 the user has a limited ability to reraise previously resolved issues; this will typically result in the system giving the same answer again. However, this kind of reraising is not very useful since the user is not able to modify her own answers to the system's previous questions.

Global question reaccommodation (Issue reaccommodation)

In general, if the user provides an alternative resolution of an issue which has been previously resolved, this triggers a reraising of that issue. If the previous answer is incompatible with the new one, the old answer is removed. This allows the user to change his/her mind during the dialogue. Here is an example dialogue with the system:

(DIALOGUE 4.6)

```
S> Welcome to the travel agency!
U> price information please
S> You asked about price.  How do you want to travel?
```

```
U> a flight, april the fifth
S> by flight.  in april.  the fifth.  Okay.  What city do you want to go
to?
U> london
S> Okay.  to london.
U> actually, i want to go on the fourth
S> the fourth.  What city do you want to go from?
```

Initially, integration of the answer using **integrateUsrAnswer** (Section 3.6.6) will fail since there is no matching question on ISSUES. The system will then try various accommodation strategies, including accommodation from /SHARED/COM formulated in (RULE 4.6).

(RULE 4.6) RULE: **accommodateCom2Issues**
CLASS: accommodate

$$\text{PRE:} \left\{ \begin{array}{l} \$/\text{PRIVATE}/\text{NIM}/\text{ELEM}/\text{SND}{=}\mathsf{answer}(A) \\ \text{in}(\$/\text{SHARED}/\text{COM},\ P) \\ \$\text{DOMAIN} :: \text{question}(Q) \\ \$\text{DOMAIN} :: \text{relevant}(A,\ Q) \\ \$\text{DOMAIN} :: \text{relevant}(P,\ Q) \end{array} \right.$$

$$\text{EFF:} \left\{ \text{push}(/\text{SHARED}/\text{ISSUES},\ Q) \right.$$

This accommodation rule looks for an answer A among the moves which have not yet been integrated (first condition). It then looks for a proposition among the shared commitments established in the dialogue so far (second condition) which according to the system's domain resource is an appropriate answer to some question for which A is also an answer (third to fifth conditions). Given that in this simple system answers can only be relevant to a single question[10], this strategy will be successful in identifying cases where we have two answers to the same question. A system that deals with more complex dialogues where this is not the case would need to keep track of closed issues in a separate list of closed issues. Thus the conditions will succeed if there is a question such that both the user answer and a stored proposition are relevant answers to it; in the example dialogue above, "departure date is the fourth" and "departure date is the fifth" are both relevant answers to the question "which day do you want to travel?". If such a question is found it is accommodated to ISSUES, that is, it becomes an open issue again.

When **accommodateCom2Issues** has been successfully applied, the retract rule in (RULE

[10]That is, in the full form in which they appear in $/SHARED/COM. "Chicago" can be an answer to "Which city do you want to go to?" and "Which city do you want to go from?" but when it has been combined with the questions the result will be "destination(Chicago)" and "from(Chicago)" respectively and it is this which is entered into the commitments.

4.7) will remove the incompatible information from the system's view of shared commitments represented in /SHARED/COM.

(RULE 4.7) RULE: **retract**
CLASS: integrate

$$\text{PRE:}\begin{cases}\$/\text{PRIVATE/NIM/ELEM/SND}=\mathsf{answer}(A)\\ \text{in}(\$/\text{SHARED/COM},\ P')\\ \text{fst}(\$/\text{SHARED/ISSUES},\ Q)\\ \$\text{DOMAIN} :: \text{relevant}(P,\ Q)\\ \$\text{DOMAIN} :: \text{relevant}(A,\ Q)\\ \$\text{DOMAIN} :: \text{combine}(Q,\ A,\ P)\\ \$\text{DOMAIN} :: \text{incompatible}(P,\ P')\end{cases}$$

$$\text{EFF:}\begin{cases}\text{del}(/\text{SHARED/COM},\ P')\end{cases}$$

The conditions here are similar to those in (RULE 4.6). We look for an unintegrated answer (first condition) which is relevant to a question at the head of the list of open issues (third and fifth conditions) and for which there is already a relevant answer in the shared commitments (second and fourth conditions). Finally, we determine that the result of combining the answer with the question (sixth condition) is incompatible with the answer already found (seventh condition). If all this is true we delete the answer which is currently in the shared commitments. This will finally allow the new answer to be integrated by a rule that integrates an answer from the user, and a further rule will remove the resolved issue from QUD. Note that this rule is of class integrate. As is indicated in Appendix B, it is tried before any other integration rule, to avoid integration of conflicting information.

Note also that the "incompatible" relation is defined as a part of the domain resource, and can thus be domain specific. The simple kind of revision that IBiS currently deals with is also handled by some form-based systems (although they usually do not give feedback indicating that information has been removed or replaced, as IBiS does). For example, Chu-Carroll (2000) achieves a similar result by extracting parameter values from the latest user utterance and subsequently (if possible) copying values from the previous form for any parameters not specified in the latest utterance. A similar mechanism is referred to as "overlay" by Alexandersson and Becker (2000). While we are dealing only with very simple revision here, the rule in (RULE 4.7) and the "incompatible" relation can be seen as placeholders for a more sophisticated mechanism of belief revision.

It is also possible to remove the old answer by denying it (asserting its negation) as in (DIALOGUE 4.7).

(DIALOGUE 4.7)

```
S> Welcome to the travel agency!
U> price information for a flight to paris on april the fifth
S> You asked about price.  by flight.  to paris.  in april.  the fifth.
What city do you want to go from?
U> actually, not the fifth
S> not the fifth.  So, what day do you want to leave?
```

In this case, the system will explicitly reraise the issue to get a new response from the user. Again, the system will use the rule in 25 and signal reraising using ”so, ”. All the rules will be applied as in the previous case, but the departure date question will not be removed since it is not resolved by the given answer. Eventually, this leads to the system reraising the question.

Reraising of dependent questions (dependent issue reaccommodation)

In some cases, an issue might be reraised which influences the answer to a further issue that has also been resolved. For example, the choice of price class for a flight influences the price of the flight. In this case, the influenced question also needs to be reaccommodated and answered again.

(DIALOGUE 4.8)

```
S> Welcome to the travel agency!
U> what's the price of a flight from london to paris april the fifth?
S> You asked about price.  by flight.  from london.  to paris.  in april.
the fifth.  What class did you have in mind?
U> as cheap as possible
S> cheap.  Okay.  The price is 123 crowns.
U> actually, i might go for business class
S> first class.  Okay.  Concerning your question about price :  The price
is 1234 crowns.
```

The rule that achieves the reraising of a dependent question COM-to-ISSUES accommodation is shown in (RULE 4.8).

$$\text{(RULE 4.8)}\quad \begin{array}{l} \textsc{rule:}\ \textbf{accommodateCom2IssuesDependent} \\ \textsc{class:}\ \mathsf{accommodate} \\ \textsc{pre:} \left\{ \begin{array}{l} \$/\textsc{private}/\textsc{nim}/\textsc{elem}/\textsc{snd}{=}\mathsf{answer}(A) \\ \text{in}(\$/\textsc{shared}/\textsc{com},\ P) \\ \$\textsc{domain} :: \text{question}(Q) \\ \$\textsc{domain} :: \text{relevant}(A,\ Q) \\ \$\textsc{domain} :: \text{relevant}(P,\ Q) \\ \text{is\_empty}(\$/\textsc{shared}/\textsc{issues}) \\ \$\textsc{domain} :: \text{depends}(Q',\ Q) \\ \text{in}(\$/\textsc{shared}/\textsc{com},\ P') \\ \$\textsc{domain} :: \text{relevant}(P',\ Q') \end{array} \right. \\ \textsc{eff:} \left\{ \begin{array}{l} \text{del}(/\textsc{private}/\textsc{bel},\ P') \\ \text{del}(/\textsc{shared}/\textsc{com},\ P') \\ \text{push}(/\textsc{shared}/\textsc{issues},\ Q') \\ \text{push}(/\textsc{shared}/\textsc{issues},\ Q) \\ \text{push}(/\textsc{private}/\textsc{agenda},\ \mathsf{respond}(Q')) \end{array} \right. \end{array}$$

This rule is similar to 6 except that is looks for a question which depends on the question it finds corresponding to the answer provided by the user. It puts both question onto the list of open issues and plans to respond to the dependent question. This rule, as currently implemented, is specific to the particular case treated in the system. There is, of course, a great deal more to say about what it means for one question to be dependent on another and how the system knows whether it should respond to dependent questions or raise them with the user.

4.6.7 Opening up implicit grounding issues

In Chapter 3 we outlined a general issue-based account of grounding, where issues of contact, perception, understanding and acceptance of utterances may be raised and addressed. Parts of this account were implemented in IBiS2, allowing the system e.g. to raise understanding questions regarding the user's input (e.g. "To Paris, is that correct?"). This is a case of explicitly raising the understanding-question which results in this question being under discussion.

The system could also produce positive explicit feedback (e.g. "To Paris"); this kind of feedback does not explicitly raise the understanding question, and there is no obligation on the user to respond to it before the dialogue can proceed. However, it can be argued that even positive feedback raises grounding-related issues, although not explicitly. This is given some support from the fact that it is possible for the user to protest against the system's feedback in case the system got something wrong.

According to Ginzburg, an assertion can be followed by any utterance addressing the acceptance of this question as a fact, e.g. by saying “no!”. This is then regarded as a short answer to the acceptance question; in effect, a rejection. In the case of an assertion addressing understanding (i.e. positive understanding feedback), the acceptance question can be paraphrased “Is it correct that you meant ’to Paris’?”. That is, the acceptance-question regarding the system’s understanding is exactly the same question which is raised explicitly by an interrogative feedback utterance.

In IBiS, we have chosen not to represent acceptance-questions explicitly; however, in the case of positive explicit grounding there are good reasons to do so. Positive feedback has the advantage of increased efficiency compared to interrogative feedback, but the disadvantage is that the user is not able to correct the system’s interpretation. However, if the positive feedback move implicitly raises the question whether the system’s interpretation was correct, we can use this to allow the user to reject faulty system interpretations. Besides, we already have mechanisms in place for representing and dealing with answers to the understanding-question.

To model the fact that the acceptance question regarding understanding is implicit rather than explicit, we push it onto the local QUD only. If the user addresses it (e.g. by saying “no”), the implicit issue is “opened up”, i.e. it becomes an open issue; it is pushed on ISSUES.

(RULE 4.9) RULE: **accommodateQUD2Issues**
CLASS: accommodate

$$\text{PRE:} \left\{ \begin{array}{l} \$/\text{PRIVATE/NIM/ELEM/SND}{=}\mathsf{answer}(A) \\ \text{in}(\$/\text{SHARED/QUD},\ Q) \\ \$\text{DOMAIN} :: \text{relevant}(A,\ Q) \\ \text{not in}(\$/\text{SHARED/ISSUES},\ Q) \end{array} \right.$$

$$\text{EFF:} \left\{ \ \text{push}(/\text{SHARED/ISSUES},\ Q) \right.$$

The rule in (RULE 4.9) picks out a non-integrated answer-move which is relevant to a question on QUD which is not currently an open issue, and pushes it on ISSUES.

To handle integration responses to positive understanding feedback, we also need to modify the **integrateNegIcmAnswer** rule described in Section 3.6.6. A significant difference between positive and interrogative feedback in IBiS is that the former is associated with cautiously optimistic grounding, while the latter is used in the pessimistic grounding strategy. This means that a negative response to feedback on the understanding level must be handled differently depending on whether the content in question has been added to the dialogue gameboard or not. Specifically, if the positive feedback is rejected the optimistic grounding assumption must be retracted.

```
(RULE 4.10) RULE: integrateNegIcmAnswer
            CLASS: integrate
            PRE: { $/PRIVATE/NIM/FST/SND=answer($A$)
                   fst($/SHARED/ISSUES, $Q$)
                   $DOMAIN :: resolves($A$, $Q$)
                   fst($/SHARED/QUD, $Q$)
                   $DOMAIN :: combine($Q$, $A$, $P$)
                   $P$=not(und($DP$*$C$))
            EFF: { pop(/PRIVATE/NIM)
                   pop(/SHARED/ISSUES)
                   if_do(in($/SHARED/COM, $C$) or
                       $C$=issue($Q'$) and in($/SHARED/ISSUES, $Q'$), [
                       /SHARED/QUD := $/PRIVATE/TMP/$DP$/QUD
                       /SHARED/ISSUES := $/PRIVATE/TMP/$DP$/ISSUES
                       /SHARED/COM := $/PRIVATE/TMP/$DP$/COM
                       /PRIVATE/AGENDA := $/PRIVATE/TMP/$DP$/AGENDA
                       /PRIVATE/PLAN := $/PRIVATE/TMP/$DP$/PLAN ])
                   push(/PRIVATE/AGENDA, icm:und*pos:$DP$*not($C$))
                   clear(/PRIVATE/NIM)
                   init_shift(/PRIVATE/NIM)
```

The rule in (RULE 4.10) is similar to those for integrating "normal" user answers (see Section 3.6.6), because of the special nature of grounding issues, we include issue downdating in the rule rather than adding a further rule for downdating ISSUES for this special case. This means the rule has to check that the answer resolves the grounding issue, rather than merely checking that it is relevant; this is done in the third condition. The content resulting from combining the issue on QUD and the answer is computed in the fifth condition. Finally, the sixth condition checks that the content is not(und(DP*C)) where DP is a DP and C is the content that is being grounded (or in this case, not grounded).

The second update removes the grounding question from ISSUES. The third update first checks if C has been optimistically grounded. In this case, the optimistic grounding assumption regarding the grounding of C is retracted. This is where the new TMP/USR field, containing relevant parts of the information state as they were before the latest user utterance was optimistically assumed to be grounded, is used. If C has not been optimistically assumed to be grounded, nothing in particular needs to be done.

The fourth update adds positive feedback that the system has understood that C was false. Note that **not**(C) is not added to /SHARED/COM. The reason for this is that the negated proposition is not something that the user intended to add to the DGB - it was simply a result of a misunderstanding by the system.

Note also that this feedback will not raise a grounding issue according to the definition of question-raising ICM in Section 3.7.1. Since the content **not**(C) has not been added to the information state, there is no point in dealing with grounding.

The final two updates clear the NIM queue, which means that the system will disregard any moves which have not yet been integrated. One motivation for this is that if the system has misheard some part of the user's utterance, it is likely that it also misheard the rest. Clearing the NIM stack is also useful in dialogues such as that in (DIALOGUE 4.9).

(DIALOGUE 4.9)

S> `Welcome to the travel agency!`
U> `flight to paris`
$S> `flight. to paris. I dont quite understand. You want to know about price, is that correct?`
U> `no`
S> `You did not ask about price.`

When integrating the user's "no" (a negative response to system ICM), the answer-moves realized in the utterance "flight to paris" are discarded;. The system has tried to make sense of it but the user rejected this attempt. At this point, the system simply cannot deal with them and rather than getting stuck in trying to figure out what the user meant, the moves are thrown out.

Note that the rule as implemented is actually more general than what is needed for (or used in) IBiS3. Since the part of TMP that it backtracks to depends on the DP variable, in principle it could be used for cases where the user gives positive feedback and the system rejects this as mistaken.

A sample dialogue with a negative response to an implicit grounding question is shown in (DIALOGUE 4.10).

(DIALOGUE 4.10)

S> `Welcome to the travel agency!`

U> `visa information please` (0.78) (user actually said something else)

getLatestMoves
backupSharedUsr
integrateUsrAsk

findPlan
downdateQUD
backupSharedSys
selectIcmOther
selectIcmOther

S> `Okay. You want to know about price.`

getLatestMoves
integrateOtherICM
integrateUndPosICM

$$
\begin{bmatrix}
\textsc{private} & = & \begin{bmatrix}
\textsc{agenda} & = & \langle\langle\ \textsf{icm:loadplan}\ \rangle\rangle \\
\textsc{plan} & = & \left\langle \begin{array}{l}
\textsf{findout}(?A.\textsf{how}(A)) \\
\textsf{findout}(?B.\textsf{dest\_city}(B)) \\
\textsf{findout}(?C.\textsf{dept\_city}(C)) \\
\textsf{findout}(?D.\textsf{month}(D)) \\
\textsf{findout}(?E.\textsf{dept\_day}(E)) \\
\textsf{findout}(?F.\textsf{class}(F)) \\
\textsf{consultDB}(?G.\textsf{price}(G))
\end{array} \right\rangle \\
\textsc{bel} & = & \{\} \\
\textsc{tmp} & = & \begin{bmatrix}
\textsc{usr} & = & \begin{bmatrix}
\textsc{com} & = & \{\} \\
\textsc{qud} & = & \langle\rangle \\
\textsc{issues} & = & \langle\rangle \\
\textsc{agenda} & = & \langle\langle\rangle\rangle \\
\textsc{plan} & = & \langle\rangle
\end{bmatrix}
\end{bmatrix} \\
\textsc{nim} & = & \langle\langle\rangle\rangle
\end{bmatrix} \\
\textsc{shared} & = & \begin{bmatrix}
\textsc{com} & = & \{\} \\
\textsc{issues} & = & \langle\ ?H.\textsf{price}(H)\ \rangle \\
\textsc{qud} & = & \langle\ \textsf{und}(\textsf{usr*issue}(?I.\textsf{price}(I)))\ \rangle \\
\textsc{pm} & = & \{\ \textsf{ask}(?H.\textsf{price}(H))\ \} \\
\textsc{lu} & = & \begin{bmatrix}
\textsc{speaker} & = & \textsf{sys} \\
\textsc{moves} & = & \left\{ \begin{array}{l}
\textsf{icm:und*pos:usr*issue}(?I.\textsf{price}(I)) \\
\textsf{icm:acc*pos}
\end{array} \right\}
\end{bmatrix}
\end{bmatrix}
\end{bmatrix}
$$

U> `no`

getLatestMoves
accommodateQUD2Issues
{ push(/SHARED/ISSUES, und(usr*issue(?A.price(A))))
integrateNegIcmAnswer

```
{ pop(/PRIVATE/NIM)
  pop(/SHARED/ISSUES)
  if_do(in($/SHARED/COM, issue(?A.price(A))) or
     issue(?A.price(A))=issue(D) and in($/SHARED/ISSUES, D), [
     /SHARED/QUD := $/PRIVATE/TMP/USR/QUD
     /SHARED/ISSUES := $/PRIVATE/TMP/USR/ISSUES
     /SHARED/COM := $/PRIVATE/TMP/USR/COM
     /PRIVATE/AGENDA := $/PRIVATE/TMP/USR/AGENDA
     /PRIVATE/PLAN := $/PRIVATE/TMP/USR/PLAN ])
  push(/PRIVATE/AGENDA, icm:und*pos:usr*not(issue(?A.price(A))))
  clear(/PRIVATE/NIM)
  init_shift(/PRIVATE/NIM)
```

downdateQUD

```
[ PRIVATE = [ AGENDA = 〈〈 icm:loadplan                                   〉〉 ]
            [          〈〈 icm:und*pos:usr*not(issue(?A.price(A)))        〉〉 ]
            [ PLAN   = 〈〉                                                      ]
            [ BEL    = {}                                                      ]
            [ TMP    = [ USR = [ COM    = {}   ] ]                             ]
            [          [       [ QUD    = 〈〉   ] ]                             ]
            [          [       [ ISSUES = 〈〉   ] ]                             ]
            [          [       [ AGENDA = 〈〈〉〉 ] ]                             ]
            [          [       [ PLAN   = 〈〉   ] ]                             ]
            [ NIM    = 〈〈〉〉                                                    ]
  SHARED  = [ COM    = {}                                                      ]
            [ ISSUES = 〈〉                                                      ]
            [ QUD    = 〈〉                                                      ]
            [ PM     = { icm:und*pos:usr*issue(?A.price(A)), icm:acc*pos }     ]
            [ LU     = [ SPEAKER = usr                                     ]   ]
            [          [ MOVES   = { answer(not(und(usr*issue(?A.price(A))))) } ] ] ]
```

backupSharedSys
selectIcmOther

S> `You did not ask about price.`

Feedback for utterances realizing several moves

In the case where a user utterance provides answers to several questions (and perhaps also asks a question), the strategy we have chosen is to produce feedback for each move individually to give the user a chance to correct mishearings or misunderstandings. If positive feedback regarding one piece of information is rejected, the optimistic assumption is retracted by modifying the information state according to the /PRIVATE/TMP/USR saved state. If feedback regarding one piece of information is accepted, this information should not disappear if a later feedback utterance from the system should be rejected.

(DIALOGUE 4.11)

```
S> Welcome to the travel agency!
U> price information please
S> Okay.  I need some information.  How do you want to travel?
U> a flight to ferris [0.78]
S> Okay.  by flight.
U> yes
S> to paris.
U> no
S> not to paris. So, what city do you want to go to?
```

For example, in the dialogue in (DIALOGUE 4.11), the user accepts the system's feedback "by flight", but rejects "to paris"; however, the information that the user wants to travel by flight is retained.

To handle this, each time positive feedback is accepted, the parts of the /PRIVATE/TMP/USR structure corresponding to the SHARED field are modified. A further modification is thus needed for the **integratePosIcmAnswer** rule previously defined in Section 3.6.6.

```
(RULE 4.11) RULE: integratePosIcmAnswer
            CLASS: integrate
                   { $/PRIVATE/NIM/FST/SND=answer($A$)
                   { fst($/SHARED/ISSUES, $Q$)
            PRE:   { $DOMAIN :: resolves($A$, $Q$)
                   { fst($/SHARED/QUD, $Q$)
                   { $DOMAIN :: combine($Q$, $A$, $P$)
                   { $P$=und($DP$*$Content$)
                   { pop(/PRIVATE/NIM)
                   { pop(/SHARED/ISSUES)
                   { if_then_else($Content$=issue($Q'$), [
                   {     push(/PRIVATE/TMP/$DP$/QUD, $Q'$)
                   {     push(/PRIVATE/TMP/$DP$/ISSUES, $Q'$)
                   {     push(/PRIVATE/TMP/$DP$/AGENDA, respond($Q'$)) ],
                   {     add(/PRIVATE/TMP/$DP$/COM, $Content$))
            EFF:   { if_do(not ( in($/SHARED/COM, $Content$) or
                   {     $Content$=issue($Q'$) and in($/SHARED/ISSUES, $Q'$) ),
                   {     if_then_else($Content$=issue($Q'$), [
                   {         push(/SHARED/QUD, $Q'$)
                   {         push(/SHARED/ISSUES, $Q'$)
                   {         push(/PRIVATE/AGENDA, respond($Q'$)) ],
                   {     add(/SHARED/COM, $Content$)))
```

The conditions are similar to those of the previous version of the rule, except for inspecting ISSUES instead of QUD. The first two updates are also the same. The third update adds the content *Content* which is being grounded to TMP/USR (in case *DP* is usr, which it always is in IBiS3). This means that if future feedback (concerning the same utterance) from the system is rejected, the system will backtrack to a state where *Content* is integrated. The conditionals in the third and fourth updates reflect the fact that questions are integrated differently from propositions. The fourth update is similar to the third update in the previous version of the rule.

Implicit acceptance

Before we move on there is one more thing to consider. If the user does not reject the system's positive feedback concerning a piece of information, this is regarded as an implicit acceptance. Therefore, we also need to add a **noFollowup** rule, for cases where positive system feedback is not responded to at all (i.e. the user does not take the turn offered).

(RULE 4.12) RULE: **noFollowup**

CLASS: (none)

$$\text{PRE:} \left\{ \begin{array}{l} \$\textsc{input},\ \text{'TIMED\_OUT'} \\ \text{in}(\$/\textsc{shared}/\textsc{pm},\ \mathsf{icm{:}und{*}pos{:}usr{*}}\mathit{Content}) \end{array} \right.$$

$$\text{EFF:} \left\{ \begin{array}{l} \text{if\_then\_else}(\mathit{Content}{=}\mathsf{issue}(Q),\ [\\ \quad \text{push}(/\textsc{private}/\textsc{tmp}/\textsc{usr}/\textsc{qud},\ Q) \\ \quad \text{push}(/\textsc{private}/\textsc{tmp}/\textsc{usr}/\textsc{issues},\ Q) \\ \quad \text{push}(/\textsc{private}/\textsc{tmp}/\textsc{usr}/\textsc{agenda},\ \mathsf{respond}(Q))\ ], \\ \text{add}(/\textsc{private}/\textsc{tmp}/\textsc{usr}/\textsc{com},\ \mathit{Content})) \end{array} \right.$$

The first condition is true only if the user did not produce any utterance (that the system heard) during her latest turn[11]. The second condition checks that the moves performed in the previous utterance includes positive understanding feedback regarding *Content*. The first updates are identical to the third update in the **integratePosIcmAnswer** rule in Section 4.6.7.

Below is a dialogue example involving positive, implicit positive, and negative followups to system feedback.

(DIALOGUE 4.12)

```
S> Welcome to the travel agency!
U> price information please
S> Okay.  Lets see.  How do you want to travel?
U> a flight to paris in april
S> Okay.  by flight.
U> yes
S> to paris.
U>
S> in april.
U> no
S> not in april. What city do you want to go to?
```

Implicit questions and elliptical answers

In the case of implicit acceptance questions in English (and Swedish) it appears that they can be addressed by short answers; however, we cannot assume that all implicit issues can be addressed elliptically. The use of QUD for storing implicit issues relies on the fact

[11]See Section 3.6.6 for an explanation of 'TIMED_OUT'.

that questions on QUD have not necessarily been raised explicitly; however, questions on QUD are also by definition available for resolution of short answers. To represent implicit questions which cannot be addressed elliptically, a further local data structure for implicit questions under discussion would be needed.

4.7 Further implementation issues

In this section we describe parts of the implementation of IBiS3 which have not been discussed earlier in this chapter, and which are not directly reused from IBiS2.

4.7.1 Dialogue moves

For IBiS3, only one dialogue move has been added: ICM indicating accommodation of a dependent issue. In English, we have chosen "alright, you want to know about ..." to indicate that some inference has been performed, and that it has been successful. This choice is based on the intuition that this indicates some process inference which has concluded successfully; this should be regarded as a preliminary and temporary solution awaiting further corpus and usability studies.

- icm:accommodate:Q **:** Move if Q **:** Question

4.7.2 IBiS3 update module

Update rules

The main additions to the update rule collection needed to handle accommodation and reaccommodation were described above in Section 4.6.

In this section, we describe changes applied to other rules from IBiS2 to fit with the modified information state used by IBiS3.

Backing up tmp/usr

The TMP/USR field contains copies of parts of the information state as they were before the latest user utterance was integrated. If the optimistic assumption should turn out to be wrong, the TMP/USR field is used to undo the optimistic grounding assumption without the need for complex revision processing (see Section 4.6.7).

The **backupSharedUsr** in (4.8) is called each time an utterance is to be integrated and stores the current QUD, ISSUES, COM, PLAN and AGENDA fields; these are all potentially affected by the integration of the moves in the latest utterance, and are also important for determining what to do next.

This backtracking mechanism only applies to domain-level communication; user ICM moves are always optimistically assumed to be correctly understood and integration always succeeds. Since ICM "subdialogues", such as that in (DIALOGUE 4.13) are used to establish the fact that a previous user utterance was misunderstood by the system, it is important that TMP/USR is not overwritten during the subdialogue. For example, the **backupSharedUsr** rule should not trigger before integrating the user's "pardon" or the user's answer "`no`" to the system ICM "`by boat`".

(DIALOGUE 4.13)

S> `Okay.  Lets see.  How do you want to travel?`
U> `by boat` [0.76] (user actually said something else)
S> `Okay.  by boat.`
U> `pardon ?`
S> `Okay. by boat.`
U> `no`
S> `not by boat. So, how do you want to travel?`

(RULE 4.13) RULE: **backupSharedUsr**

CLASS: (none)

$$\text{PRE:}\begin{cases}\$\text{LATEST\_SPEAKER}=\mathsf{usr}\\ \$\text{LATEST\_MOVES}=Moves\\ \text{not in}(Moves,\ \mathsf{icm}{:}X)\\ \text{not in}(Moves,\ \mathsf{no\_move})\\ \text{not (fst}(\$/\text{SHARED/QUD},\ \mathsf{und}(\mathsf{usr}{*}C))\text{ and}\\ \quad\text{in}(A,\ \mathsf{answer}(D))\text{ and}\\ \quad\$\text{DOMAIN} :: \text{relevant}(D,\ \mathsf{und}(\mathsf{usr}{*}C)))\end{cases}$$

$$\text{EFF:}\begin{cases}/\text{PRIVATE/TMP/USR/QUD} := \$/\text{SHARED/QUD}\\ /\text{PRIVATE/TMP/USR/ISSUES} := \$/\text{SHARED/ISSUES}\\ /\text{PRIVATE/TMP/USR/COM} := \$/\text{SHARED/COM}\\ /\text{PRIVATE/TMP/USR/AGENDA} := \$/\text{PRIVATE/AGENDA}\\ /\text{PRIVATE/TMP/USR/PLAN} := \$/\text{PRIVATE/PLAN}\end{cases}$$

The first condition checks that the latest speaker was indeed the user; if not, the rule should of course not trigger. The next four conditions are used to prevent triggering in case of an ICM subdialog, i.e. if the user produced an ICM move or responded to one from the system. (Note that no_move may count as implicit ICM if the user does not respond to ICM from the system; see Section 4.6.7). The fifth condition checks if the user utterance contains an answer relevant to a grounding-question on QUD. The effects simply copy the contents of TMP/USR to the corresponding paths in the information state.

Integration rules and nim

In IBiS2, the integration rules inspect NIM using the condition in(/PRIVATE/NIM, *Moves*). Since TRINDIKIT uses backtracking to find instantiations of variables in conditions (see Appendix A), this results in each integration rule looking through the whole queue of non-integrated moves. Thus, in IBiS2 the ordering of the integration rules determines which move is integrated first. This is okay for dialogues with a very simple structure, but when dialogues become more complex (e.g. because of accommodation), the ordering of the moves becomes more important.

Therefore, in IBiS3 all integration rules inspect only the first move on the NIM queue, using the condition fst(/PRIVATE/NIM, *Move*) or similar. In combination with the queue-shifting technique described in Section 4.7.2, this means that the algorithm tries to integrate moves in the order they were performed.

Update algorithm

Because of the more complex dialogues handled by IBiS3, the update algorithm is a bit more complex than that for IBiS2.

(4.8)
```
1 if  not ($LATEST_MOVES == failed)
2 then ⟨ getLatestMoves,
3         try backupSharedUsr,
4         try irrelevantFollowup,
5         repeat ⟨
6                 repeat( ⟨ integrate,
7                           try downdate_issues,
8                           try removeFindout,
9                           try load_plan ⟩,
10                        orelse apply shift( /PRIVATE/NIM) )
11                until fully_shifted($/PRIVATE/NIM),
12                apply shift(/PRIVATE/NIM),
13                try select_action
14                accommodate ⟩,
15        apply cancel_shift( /PRIVATE/NIM ),
16        repeat exec_plan,
17        try downdate_qud ⟩
18 else ⟨ failedFollowup orelse unclearFollowup ⟩
```

Line 1 checks that the interpretation of the latest utterance was successful (of course, in the case of system utterances this is always true). If not, the **failedFollowup** and **unclearFollowup** rules in line 18, described in Section 3.6.8, are tried. If interpretation was successful, the latest moves are incorporated in the information state proper by the **getLatestMoves** rule (see Section 3.6.7). After this the **backupSharedUsr** rule is tried; its conditions are satisfied, the rule will trigger and store a copy of relevant parts of the information state in case the system makes an optimistic grounding assumption which turns out to be mistaken (see Section 4.7.2). Also, before integration starts, the **irrelevantFollowup** rule described in Section 3.6.8 is tried to catch cases where a system question has been ignored by the user.

After this, a loop involving integration and accommodation is executed until nothing more can be integrated (i.e. until the loop can no longer be executed). The basic idea is this: first try to integrate as many moves as possible by cycling through the NIM queue; then, if accommodation can be applied, do the same thing again. Repeat this until nothing can be integrated and no accommodation is possible.

The first part of this loop starts in line 6 and is itself a loop for cycling through all non-integrated moves and trying to integrate them. If integration succeeds, the algorithm tries to remove any resolved issues from ISSUES and PLAN, and if necessary load a new plan (e.g. if an ask move from the user was integrated). Then it tries integration again. If integration fails, the NIM queue is shifted one step, i.e. the topmost element is removed from the top and pushed to the end of the queue. Then, integration is tried again. This continues until the queue has been completely cycled through once, and all moves have had a shot at being integrated.

After this loop is finished, accommodation will attempt to adjust the /SHARED field so that any moves still not integrated may be understood on the pragmatic level, and integrated. However, we need to avoid a problem that arises as a consequence of having the integration rules handle pragmatic understanding, acceptance, and integration in a single step. The problem arises if some move is regarded as relevant (i.e. understood on the pragmatic level) but not acceptable, or if a relevant move has low reliability and should be verified before being integrated. In this case, accommodation should not be tried since the purpose of accommodation is to understand some utterance on the pragmatic level, and this has already been achieved. To solve this problem, some action selection rules (of class select_action) have been moved from the selection module to the update module (for a list of these rules, see Appendix B). Before trying accommodation, line 13 of the update algorithm thus tries to select rejection moves and interrogative feedback moves to catch any moves which have already been understood.

Line 14 calls the accommodation rule class. If this succeeds, there is a chance that some moves that could not be integrated before can now be integrated, so the loop starting in line 6 is restarted. When nothing can be integrated and nothing can be accommodated, the sequence starting at line 6 and ending at line 14 cannot be executed, and consequently the loop started in line 5 will be finished. Line 15 cancels shifting of the NIM queue (see Section A.2.1).

Any loaded plan is executed in line 16 by repeatedly applying the exec_plan rule class until no more execution is possible at the current stage of the dialogue. Finally, QUD is downdated.

As an example of how integration and accommodation interact, in the dialogue in (DIALOGUE 4.14), "to paris" is integrated before accommodation is tried, so the only question available for ellipsis resolution of "paris" is the one concerning departure city.

(DIALOGUE 4.14)

S> `Welcome to the travel agency!`
U> `price london to paris` [0.78]

```
S> Okay.  You want to know about price.
S> I need some information.  to paris.
S> from london.
S> How do you want to travel?
```

Accommodation rule ordering For the accommodate rule class, the ordering in which the various accommodation rules are tried may be important in some cases. The ordering used in IBiS3 is shown in (4.9).

(4.9) accommodate

1. **accommodateIssues2QUD**
2. **accommodateQUD2Issues**
3. **accommodatePlan2Issues**
4. **accommodateCom2Issues**
5. **accommodateCom2IssuesDependent**
6. **accommodateDependentIssue**

This order in which to try the accommodation rules has been chosen based on intuitions about how accessible questions are depending on where they are retrieved. By experimenting with the ordering, different behaviours can be obtained. The current ordering should be regarded as provisional, and finding the "best" ordering is an object for future research. It may also sometimes be necessary to do clarification if an answer matches several questions whose accommodation rules have the same or nearly the same priority; this has not been implemented in IBiS3.

Possible criteria for judging whether one ordering is better than another are (1) how reasonable the resulting behaviours are, (2) how efficient the overall processing becomes, and (3) how similar to human cognitive processes corresponding to accommodation the processing is (assuming question accommodation is cognitively plausible).

First, accommodation involving only ISSUES and QUD is tried, since these are the central structures for dealing with questions. If this fails, accommodation from the dialogue plan is tried; if this fails, reaccommodation from COM is attempted. First "normal" reaccommodation, then dependent reaccommodation. Finally, dependent issue accommodation is tried; this is tried last since it finds the question in the domain resource rather than the information state proper.

4.7.3 Selection module

The selection module is almost unchanged from IBiS2. Some minor adjustments have been made to adapt the rules to the changes in the information state type: that objects in NIM are pairs of DPs and moves, and that TMP is divided into two substructures.

4.8 Discussion

In this section we discuss some variations on IBiS3, show some additional "emergent" features, and discuss various aspects of question accommodation.

4.8.1 Phrase spotting and syntax in flexible dialogue

As it turns out, IBiS3 sometimes runs into trouble if the interpreter recognizes several answers to the same question in an utterance. Whereas IBiS2 would simply integrate the first answer and ignore the second, IBiS3 will try to make sense of all the moves in an utterance, which may lead to problems if the accommodation rules are not designed to cover the case at hand.

For example, if the system recognizes "paris to london" as a first utterance in a dialogue, the system will try dependent issue accommodation (see Section 4.6.4) and note that the set of answers (answer(**paris**) and answer(**dest_city**(**london**))) is (indirectly) relevant to both the price issue and the visa issue. It might seem that this is wrong, since the two answers are in fact relevant to the same question (regarding destination city) in the "visa" plan, whereas it is relevant to two separate questions (destination and departure city) in the "price" plan, so it should be indirectly relevant only to the "price" issue. But in general, one cannot require that the two answers must be answers to different questions, since the second answer may be a correction of the first. This may of course be signalled more clearly, as in "to paris uh no to london", but the correction signals may be left out, inaudible, or not recognized.

One way to solve this problem is to sometimes look for constructions which realize more than one move, and do some "cleaning up" in the interpretation phase so that the DME will not get into trouble. For example, we can add a lexical entry looking for phrases of the form "X to Y" and interpret this as "from X to Y", i.e. answer(**dept_city**(X)) andanswer(**dest_city**(Y)).

A related problem occurs if the user first chooses Gothenburg as departure city and then says "`not from gothenburg london`". Since plan-to-issues accommodation has precedence over com-to-issues, "london" will be integrated first by accommodating the destination city question, which is wrong. One solution is of course to give com-to-issues accommodation precedence, but then for "`paris from london`", "`from london`" will first be integrated and then "`paris`" will be seen as a revision of the departure city, which is also wrong.

As mentioned before in Section 4.7.2, the exact precedence ordering between accommodation rules is a topic for future research, and it may sometimes be necessary to do clarification if an answer matches several questions whose accommodation rules have the same or nearly the same priority. However, an easier solution is to add a further interpretation rule saying that "not P X, Y" should be interpreted as a paraphrase of "not P X, P Y".

A slightly irritating but not very serious "bug" in IBiS occurs if a user utterance contains two answers to the same question (e.g. "to kuala lumpur to london"), and the first of these is an invalid database parameter. The first answer will be rejected, and appropriate feedback will be put on the agenda. The second answer will then (correctly) replace the first answer using retraction, but the rejection feedback concerning the now replaced first answer remains on the agenda. This means that the system will give some irrelevant information, namely that the first answer was rejected. This can be fixed to some extent by interpreting phrases of the form "$PXPY$" as "PY", i.e. the second part is regarded as a correction of the first part. Similarly, phrases of the form "PX `no` $(P)Y$", where "`no`" is regarded as a correction indicator and the second P is optional, can also be interpreted as "PY". In general, it is useful to detect corrections in the interpretation phase to avoid potentially expensive revisions in the integration phase.

Of course, these simple fixes will only get us so far, since they only capture the very simplest cases. For example, we would not be able to notice that an utterance contains two answers to the same question unless they are adjacent. What is really needed here is a proper parser (e.g. a HPSG-based parser) and grammar.

What this shows us, then, is that taking syntax into account in interpretation becomes increasingly important when the dialogue management becomes more flexible. This might not be very surprising, but by starting out with very simple keyword spotting and keeping it as long as possible, we find out exactly where it becomes insufficient. Had we used a sophisticated parser from the outset, we would not have had this insight.

4.8.2 Relaxing constraints using denial and dependent reaccommodation

In inquiry-oriented dialogue it is often useful to be able to relax constraints on the database search, especially in cases where no result is returned from the database. As it turns out, existing accommodation mechanisms in IBiS3 in combination with the slightly more complex database result format which was introduced as an extension of IBiS2 (see Section 2.12.4) provides a way of doing exactly this.

The idea is this: if the system returns an empty or otherwise unsatisfactory database result, the user can modify the information given to the system by denying some part of it. This triggers reaccommodation of the dependent issue that the database search tried to find an answer to, and the corresponding plan is reloaded. Since some information is now missing, the system will again ask for it; the rest of the actions in the plan will be removed since the questions have already been answered. By rejecting the repeated question (e.g. by saying "It doesn't matter" or "I don't know"), the user makes the system search the database again, but this time with relaxed constraints.

(DIALOGUE 4.15)

```
S> Welcome to the travel agency!
U> a flight to paris from gothenburg in april
S> Alright.  You want to know about price.
S> Lets see.  Okay.  What day do you want to leave?
U> the fifth
S> Okay.  What class did you have in mind?
U> business class
$S> Okay.  Sorry, there is nothing matching your request about price.
U> okay, not business class then
S> You want to know about price.
$S> Lets see.  Okay.  So, What class did you have in mind?
U> it doesnt matter
S> The price is  7654  crowns. economy class.
```

The example in (DIALOGUE 4.15) illustrates the principle. To make this feature more useful, one would need to be able to give disjunctive answers ("I want to leave on the twenty-first or the twenty-second"), specify a range of allowable "parameter values" ("I want to leave between the twenty-first and the twenty-fifth"), and give answers which are unspecific in relation to the question ("I want to go to France" in response to "What city do you want to go to"). Of course, this is a useful feature also when providing answers in the initial database query specification stage. This is an area for further implementation.

The nice thing about this approach to relaxing constraints is that it requires no special machinery, but it emerges as a possibility given mechanisms which are independently motivated.

4.8.3 "Smart" interpretation

The dialogues in (DIALOGUE 4.16) and (DIALOGUE 4.17) show how the integration strategy in IBiS2 is able to use all the information provided in an utterance as clues for how to interpret underspecified information. The interpretation of "`paris`" as destination or departure city depends on what other information is given in the utterance, specifically whether destination or departure city is specified elsewhere in the utterance.

(DIALOGUE 4.16)

```
U> price information please
S> Okay.  Lets see.  How do you want to travel?
U> paris from london [0.78]
S> Okay.  from london.
S> to paris.
```

(DIALOGUE 4.17)

```
U> price information please
S> Okay.  Lets see.  How do you want to travel?
U> paris to london [0.78]
S> Okay.  to london.
S> from paris.
```

The sample dialogue in (DIALOGUE 4.18) shows how the system can integrate successive hints from the user about what she wants to do.

(DIALOGUE 4.18)

```
S> Welcome to the travel agency!
U> um i would like to go to paris
S> to paris.  I dont quite understand.  Do you mean to ask about visa or to
ask about price?
U> um i want to go by flight
S> Alright.  You want to know about price.
```

4.8.4 Separating understanding, acceptance, and integration

The integration rules in IBiS3 perform several tasks: contextual interpretation (e.g. for underspecified answers), deciding whether to accept or reject a move and their contents, and (if acceptance is decided on) integration of the full effects of the move. While this was a good approach in IBiS1 and IBiS2, in IBiS3 this approach sometimes obscures the workings of the system and make rules rather complex.

An alternative approach would be to implement contextual interpretation, the acceptance/rejection decision, and integration as separate rule classes. The contextual interpretation rules would take moves off a queue of moves provided by the interpretation module (corresponding to the current NIM field); we could call this queue of possibly underspecified moves SUM (Semantically Understood Moves). The resulting fully specified moves could then be added to a PUM (Pragmatically Understood Moves) queue, which would serve as input for the acceptance/rejection decision rules. Rejected moves would be put on a RM (Rejected Moves) queue, which would later be inspected in the selection phase to produce suitable feedback. Accepted moves would be added to an AM (Accepted Moves) stack, which in turn would serve as input to the integration rules. While this would probably require a larger number of rules and also some additional data structures in the information state, the complexity of the individual rules could be greatly reduced and the clarity of the overall processing would improve. It is also likely that this would lead to a less bug-prone and theoretically more satisfying implementation.

4.8.5 Accommodation and the speaker's own utterances

In this chapter we have been mainly concerned with issue accommodation as a way of interpreting utterances from the other DP (for a dialogue system, the user). But how does accommodation relate to the generation and integration of one's own utterances? This issues does not come up in IBiS since the system never produces utterances that can be expected to require accommodation on the part of the user (e.g. ending a long dialogue with "$100" rather than "The price is $100").

Ginzburg allows the speaker to update QUD with a question and then address it. This will (probably) require accommodation on the part of the hearer. The sequence of events here is roughly the following (S is the speaker, H the hearer):

- S pushes Q on QUD, then addresses Q
- S integrates A

- H accommodates Q, integrates A

However, we have noted above in Section 3.3.4 that this seems inconsistent with the view of QUD as something that is assumed to be shared. Possibly, one could have a "fuzzier" concept of QUD (and perhaps the DGB in general) that leaves some freedom of modifying it privately, as long as the hearer can be expected to accommodate these modifications.

The other alternative is to allow the speaker to generate utterances that do not exactly match the current information state, and then perform accommodation to integrate her own utterance. In this case, the sequence of events is instead:

- S addresses Q, believing that the information state can be adjusted (using accommodation) so as to make this utterance felicitous
- S and H accommodate Q and integrate A

Whether the choice between these two approaches make any real difference to the internal processing and/or external behaviour of the system remains a future research issue. For example, if QUD is updated with Q before A is produced, and the utterance realizing A is interrupted, should Q be removed from QUD?

4.8.6 Accommodation vs. normal integration

As we have seen, question accommodation allows a generalized account for how answers are integrated into the information state, regardless of the status of the corresponding question. The accommodation procedure may also have side-effects (e.g. loading a new dialogue plan) which serve to drive the dialogue forward.

Instead of giving rules for accommodation and integration separately, one could deny the existence of accommodation and just give more complex integration rules. The integration rule for short answers requires that there is a question on the QUD to which the latest move is an appropriate answer, and the accommodation rules are used if no such question can be found. The alternative is to skip the QUD requirement, thus incorporating the accommodation mechanisms into the integration rule, which would then split into several rules. For example, there would be one rule for integrating answers by matching them to questions in the plan directly.

Apart from the theoretical argument that question accommodation provides a generalization of the way answers are integrated, there are also practical motivations. In particular, the fact that several steps of accommodation may be necessary to integrate a single answer means that the total number of rules for integrating answers would be higher if accommodation was not used - one would need at least one integration rule for each possible combination of accommodation rules.

A further argument which is not explored in this thesis (but see Engdahl *et al.*, 1999) is that question presupposition and accommodation interact with intra-sentential information structure in interesting and useful ways.

4.8.7 Dependent issue accommodation in VoiceXML?

On a close reading of the VoiceXML specification (McGlashan *et al.*, 2001), it may appear that VoiceXML offers a mechanism similar to dependent issue accommodation[12]. In VoiceXML, a grammar can have scope over a single slot, over a form, or over a whole document (containing several forms). Given a grammar with document scope (defining a set of sentences which the VoiceXML interpreter will listen for during the whole dialogue), if the user gives information which does not match the currently active form, VoiceXML will jump to a form matching the input[13]. This corresponds roughly to the dependent issue accommodation mechanism in IBiS . However, if input matches more than one task (e.g. "raise the volume" could match a task related to the TV or one related to the CD player), VoiceXML will not ask which of these tasks the user wants to perform but instead go to the one it finds first, regardless of what the user intended. Generally, it is hard to see how clarification questions could be handled in a general way in VoiceXML, since they do not belong to a particular form.

4.9 Summary

To enable more flexible dialogue behaviour, we made a distinction between a local and a global QUD (referring to the latter as "open issues", or just "issues"). The notions of

[12]This discussion is based on the VoiceXML specification rather than hands-on experience of VoiceXML. This means that some unclarity remains about the capabilities of VoiceXML in general, and individual implementations of VoiceXML servers in particular. For both these reasons, the discussion should be regarded as tentative and open for revision. However, it should also be pointed out that it is fairly clear what is supported in VoiceXML; most of the unclarities refer to what is possible, but not explicitly supported, in VoiceXML. In general, it is more important to know what is supported by a standard than what is possible, since almost anything is possible in any programming environment (given a sufficient number of hacks).

[13]Although the VoiceXML documentation does not provide any examples of this kind of behaviour, it appears to be possible, at least in principle.

question and issue accommodation were then introduced to allow the system to be more flexible in the way utterances are interpreted relative to the dialogue context. Question accommodation allows the system to understand answers addressing issues which have not yet been raised. In cases of ambiguity, where an answer matches several possible questions, clarification dialogues may be needed.

Chapter 5

Action-oriented and negotiative dialogue

5.1 Introduction

In this chapter, we extend the issue-based approach to simple action-oriented and negotiative dialogue. First, we deal with action-oriented dialogue (AOD), which involves DPs performing non-communicative actions such as e.g. adding a program to a VCR or reserving tickets in a travel agency. We extend the IBiS system to handle a simple kind of AOD. In addition to issues and questions under discussion, this system also has to keep track of actions. Usually, it is useful for an AOD system to also handle IOD.

The concept of issue accommodation is extended to action accommodation. We also show how multiple simultaneous plans may be used to enable more complex dialogue structures, and how multiple plans interact with actions and issues. We show how dialogue plans may be constructed from menus, and illustrate menu-based AOD with examples from an implementation of a menu-based VCR interface.

Next, we turn to negotiative dialogue, and describe an issue-based account of a simple kind of collaborative negotiative dialogue. We also sketch a formalization of this account and discuss its implementation in IBiS.

5.2 Issues and actions in action-oriented dialogue

In IBiS3, each dialogue plan was aimed at resolving a specific issue. In general, of course, not all dialogue is aimed at resolving issues; often it is aimed towards the performance of some (non-communicative) action. For example, turning on or off the lights in a room, adding a program to a VCR, calling somebody up, or making a reservation in a travel agency. Action oriented dialogue in general places obligations on DPs to perform actions, either during the dialogue or after. For example, booking a ticket involves an obligation on the clerk to send a ticket to the customer, and on the customer to pay for the ticket. Requesting a VCR manager to add a program puts an obligation on the manager add the program to the VCR timer recording memory bank.

We will be dealing with a simple kind of AOD, where each action can only be performed by one of the DPs, similar to our assumptions regarding issues. This allows a simple representation of actions that does not take into account who has the obligation to perform each action. Since we are giving examples from a device control domain (VCR control), we will in fact only deal with the case where all actions are performed by the system[1].

Previous work with GoDiS, the predecessor of IBiS, has also addressed the case where the user performs all the actions (Larsson, 2000, Larsson and Zaenen, 2000).

5.3 Extending IBiS to handle action oriented dialogue

In this section, we describe additions to the information state, semantics, and dialogue moves. Update rules will be discussed in Section 5.6.

5.3.1 Enhancing the information state

In this section, we show how the IBiS information state needs to be modified to handle Action Oriented Dialogue. The new information state type is shown in Figure 5.1.

The only addition is the ACTIONS field which has been added to /SHARED and /PRIVATE/TMP. We assume the actions stack is an open stack, which is the same structure that we use for ISSUES.

[1]Of course, even in this simple domain it cannot really be assumed generally that the system performs all the actions; one could well imagine a VCR control dialogue system which, for example, requests the user to insert a tape into the VCR.

$$
\left[\begin{array}{lcl}
\textsc{private} & : & \left[\begin{array}{lcl}
\textsc{agenda} & : & \text{OpenQueue(Action)} \\
\textsc{plan} & : & \text{OpenStack(PlanConstruct)} \\
\textsc{bel} & : & \text{Set(Prop)} \\
\textsc{tmp} & : & \left[\begin{array}{lcl} \textsc{usr} & : & Tmp \\ \textsc{sys} & : & Tmp \end{array}\right] \\
\textsc{nim} & : & \text{OpenQueue(Pair(DP, Move))}
\end{array}\right] \\
\textsc{shared} & : & \left[\begin{array}{lcl}
\textsc{com} & : & \text{Set(Prop)} \\
\textsc{issues} & : & \text{OpenStack(Question)} \\
\textsc{actions} & : & \text{OpenStack(Action)} \\
\textsc{qud} & : & \text{OpenStack(Question)} \\
\textsc{pm} & : & \text{OpenQueue(Move)} \\
\textsc{lu} & : & \left[\begin{array}{lcl} \textsc{speaker} & : & \text{Participant} \\ \textsc{moves} & : & \text{Set(Move)} \end{array}\right]
\end{array}\right]
\end{array}\right]
$$

$$
Tmp = \left[\begin{array}{lcl}
\textsc{com} & : & \text{Set(Prop)} \\
\textsc{issues} & : & \text{OpenStack(Question)} \\
\textsc{actions} & : & \text{OpenStack(Action)} \\
\textsc{qud} & : & \text{OpenStack(Question)} \\
\textsc{agenda} & : & \text{OpenQueue(Action)} \\
\textsc{plan} & : & \text{OpenStack(PlanConstruct)}
\end{array}\right]
$$

Figure 5.1: IBiS4 Information State type

Semantics

To handle action AOD we need to extend our semantics. Given that α **:** Action, we have

- **action**(α) **:** Proposition
- **done**(α) **:** Proposition

Rough paraphrases of these propositions are "action α should be performed (by any DP who can perform α)", and "action α has been successfully performed", respectively.

Actions and postconditions

The set of actions that can be requested depends on the domain; for example, in the travel booking domain one action would be make_reservation, and an example from the VCR control domain is vcr_add_program. For dialogues where the the user requests actions to be performed by the system, each such action (which we may refer to as a *goal-action*) is associated with a dialogue plan.

In device control dialogue, there is also an additional kind of actions, namely those that are specified by the device itself; we refer to these as *device actions*. We will also generalize over device actions using the UPnP protocol ("Universal Plug'n'Play", Microsoft, 2000, Boye *et al.*, 2001, Lewin *et al.*, 2001); this requires a further type of *upnp action* whose arguments is a device and a device action. This allows us to access multiple devices defined using a common interface. This will be further clarified in Section 5.4.1.

Device actions and UPnP actions can be thought of as atomic actions, whereas goal actions are more complex; specifically, the execution of a single goal action (e.g. turning off all the lights in a room) may involve the execution of several device actions (e.g. turning off each individual light).

In addition to domain-specific goal actions and device actions, we still have the issue-related actions findout, raise and respond introduced in Chapter 2, and the set of dialogue moves.

For issues, the resolves relation provided a way to decide when an issue has been successfully performed and should be popped off the /SHARED/ISSUES stack. For actions, we instead need to define *postconditions* which are defined as relations between actions and propositions in the domain resource; these can then be used when to determine when an action can be removed from /SHARED/ACTIONS.

5.3.2 Dialogue moves

In addition to the dialogue moves introduced in Chapters 2 and 3, IBiS4 uses the following two moves:

- request(α), where α : Action
- confirm(α), where α : Action

These two moves are sufficient for activities where actions are performed instantly or near-instantly, and always succeed. If these requirements are not fulfilled, the confirm move can be replaced by or complemented with a more general report(α, *Status*) move which reports on the status of action α. Possible values of *Status* could be done, failed, pending, initiated etc.; report(α, done) would correspond to confirm(α).

5.4 Interacting with menu-based devices

As a sample subtype of action oriented dialogue we will explore menu-based AOD. While menu interfaces are ubiquitous in modern technology they are often tedious and frustrating. The mechanisms of accommodation introduced in Chapter 4 offers the possibility of allowing the user to present several pieces of relevant information at one time or to present information in the order in which the user finds most natural. This means that users can use their own conception of the knowledge space and not be locked to that of the designer of the menu system.

First, we describe a general method for connecting devices to IBiS, and then we show how menu interfaces can be converted into dialogue plans using a simple conversion schema.

5.4.1 Connecting devices to IBiS

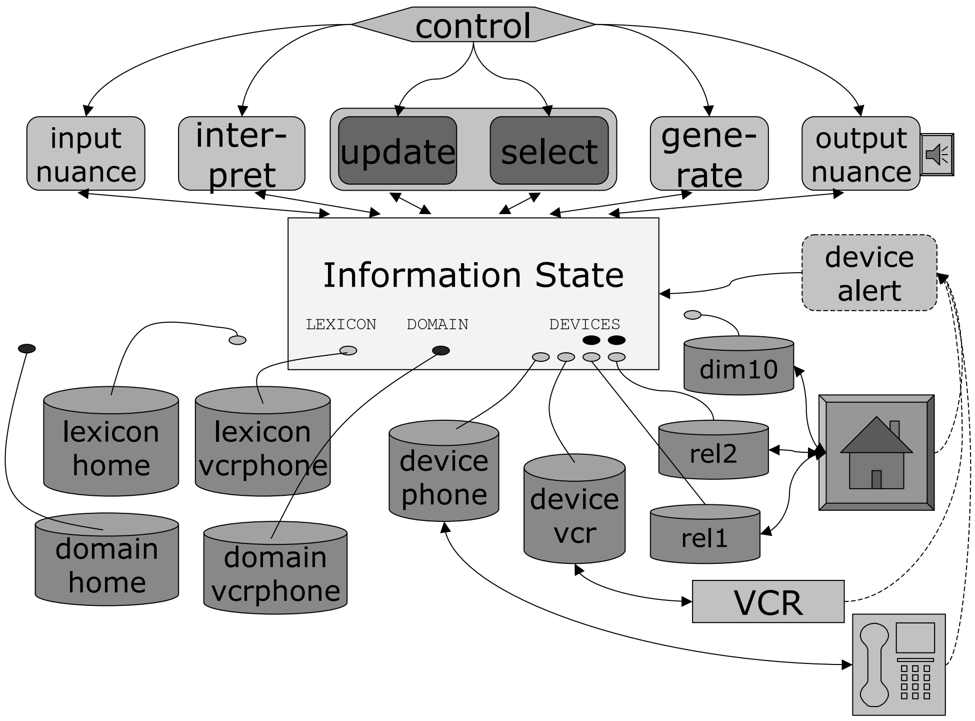


Figure 5.2: Connecting devices to IBiS

In this section we describe briefly how IBiS can interact with devices using the UPnP protocol. In Figure 5.2, we see an impression of how various devices can be connected to

IBiS. We will mainly be dealing with devices that can be modelled as resources, i.e. that are *passive* (or *reactive*) in the sense that they cannot send out information unless queried by some other module. Of course, many devices are not passive in this sense but rather *active* (or *pro-active*), e.g. burglar alarms or robots. To handle active devices, we would need to build a TrindiKit module which could write information to a designated part of the information state based on signals from the device; this information could then trigger various processes in other modules. Still, even for an active device the solution we present here would be very useful; minimally, we would only need to add a module which sets a flag in the information state whenever the device indicates that something needs to be taken care of, triggering other modules to query the device about exactly what has happened.

To be able to hook up passive UPnP devices to IBiS, we need the following:

1. device handler resources which communicate directly with the device itself; the device handlers can be said to represent the device in IBiS;
2. a resource type for UPnP devices, specifying how devices may be accessed as objects of this type;
3. a resource interface variable to the TIS whose values are of the UPnP resource type; this variable hooks up devices to the TIS;
4. plan constructs for interacting with devices, and update rules for executing these plan constructs;
5. dialogue plans for interacting with devices.

UPnP device handlers

The device handler mediates communication between IBiS and the device itself, and can be said to represent the device for IBiS. We assume that each specific device has a unique ID, and is accessed via a separate device handler process. A device handler is built for a certain device type (e.g. the Panasonic NV-SD200 VCR), and each device of that type needs to be connected to a process running the device handler, in order to be accessed by IBiS.

For UPnP devices, the device handler contains a specification partly derivable from the UPnP specifications, but made readable for IBiS (i.e. converted from XML to prolog).

The device handler does the following:

- specifies a set of actions and associated arguments

- specifies a set of variables, their range of allowed values, and (optionally) their default value
- routines for setting and reading variables (dev_set and dev_get), for performing queries (dev_query), and for executing actions (dev_do)
- accesses the devicesimulation

The UPnP resource interface

In order to hook up a device to IBiS one needs to define an abstract datatype for devices and declare a set of conditions and operations on that datatype. For IBiS, we implement a generic resource interface in the form of an abstract datatype for UPnP devices.

In UPnP, a device is defined in terms of

- a set of variables
- a set of actions with optional arguments

In addition to getting the value of a variable, setting a variable to a new value, and issuing a command, we also add the option of defining *queries* to the device. These queries allow more complex conditions to be checked, e.g. whether two variables have the same value.

Based on this we define the datatype upnp_dev as in (5.1); here, Var is a device variable; Val is the value of a device variable, $Query$ is a question, $Answer$ is a proposition, α_{dev} is a device action, and $PropSet$ is a set of propositions.

$$(5.1)\quad \begin{array}{l} \textsc{type}\text{: upnp\_dev} \\ \textsc{rel}\text{: } \left\{ \begin{array}{l} \text{dev\_get}(Var, Val) \\ \text{dev\_query}(Query,\ Answer) \end{array} \right. \\ \textsc{op}\text{: } \left\{ \begin{array}{l} \text{dev\_set}(Var,\ Val) \\ \text{dev\_do}(\alpha_{dev},\ PropSet) \end{array} \right. \end{array}$$

Device actions may have one or more parameters; for example, in the VCR control domain there is an action AddProgram which takes parameters specifying date, program number, start time, and end time. The *PropSet* argument of dev_do is a set of propositions, some of which may serve as arguments to α_{dev}. In the resource interface definition, this set is searched by the device interface for arguments. This means that *PropSet* is not the exact set of arguments needed for α_{dev}; rather, it is a repository of potential arguments.

The relation between UPnP actions, device actions, and device operations is exemplified below:

- dev_do(my_vcr, AddProgram) is a UPnP action, which may appear in a plan
- AddProgram is a device action
- dev_do(AddProgram, **{channel_to_store(1), start_time_to_store(13:45),** ... **}**) is a device update operation

In addition to the datatype definition, one can define objects to be of that datatype. For each device that the system should recognize, the device ID should be declared to be of type upnp_dev.

5.4.2 From menu to dialogue plan

Having describe a general method for connecting devices to IBiS, we will now show how menu interfaces can be converted into dialogue plans using a simple conversion schema. We assume menu interfaces consist of (at least) the following elements:

- *multi-choice lists*, where the user specifies one of several choices
- *dialogue windows*, where the user enters requested information using the keyboard
- *tick-box*, which the user can select or de-select
- *pop-up messages* confirming actions performed system

The correspondence between menu elements and plan constructs is shown in Table 5.1.

Regarding confirmations, we provide a general solution for confirming actions in Section 5.6.3. Confirmations thus do not need to be included in the plan.

5.4.3 Extending the resolves relation for menu-based AOD

In menu-based AOD, the system may ask an alternative-question about which action the user wants the system to perform. The user may then answer by choosing one of the listed

Menu construct	Plan construct
multi-choice list $\langle\alpha_1, \alpha_2, \ldots, \alpha_n\rangle$	action to resolve alternative question about action findout($\Big\{$ **?action**(α_1), …, **?action**(α_n) $\Big\}$)
tick-box or equivalent +/- P	action to resolve *y/n*-question findout($?P$)
dialogue window *parameter*=_	action to resolve *wh*-question findout($?x.parameter(x)$)
pop-up message confirming α	confirm(α)

Table 5.1: Conversion of menus into dialogue plans

alternatives. However, if the user selects an action which is not in the listed alternatives but further down in the hierarchy of actions, this should also be regarded as as an answer that resolves the system's question. To handle this, we need to extend the definition of the resolves relation (see Section 2.4.6).

(5.2) **action**(α) resolves **{?action**(α_1), …, **?action**(α_n)**}** if

- $\alpha = \alpha_i$ or
- α_i dominates α $(1 \leq i \leq n)$

The dominates relation is defined recursively as in (5.3).

(5.3) α dominates α' if

- there is a plan P for α such that P includes findout($AltQ$) and **?action**(α)$\in$ $AltQ$, or
- α dominates some action α'' and α'' dominates α'

The idea is, then, that domination reflects the menu structure so that an action dominates any actions below it in the menu.

5.5 Implementation of the VCR control domain

A VCR menu section

We start from a section of the menu structure for a VCR as shown in (5.4).

(5.4)

- toplevel: ⟨ change-play-status, change-channel, timer-recording, . . . ⟩
  - change play status: ⟨ play, stop, . . . ⟩
  - change channel
    - new-channel = _
    - confirm new channel
  - timer recording: ⟨ add-program, delete-program ⟩
    - add program
      - channel-to-store = _
      - date-to-store = _
      - start-time-to-store = _
      - end-time-to-store = _
      - confirm program added
    - delete program
      - display existing programs
      - program-to-delete: _
      - confirm program deleted
  - change-settings: ⟨ set-clock, . . . ⟩

Dialogue plans for VCR control

Using the conversion schema in Table 5.1 we can convert the menu structures in (5.4) into dialogue plans as those shown in (5.5).

(5.5) a. ACTION : vcr_top

PLAN: $\langle$

raise($?x.$**action**(x))

$$\text{findout}(\left\{\begin{array}{l}\textbf{?action(vcr\_change\_play\_status)}\\ \textbf{?action(vcr\_new\_channel)}\\ \textbf{?action(vcr\_timer\_recording)}\\ \textbf{?action(vcr\_settings)}\end{array}\right\})$$

$\rangle$

POST : -

b. ACTION : vcr_timer_recording

$$\text{PLAN: findout}(\left\{\begin{array}{l}\textbf{?action(vcr\_add\_program),}\\ \textbf{?action(vcr\_delete\_program)}\end{array}\right\})$$

POST : **done(vcr_add_program)** or **done(vcr_delete_program)**

c. ACTION : vcr_add_program

PLAN: $\langle$

findout($?x.$**channel_to_store**(x))

findout($?x.$**date_to_store**(x))

findout($?x.$**start_time_to_store**(x))

findout($?x.$**stop_time_to_store**(x))

dev_do(vcr, 'AddProgram')

$\rangle$

POST : **done('AddProgram')**

5.6 Update rules and dialogue examples

In this section we show how update rules for action oriented dialogue have been implemented in IBiS4, and give examples of dialogues from the VCR control domain.

5.6.1 Integrating and rejecting requests

First, we introduce update rules for integrating request moves. Since we are limiting this implementation to domains where the system performs all the actions, we will not provide rules for integrating requests from the system to the user; however, these could be straightforwardly implemented since the relation between system requests and user requests is very similar to the relation between system and user ask moves.

The rule for integrating user requests is shown in (RULE 5.1).

(RULE 5.1) RULE: **integrateUsrRequest**
CLASS: integrate

$$\text{PRE: }\left\{\begin{array}{l}\$/\textsc{private/nim/fst/snd}{=}\mathsf{request}(A)\\ \$/\textsc{shared/lu/speaker}{==}\mathsf{usr}\\ \$\textsc{score}{=}\mathit{Score}\\ \mathit{Score} > \mathsf{0.7}\\ \$\textsc{domain} :: \text{plan}(A,\ \mathit{Plan})\end{array}\right.$$

$$\text{EFF: }\left\{\begin{array}{l}\text{pop}(/\textsc{private/nim})\\ \text{add}(/\textsc{shared/lu/moves},\ \mathsf{request}(A))\\ \text{push}(/\textsc{private/agenda},\ \mathsf{icm{:}acc{*}pos})\\ \text{if\_do}(\mathit{Score} \leq \mathsf{0.9},\\ \quad \text{push}(/\textsc{private/agenda},\ \mathsf{icm{:}und{*}pos{:}usr{*}action}(A)))\\ \text{push}(/\textsc{shared/actions},\ A)\\ \text{push}(/\textsc{private/agenda},\ A)\end{array}\right.$$

This rule is similar to that for integrating user ask moves (see Section 3.6.6); instead of pushing an issue Q on ISSUES and QUD, and pushing respond(Q) on the agenda, this rule pushes the requested action A on /SHARED/ACTIONS and /PRIVATE/AGENDA.

As for user ask moves we also need to deal with the case where the system must reject an action since it does not have a plan for dealing with it. This rule is shown in (RULE 5.2).

(RULE 5.2) RULE: **rejectAction**
CLASS: select_action

$$\text{PRE: }\left\{\begin{array}{l}\text{in}(\$/\textsc{private/nim},\ \mathsf{request}(A))\\ \$/\textsc{shared/lu/speaker}{=}\mathsf{usr}\\ \text{not } \$\textsc{domain} :: \text{plan}(A,\ \mathit{Plan})\end{array}\right.$$

$$\text{EFF: }\left\{\begin{array}{l}\text{del}(/\textsc{private/nim},\ \mathsf{request}(A))\\ \text{push}(/\textsc{private/agenda},\ \mathsf{icm{:}und{*}pos{:}usr{*}action}(A))\\ \text{push}(/\textsc{private/agenda},\ \mathsf{icm{:}acc{*}neg{:}action}(A))\end{array}\right.$$

5.6.2 Executing device actions

The update rule for executing the dev_do device action is shown in (RULE 5.3).

(RULE 5.3)
$$\begin{array}{l}
\text{RULE: } \textbf{exec\_dev\_do} \\
\text{CLASS: } \mathsf{exec\_plan} \\
\text{PRE: } \left\{ \text{fst}(\$/\text{PRIVATE/PLAN}, \mathsf{dev\_do}(Dev, A_{dev})) \right. \\
\text{EFF: } \left\{ \begin{array}{l}
\text{pop}(/\text{PRIVATE/PLAN}) \\
!\ \$/\text{SHARED/COM}{=}PropSet \\
\text{DEVICES}/Dev :: \text{dev\_do}(PropSet, A_{dev}) \\
\text{add}(/\text{PRIVATE/BEL}, \mathsf{done}(A_{dev}))
\end{array} \right.
\end{array}$$

The condition looks for a dev_do upnp action in the plan, with arguments *Dev*, the device path name, and A_{dev}, the device action. The updates pop the action off the plan, and applies the corresponding update dev_do(*PropSet*, A_{dev}) to the device *Dev*. Finally, the proposition **done(**A_{dev}**)** is added the the private beliefs.

In addition, we have implemented rules for executing the dev_get, dev_set and dev_query actions.

5.6.3 Selecting and integrating confirm-moves

The selection rule for the confirm action is shown in (RULE 5.4).

(RULE 5.4)
$$\begin{array}{l}
\text{RULE: } \textbf{selectConfirmAction} \\
\text{CLASS: } \mathsf{select\_action} \\
\text{PRE: } \left\{ \begin{array}{l}
\text{fst}(\$/\text{SHARED/ACTIONS}, A) \\
\$\text{DOMAIN} :: \text{postcond}(A, PC) \\
\text{in}(\$/\text{PRIVATE/BEL}, PC) \\
\text{not in}(\$/\text{SHARED/COM}, PC)
\end{array} \right. \\
\text{EFF: } \left\{ \text{push}(/\text{PRIVATE/AGENDA}, \mathsf{confirm}(A)) \right.
\end{array}$$

The conditions in this rule check that the there is an action in /SHARED/ACTIONS whose postcondition is believed by the system to be true, however, this is not yet shared information. If this is true, a confirm action is pushed on the agenda. Eventually, this action (which also is a dialogue move) is moved to NEXT_MOVES by (RULE 5.5).

(RULE 5.5)
$$\begin{array}{l}
\text{RULE: } \textbf{selectConfirm} \\
\text{CLASS: } \mathsf{select\_move} \\
\text{PRE: } \left\{ \text{fst}(\$/\text{PRIVATE/AGENDA}, \mathsf{confirm}(A)) \right. \\
\text{EFF: } \left\{ \begin{array}{l}
\text{push}(\text{NEXT\_MOVES}, \mathsf{confirm}(A)) \\
\text{pop}(/\text{PRIVATE/AGENDA})
\end{array} \right.
\end{array}$$

When the confirmation move has been made, it is integrated by the rule in (RULE 5.6).

(RULE 5.6) RULE: **integrateConfirm**
CLASS: integrate
PRE: $\left\{ \text{\$/PRIVATE/NIM/FST/SND} = \mathsf{confirm}(A) \right.$
EFF: $\left\{ \begin{array}{l} \text{pop(/PRIVATE/NIM)} \\ \text{add(/SHARED/COM, } \mathsf{done}(A)\text{)} \end{array} \right.$

This rule adds the proposition **done(*A*)** to the shared commitments which enables the **downdateActions** rule in (RULE 5.7) to trigger.

(RULE 5.7) RULE: **downdateActions**
CLASS: downdate_issues
PRE: $\left\{ \begin{array}{l} \text{fst(\$/SHARED/ACTIONS, } A\text{)} \\ \text{\$DOMAIN :: postcond(}A\text{, } PC\text{)} \\ \text{in(\$/SHARED/COM, } PC\text{)} \end{array} \right.$
EFF: $\left\{ \text{pop(/SHARED/ACTIONS)} \right.$

This rule removes an action *A* whose postcondition is jointly believed to be true from ACTIONS[2].

5.6.4 Dialogue example: menu traversal and multiple threads

In (DIALOGUE 5.1) we show a sample dialogue interaction with the menu-based VCR application. It shows both menu traversal and accommodation, as well as dealing with multiple tasks (issues and actions).

(DIALOGUE 5.1)

S> `Welcome to the VCR manager!`

S> `Lets see.  What can I do for you?`

U>

[2]Note that **done(*α*)** is trivially a postcondition for any action *α*.

$S> Do you want to change play status , change channel , go to timer recording or go to settings?

U> go to timer recording

$S> Okay. Lets see. Do you want to add a program or delete a program?

U> add a program today

S> Okay. Lets see. What channel do you want?

U> what channel is on now

$$\left[\begin{array}{lll}
\text{PRIVATE} & = & \left[\begin{array}{lll} \text{AGENDA} & = & \langle\langle \text{ icm:acc*pos } \rangle\rangle \\ \text{PLAN} & = & \langle\rangle \\ \text{BEL} & = & \{ \text{ current\_channel(1) } \} \\ \text{TMP} & = & \ldots \\ \text{NIM} & = & \langle\langle\rangle\rangle \end{array}\right] \\
\text{SHARED} & = & \left[\begin{array}{lll} \text{COM} & = & \{ \text{ date\_to\_store(today) } \} \\ \text{ACTIONS} & = & \left\langle\begin{array}{l} \text{vcr\_add\_program} \\ \text{vcr\_timer\_recording} \\ \text{vcr\_top}\end{array}\right\rangle \\ \text{ISSUES} & = & \langle\ ?E.\text{current\_channel}(E)\ \rangle \\ \text{QUD} & = & \langle\rangle \\ \text{PM} & = & \{ \text{ ask}(?A.\text{channel\_to\_store}(A)),\ \text{icm:loadplan},\ \text{icm:acc*pos } \} \\ \text{LU} & = & \left[\begin{array}{lll}\text{SPEAKER} & = & \text{usr} \\ \text{MOVES} & = & \{ \text{ ask}(?E.\text{current\_channel}(E))\ \}\end{array}\right]\end{array}\right]
\end{array}\right]$$

S> Okay. The current channel is 1

S> Returning to add a program. What channel do you want?

U> one

S> Okay. What time do you want to start recording?

U> stop the vcr

device resource issues command to device: [UPnP] Stop

S> Okay. The VCR is now stopped.

S> Returning to add a program. What time do you want to start recording?

U> twelve thirty

S> `Okay. What time do you want to stop recording?`

U> `two fifteen`

device resource issues command to device: [UPnP] AddProgram(1,today,1230,215)

S> `Okay. The program has been added.`

S> `Returning to timer recording. Do you want to add a program or delete a program?`

N.B.: In this dialogue, the user's question "What channel is on now" is regarded as an irrelevant followup to "What channel do you want?". To change this one could add a dependency between these questions to the domain resource (see Section 2.8.2), as in (5.6).

(5.6) $\mathsf{depends}(?x.\mathbf{channel\_to\_add}(x),\ ?x.\mathbf{current\_channel}(x))$

5.6.5 Action accommodation and clarification

We also implement an action-related counterpart to the rule **accommodateDependentIssue** presented in Section 4.6.4. This allows the user to provide parameters for some action without making an explicit request.

```
(RULE 5.8)  RULE: accommodateAction
            CLASS: accommodate
                   ⎧ setof(A, $/PRIVATE/NIM/ELEM/SND=answer(A), AnsSet)
                   ⎪ $$arity(AnsSet) > 0
                   ⎪ $DOMAIN :: plan(Action, Plan)
                   ⎪ $DOMAIN :: action(Action)
                   ⎪ forall(in(AnsSet, A), in(Plan, findout(Q)) and
            PRE:   ⎨     $DOMAIN :: relevant(A, Q))
                   ⎪ not $DOMAIN :: plan(Action′, Plan′) and Action′≠Action and
                   ⎪     forall(in(AnsSet, A), in(Plan′, findout(Q)) and
                   ⎪     $DOMAIN :: relevant(A, Q))
                   ⎩ not in($/PRIVATE/AGENDA, icm:und*int:usr*action(Action))
                   ⎧ push(/SHARED/ACTIONS, Action)
                   ⎪ push(/PRIVATE/AGENDA, icm:accommodate:Action)
            EFF:   ⎨ push(/PRIVATE/AGENDA, icm:und*pos:usr*action(action))
                   ⎪ set(/PRIVATE/PLAN, Plan)
                   ⎩ push(/PRIVATE/AGENDA, icm:loadplan)
```

This rule is very similar to the **accommodateDependentIssue** (see Section 4.6.4), except that it accommodates a dependent action rather than a dependent issue.

If the system finds several actions matching the information given by the user, a clarification question is raised. This is again similar to the behaviour for issues described in Section 4.6.5; in fact, the rule below replaces the previous **clarifyDependentIssue** rule.

```
(RULE 5.9)  RULE: clarifyIssueAction
            CLASS: select_action
                   ⎧ in($/PRIVATE/NIM, pair(usr, answer(A)))
                   ⎪ setof(Action, $DOMAIN :: depends(α, Q) and
            PRE:   ⎨     $DOMAIN :: relevant(A, Q), ActionSet)
                   ⎪ remove_unifiables(Actions, Actions′)
                   ⎩ $$arity(Actions′) > 1
                   ⎧ ! setof(?IssueProp, in(Actions′, Issue′) and
                   ⎪     not $DOMAIN :: action(Issue′) and
                   ⎪     IssueProp=issue(Issue′), IssueQuestions)
                   ⎪ ! setof(?ActionProp, in(Actions′, Action′) and
            EFF:   ⎨     $DOMAIN :: action(Action′) and
                   ⎪     ActionProp=action(Action), ActionQuestions)
                   ⎪ ! union(IssueQuestions, ActionQuestions, AltQ)
                   ⎩ push(/PRIVATE/AGENDA, findout(AltQ))
```

The preconditions check that there is more than one plan (for performing actions or resolving issues) to which all answers in /PRIVATE/NIM are relevant. The first update creates an alternative question regarding all issues which the non-integrated answers are relevant to, and the second update does the same for actions. The third update joins these two alternative-questions (sets of *y/n*-questions) into a single *y/n*-question, which is added to the agenda by the final update.

5.6.6 Dialogue examples: action accommodation and clarification

The dialogue in (DIALOGUE 5.2) demonstrates action accommodation, and the dialogue in (DIALOGUE 5.3) demonstrates a clarification question by the system.

(DIALOGUE 5.2)

```
S> Welcome to the VCR manager!
S> Lets see.  What can I do for you?
U> today from five thirty on channel one
S> Alright.  add a program.
S> Lets see.  Okay.  What time do you want to stop recording?
```

(DIALOGUE 5.3)

```
S> Welcome to the VCR manager!
S> Lets see.  What can I do for you?
U> six thirty
$S> six thirty.  I dont quite understand.  Do you want to add a program
or set the clock?
U> add a program
$S> Okay.  Lets see.  Do you want to record from six thirty or until
six thirty?
U> from six thirty
S> Okay.  What channel do you want?
```

5.7 Issues under negotiation in negotiative dialogue

We will now turn to negotiative dialogue, and describe an issue-based account of a simple kind of collaborative negotiative dialogue. We also sketch a formalization of this account and discuss its implementation in IBiS.

We start from a previous formal account of negotiative dialogue (Sidner, 1994a) and argue for a slightly different idea of what negotiative dialogue is. We want to make a distinction between the process of accepting an utterance and its content, which applies to all utterances, and a concept of negotiation defined, roughly, as a discussion of several alternative solutions to some problem. This latter account is formulated in terms of *Issues Under Negotiation* (IUN), representing the question or problem to be resolved, and a set of alternative answers, representing the proposed solutions.

First, we will give a brief review of Sidner's theory and discuss its merits and drawbacks[3]. We then provide an alternative account based on the concept of Issues Under Negotiation. We explain how IUN can be added to IBiS, and give an information state analysis of a simple negotiative dialogue.

5.7.1 Sidner's theory of negotiative dialogue

As the title of the paper says, Sidner's (1994a) theory is formulated as "an artificial discourse language for collaborative negotiation". This language consists of a set of messages (or message types) with propositional contents ("beliefs"). The effects of an agent transmitting these messages to another agent is formulated in terms of the "state of communication" after the message has been received. The state of communication includes individual beliefs and intentions, mutual beliefs, and two stacks for Open Beliefs and Rejected Beliefs. Some of the central messages are

- ProposeForAccept `(PFA agt1 belief agt2)`: `agt1` expresses `belief` to `agt2`.
- Reject `(RJ agt1 belief agt2)`: `agt1` does not believe `belief`, which has been offered as a proposal
- AcceptProposal `(AP agt1 belief agt2)`: `agt1` and `agt2` now hold `belief` as a mutual belief

[3]An in-depth description of Sidner's account and its relation to the GoDiS system, including a reformulation of Sidner's artificial negotiation language in terms of GoDiS information state updates, can be found in Cooper *et al.* (2001).

- Counter `(CO agt1 belief1 agt2 belief2)`: Without rejecting `belief1`, `agt1` offers `belief2` to `agt2`

In addition, there are three kinds of acknowledgement messages, the most important being `AcknowledgeReceipt (AR agt1 belief agt2)`, which may occur after a `ProposeFor-Accept` message and results in `belief` being pushed on the stack for Open Beliefs. Acknowledgement indicates that a previous message from `agt2` about `belief` has been heard; the agents will not hold `belief` as a mutual belief until an AcceptProposal message has been sent.

While we will not give a detailed analysis of the effects of each of these messages, some observations are important for the purposes of this paper. Specifically, a counter-proposal `(CO agt1 belief1 agt2 belief2)` is analyzed as a composite message consisting of two `PFA` messages with propositional contents. The first proposed proposition is `belief2` (the "new" proposal), and the second is `(Supports (Not belief1) belief2)`, i.e. that `belief2` supports the negation of `belief1` (the "old" proposal). Exactly what is meant by "supports" here is left unspecified, but perhaps logical entailment is at least a simple kind of support.

- `(PFA agt1 belief2 agt2)`
- `(PFA agt1 (Supports (Not belief1) belief2) agt2)`

Sidner's analysis of proposals is only concerned with propositional contents. A Request for action is modelled as a proposal whose content is of the form `(Should-Do Agt Action)`. A question is a proposal for the action to provide certain information. This brings us to our first problem with Sidner's account.

Problem 1: Negotiation vs. utterance acceptance

In Sidner's theory, all dialogue is negotiative in the sense that all utterances (except acceptances, rejections, and acknowledgements) are seen as proposals. This is correct if we consider negotiation as possibly concerning meta-aspects of the dialogue. Since any utterance (content) can be rejected, all utterances can indeed be seen as proposals.

So in one sense of "negotiative", all dialogue is negotiative since assertions (and questions, instructions etc.) can be rejected or accepted as part of the grounding process. But some dialogues are negotiative in another sense, in that they contain explicitly discussions about different solutions to a problem. Negotiation, on this view, is distinct from grounding.

There is thus a stronger sense of negotiation which is not present in all dialogue. A minimum requirement on negotiation in this stronger sense could be that several alternative solutions (answers) to a problem (question or issue) can be discussed and compared before a solution is finally settled on. Sidner is aware of this aspect of negotiation, and notes that "maintaining more than one open proposal is a common feature of human discourses and negotiations." What we want to do is to find a way of capturing this property independently of grounding and of other aspects of negotiation, and use it as a minimal requirement on any dialogue that is to be regarded as negotiative.

On our view, proposal-moves are moves on the same level as other dialogue moves: greetings, questions, answers etc., and can thus be accepted or rejected on the grounding level. Accepting a proposal-move on the grounding level merely means accepting the content of the move *as a proposal*, i.e. as a potential answer to a question. This is different from accepting the proposed alternative as the *actual* solution to a problem (answer to a question).

To give a concrete example of these different concepts of negotiativity, we can compare the dialogues in Examples (5.5) and (5.6).

(5.7) A : Today is January 6th.
propose proposition
B(alt. 1) : Uhuh
accept proposition
B(alt. 2) : No, it's not!
reject proposition

(5.8) S : where do you want to go?
ask question
U : flights to paris on september 13 please
answer question
S : there is one flight at 07:45 and one at 12:00
propose alternatives, give information about alternatives
U : what airline is the 12:00 one
ask question
S : the 12:00 flight is an SAS flight
answer question
U : I'll take the 7:45 flight please
accept alternative, answer question "which flight?"

The type negotiation in (5.7) concerns acceptance-level grounding of the utterance and its content. By contrast, the type of negotiation in (5.8) concerns domain-level issues rather than some aspect of grounding.

Problem 2: Alternatives and counterproposals

When analyzing a travel agency dialogue (Sidner, 1994b), the travel agent's successive proposals of flights are seen as counterproposals to his own previous proposals, each modelled as a proposition. The difference between proposals and counterproposals is that the latter not only make a new proposal but also proposes the proposition that the new proposal conflicts with the previous proposal (by supporting the negation of the previous proposal). This can be seen as an attempt by Sidner to establish the connection between the two proposals as somehow concerning the same issue.

This analysis is problematic in that it excludes cases where alternatives are not mutually exclusive, which is natural when e.g. booking a flight (since the user presumably only want one flight) but not e.g. when buying a CD (since the user may want to buy more than one). Also, it seems odd to make counterproposals to your own previous proposals, especially since making a proposal commits you to intending the addressee to accept that proposal rather than your previous ones. In many cases (including the travel agency domain) it seems that the agent may often be quite indifferent to which flight the user selects. Travel agents may often make several proposals in one utterance, e.g. "There is one flight at 7:45 and another one at 12:00", in which case it does not make sense to see "one at 12:00" as a counterproposal as Sidner defines them.

We do not want to use the term "counterproposal" in these cases; what we need is some way of proposing alternatives without seeing them as counterproposals. The basic problem seems to be that when several proposals are "on the table" at once, one needs some way of representing the fact that they are not independent of each other. Sidner does this by adding propositions of the form `(Supports (Not belief1) belief2)` to show that belief1 and belief2 are not independent; however, this proposition not only claims that the propositions are somehow dependent, but also that they are (logically or rhetorically) mutually exclusive. In our view, this indicates a need for a theory of negotiation which makes it possible to represent several alternatives as somehow *concerning the same issue*, independently of rhetorical or logical relations between the alternatives. Negotiation, in our view, should not in general be seen in terms of proposals and counterproposals, but in terms of proposing and choosing between several alternatives.

5.7.2 Negotiation as discussing alternatives

In this section, we will attempt to provide a more detailed description of negotiative dialogue. Clearly, negotiation is a type of problem-solving (Di Eugenio *et al.*, 1998). We define negotiative dialogue more specifically to be *dialogue where DPs discuss several alternative solutions to a problem (issue) before choosing one (or several) of them.* In line with our issue-based approach to dialogue management, we propose to model negotiable problems (issues) semantically as questions and alternative solutions as alternative answers to a question.

We also propose to keep track of issues under negotiation and the answers being considered as potential solutions to each issue in the /SHARED/ISSUES field, represented as questions associated with sets of answers.

Degrees of negotiativity

Starting from this definition, we can distinguish between fully negotiative dialogue and semi-negotiative dialogue (see also Section 2.1.2). In non-negotiative dialogue, only one alternative can be discussed. In semi-negotiative dialogue, a new alternative can be introduced by revising parameters of the previous alternative; however, previous alternatives are not retained. Finally, in negotiative dialogue: several alternatives can be introduced, and old alternatives are retained and can be returned to.

Semi-negotiative information-oriented dialogue does not require keeping track of several alternatives. All that is required is that information is revisable, and that new database queries can be formed from old ones by replacing some piece of information. This property is implemented in a limited way for example in the Swedish railway information system (a variant of the Philips system described in Aust *et al.*, 1994), which after providing information about a trip will ask the user "Do you want an earlier or later train?". This allows the user to modify the previous query (although in a very limited way) and get information about further alternatives. However, it is not possible to compare the alternatives by asking questions about them; indeed, there is no sign that information about previous alternatives is retained in the system. The implementation of reaccommodation in IBiS3 (Section 4.6.6) also allowed semi-negotiative dialogue in this sense.

Factors influencing negotiation

There are a number of aspects of the dialogue situation which affect the complexity of negotiative dialogues, and allows further sub-classification of them. This sub-classification allows us to pick out a subspecies of negotiative dialogue to implement.

On our definition, negotiation does not require conflicting goals or interests, and for this reason it may not correspond perfectly to the everyday use of the word "negotiation". However, we feel it is useful to keep collaborativity (i.e. lack of conflicting goals) as a separate dimension from negotiation. Also, it is common practice in other fields dealing with negotiation (e.g. game theory, economy) to include collaborative negotiation (cf. Lewin *et al.*, 2000).

A second factor influencing negotiation is the distribution of information between DPs. In some activities, information may be symmetrically distributed, i.e. DPs have roughly the same kind of information, and also the same kind of information needs (questions they want answered). This is the case e.g. in the Coconut (Di Eugenio *et al.*, 1998) dialogues where DPs each have an amount of money and they have to decide jointly on a number of furniture items to purchase. In other activities, such as a travel agency, the information and information needs of the DPs is asymmetrically distributed. The customer has access to information about her destination, approximate time of travel etc., and wants to know e.g. exact flight times and prices. The travel agent has access to a database of flight information, but needs to know when the customer wants to leave, where she wants to travel, etc.

A third variable is whether DPs must commit jointly (as in e.g. the Coconut dialogues) or one DP can make the commitment by herself (as e.g. in flight booking). In the latter case, the acceptance of one of the alternatives can be modelled as an answer to an IUN by the DP responsible for the commitment, without the need for an explicit agreement from the other DP. In the former case, a similar analysis is possible, but here it is more likely that an explicit expression of agreement is needed from both DPs. This variable may perhaps be referred to as “distribution of decision rights”. In some dialogues (such as ticket booking) one DP has the decision rights for all negotiable issues; in this case there is no need for explicitly representing decision rights. However, if decision rights are distributed differently for different issues, an explicit representation of rights is needed.

Ticket booking dialogue, and dialogue in other domains with clear differences in information and decision-right distribution between roles, has the advantage of making dialogue move interpretation easier since the presence of a certain bits of information in an utterance together with knowledge about the role of the speaker and the role-related information distribution often can be used to determine dialogue move type. For example, an utterance containing the phrase “to Paris” spoken by a customer in a travel agency is likely to be intended to provide information about the customer’s desired destination.

5.7.3 Issues Under Negotiation (IUN)

In this section we discuss the notion of Issues Under Negotiation represented by questions, and how proposals relate to such issues. We also discuss how this approach differs from Sidner’s.

Negotiable issues and activity

Which issues are negotiable depends on the activity. For example, it is usually not the case that the name of a DP is a negotiable issue; this is why it would perhaps seem counterintuitive to view an introduction ("Hi, my name is NN") as a proposal (as is done in Sidner, 1994b). However, it cannot be ruled out that there is some activity where even this may become a matter of negotiation. Also, it is usually possible in principle to make any issue into a negotiable issue, e.g. by raising doubts about a previous answer (see Section 5.8.2) .

Alternatives as answers to Issues Under Negotiation

Given that we analyze Issues Under Negotiation as questions, it is natural to analyze the alternative solutions to this issue as potential answers. On this view, a proposal has the effect of adding an alternative answer to the set of alternative answers to an IUN. For a DP with decision rights over an IUN, giving an answer to this IUN is equivalent to accepting one of the potential answers as the actual answer. That is, an IUN is resolved when an alternative answer is accepted.

Here we see how our concept of acceptance differs from Sidner. On our view a proposed alternative can be accepted in two different ways: as a proposal, or as *the* answer to an IUN. Accepting a proposal move as adding an alternative corresponds to meta-level acceptance. However, accepting an alternative as the answer to an IUN is different from accepting an utterance. Given the optimistic approach to acceptance, all proposals will be assumed to be accepted *as proposals*; however, it takes an answer-move to get the proposed alternative accepted as the solution to a problem.

Semantics

To represent issues under negotiation, we will use pairs of questions (usually *wh*-questions but possibly also *y/n*-questions) and sets of proposed answers. This is in fact an alternative representation of alternative-questions to that which we have used previously. The additional semantic representation is shown in (5.9).

(5.9) $Q \bullet AnsSet$: AltQ if Q : WHQ (or Q : YNQ) and $AnsSet$: Set(ShortAns)

5.7.4 An example

In the (invented) example in Figure 5.3, the question on ISSUES is $?x.$**desired_flight**(x), i.e. "Which flight does the user want?". The user supplies information about her desired destination and departure date; this utterance is interpreted as a set of answer-moves by the system since it provides answers to questions that the system has asked or was going to ask. As a response to this, the system performs a database search which returns two flights **f1** and **f2** matching the specification, and stores the database results in /PRIVATE/BEL. The system then proposes these flights as answers to the current IUN. The system also supplies some information about them. As a result, the IUN is now associated with two alternative answers, **f1** and **f2**. Finally, the user provides an answer to the current IUN, thereby accepting one of these alternatives as the flight she wants to take.

This dialogue does not include any discussion or comparison of alternatives, but it could easily be extended to cover e.g. the dialogue in (5.8).

5.8 Discussion

5.8.1 Negotiation in inquiry-oriented dialogue

The model presented here is not committed to the view that negotiation only takes place in the context of collaborative planning, or even action-oriented dialogue. In the sense of negotiative dialogue used here, i.e. dialogue involving several alternative solutions to some problem, negotiation may also concern matters of fact. This can be useful e.g. in tutorial dialogue where a tutor asks a question, gives some alternative answers, and the student's task is to reason about the different alternatives and decide on one of them. In the travel agency domain, it is often not necessary to explicitly represent e.g. that deciding on a flight is a precondition of a general plan for travelling; instead, we can represent it simply as a fact concerning which flight the user wants to take.

A related point is that collaborative planning dialogue is not necessarily action-oriented dialogue, since the activity of planning may be directed at coming up with an abstract plan regardless of who actually performs the actions in the plan. Only when some DP becomes obliged to carry out some part of the plan does the dialogue become what we refer to as an action-oriented dialogue.

A> `flights to paris, june 13`
answer(**desired_dest_city(paris)**)
answer(**desired_dept_date(13/5)**)

B> `OK, there's one flight leaving at 07:45 and one at 12:00`
propose(**f1**)
propose(**f2**)
inform(**dept_time(f1,07:45)**)
inform(**dept_time(f2,12:00)**)

$$\left[\begin{array}{lll}
\text{PRIVATE} & = & \left[\begin{array}{lll}
\text{AGENDA} & = & \langle\ \text{findout}(?x.\textbf{desired\_flight}(x))\ \rangle \\
\text{PLAN} & = & \left\langle \begin{array}{l} \text{findout}(?x.\textbf{credit-card-no}(x)) \\ \text{updateDB(add\_reservation)} \end{array} \right\rangle \\
\text{BEL} & = & \left\{ \begin{array}{l} \textbf{flight(f1)} \\ \textbf{dept\_time(f1,0745)} \\ \ldots \end{array} \right\}
\end{array}\right] \\
\text{SHARED} & = & \left[\begin{array}{lll}
\text{COM} & = & \left\{ \begin{array}{l} \textbf{dept\_time(f1,0745)} \\ \textbf{dept\_time(f2,1200)} \\ \textbf{desired\_dest\_city(paris)} \\ \textbf{desired\_dept\_date(13/5)} \ldots \end{array} \right\} \\
\text{ISSUES} & = & \langle\ ?x.\textbf{desired\_flight}(x) \bullet \{\ \textbf{f1}, \textbf{f2}\ \}\ \rangle \\
\text{ACTIONS} & = & \langle\ \text{book\_ticket}\ \rangle \\
\text{XS QUD} & = & \langle\rangle \\
\text{LU} & = & \left[\begin{array}{lll}
\text{SPEAKER} & = & \text{sys} \\
\text{MOVES} & = & \left\{ \begin{array}{l} \text{propose}(\textbf{f1}) \\ \text{propose}(\textbf{f2}) \\ \ldots \end{array} \right\}
\end{array}\right]
\end{array}\right]
\end{array}\right]$$

A> `I'll take the 07:45 one`
answer(**desired_flight**(X)**&dept_time**(X, **07:45**))
(after contextual interpretation: answer(**desired_flight(f1)**))

$$\left[\begin{array}{lll}
\text{PRIVATE} & = & \left[\begin{array}{lll}
\text{AGENDA} & = & \langle\ \text{findout}(?x.\textbf{credit-card-no}(x))\ \rangle \\
\text{PLAN} & = & \langle\ \text{updateDB(add\_reservation)}\ \rangle \\
\text{BEL} & = & \left\{ \begin{array}{l} \textbf{flight(f1)} \\ \textbf{dept\_time(f1,0745)} \\ \ldots \end{array} \right\}
\end{array}\right] \\
\text{SHARED} & = & \left[\begin{array}{lll}
\text{COM} & = & \left\{ \begin{array}{l} \textbf{desired\_flight(f1)} \\ \textbf{dept\_time(f1,0745)} \\ \textbf{dept\_time(f2,1200)} \\ \textbf{desired\_dest\_city(paris)} \\ \textbf{desired\_dept\_date(13/5)} \ldots \end{array} \right\} \\
\text{ISSUES} & = & \langle\rangle \\
\text{ACTIONS} & = & \langle\ \text{book\_ticket}\ \rangle \\
\text{QUD} & = & \langle\rangle \\
\text{LU} & = & \left[\begin{array}{lll}
\text{SPEAKER} & = & \text{sys} \\
\text{MOVES} & = & \left\{ \begin{array}{l} \text{answer}(\textbf{desired\_flight(f1)} \\ \ldots \end{array} \right\}
\end{array}\right]
\end{array}\right]
\end{array}\right]$$

Figure 5.3: Example dialogue

5.8.2 Rejection, negotiation and downshifting

In the context of discussing referent identification in instructional assembly dialogues, Cohen (1981) makes an analogy between shifts in dialogue strategy and shifting gears when driving a car. In a dialogue in high gear, the speaker introduces several subgoals in each utterance, whereas fewer goals are introduced in low-gear dialogue. The type of subgoals discussed by Cohen are mainly identifying a referent, requests to pick up objects, and requesting an assembly action. As long as the dialogue proceeds smoothly and the hearer is able to correctly identify referents and carry out actions, the speaker requests assembly actions and expects the hearer to be able to identify and pick up the objects referred to without explicit requests for this. However, when this fails and the hearer fails to identify a referent, the speaker may shift into a lower gear (downshift) and make explicit requests for identification of referents. At a later stage, the speaker may shift to a higher gear and request the hearer to pick up an object and then to perform an assembly action. Finally, the speaker may return to the initial gear and only make requests for assembly actions.

Severinsson (1983) views to the process of downshifting as making *latent* subgames into *explicit* subgames. In the case mentioned above, the goals of the latent subgames are (1) to get the hearer to identify a referent, and (2) for the hearer to pick up the object referred to. In high gear, these subgames are latent in the sense that they do not give rise to any utterances (dialogue moves). When the latent subgames become explicit, the process that was previously carried out silently is instead carried out using utterances.

This view fits well with the concept of tacit moves introduced in Section 4.4.2. Updates for latent referent identification and utterance acceptance can be regarded as tacit moves (or games) corresponding to explicit referent identification or negotiation subdialogues, similar to the way that question accommodation updates are tacit moves corresponding to the ask dialogue moves.

Both these notions, shifting gears in dialogue and latent subgames, are useful for shedding light on the relation between negotiative dialogue and utterance acceptance. Firstly, the notions of optimism and pessimism regarding grounding strategies seem intimately related to the notion of gears, both metaphorically and factually. Metaphorically, we may say that an optimistic driver will use a higher gear than a pessimistic one; only when she encounters a bumpy road will she shift into lower gear (thus taking a more pessimistic approach). Later, when the road becomes smoother, she may again resume her optimistic strategy and use a higher gear. Similarly, speakers can be expected to switch between higher and lower gears, and between optimistic and pessimistic grounding strategies regarding the grounding of their utterances. Thus we claim that the notion of shifting gears is applicable not only to referent identification, but also to other grounding related games, including utterance acceptance.

In Chapter 3, we talked about optimism and pessimism in regard to grounding on the acceptance level; we now add that DPs may shift gears regarding grounding on the acceptance level. In a dialogue in high gear, the speaker optimistically assumes the hearer to accept her utterances. However, should the speaker reject some utterance, the dialogue is downshifted and the latent uptake subgame becomes explicit. We would claim (contrary to Sidner) that it is only when the dialogue is downshifted in this sense that moves such as questions and assertions should be regarded as proposals. At this stage, DPs may introduce alternatives to the proposal, and they may argue for or against proposals.

The concept of downshift is related to Ginzburg's case where a proposition p is rejected as a fact but $?p$ is accepted as a question for discussion. This appears to be a potential case of downshifting which could be modelled by regarding $\mathbf{?}p\mathbf{\bullet\{yes,\ no\}}$ as an issue under negotiation. In addition, alterations of p may be proposed, roughly corresponding to Clark's "cooperative alterations". It appears this can be modelled as an issue under negotiation $\mathbf{?}x\mathbf{.}p_x \bullet \{a, b, \ldots\}$ (where p_x is the proposition p with some argument a replaced by x, and thus $p = p_x(a)$). The alterations are then represented as alternatives $b, \ldots$ to a.

Thus, if a question q has been raised in a dialogue and if an answer a relevant to q is rejected (on the grounding level), q may become negotiable (depending on the activity). If so, the DP who rejected a may propose an alternative answer a' to q. It is then possible for the DPs to start a (probably argumentative) negotiation regarding which of a and a', or perhaps some other answer, should be accepted as the answer to q. We thus believe that downshifting of dialogue from optimistic acceptance to negotiation can shed light on various grounding-related phenomena, e.g. alterations (see Section 3.2.1), and the relation between grounding and negotiation.

5.9 Summary

Firstly, we extended the issue-based approach to action-oriented dialogue, and implemented a dialogue interface to a VCR where dialogue plans were based on an existing menu interface. We modified the information state by adding a field /SHARED/ACTIONS, and also added two new dialogue moves specific to AOD request and confirm. We also implemented update rules in IBiS to handle integration and selection of these moves, as well as interaction with a device, and also provided an additional accommodation rule for actions.

Secondly, we proposed a view of negotiation as discussing several alternative solutions to an issue under negotiation. On our approach, an issue under negotiation is represented as a question, e.g. what flight the user wants. In general, this means viewing problems as issues and solutions as answers. This approach has several advantages. Firstly, it provides a straightforward an intuitively sound way of capturing the idea that negotiative dialogue involves several alternative solutions to some issue or problem, and that proposals introduce such alternatives. Secondly, it distinguishes two types of negotiation (grounding-related negotiation and negotiation of issues) and clarifies the relation between them.

Chapter 6

Conclusions and future research

6.1 Introduction

In this final chapter, we first summarize the previous chapters. We will then use the results to classify various dialogue types and activities, and say something about the relation of the issue-based model to Grosz and Sidner's (1986) account of dialogue structure. Finally, we discuss future research issues.

6.2 Summary

In Chapter 1, we presented the aim of this study and gave some initial motivations for exploring the issue-based approach to dialogue management. We then gave a brief overview of the thesis and the related versions of the IBiS system. Finally, we gave a very brief introduction to the TrindiKit architecture and the information state approach to dialogue implemented therein.

In Chapter 2, we laid the groundwork for further explorations of issue-based dialogue management and its implementation in the IBiS system. As a starting point we used Ginzburg's concept of Questions Under Discussion, and we explored the use of QUD as the basis for the dialogue management (Dialogue Move Engine) component of a dialogue system. The basic uses of QUD is to model raising and addressing issues in dialogue, including the resolution of elliptical answers. Also, dialogue plans and a simple semantics were introduced and implemented.

In Chapter 3 we discussed general types and features of feedback as it appears in human-human dialogue. Next, we discussed the concept of grounding from an information state update point of view, and introduced the concepts of optimistic, cautious and pessimistic grounding strategies. We then related grounding and feedback to dialogue systems, and discussed the implementation of a partial-coverage model of feedback related to grounding in IBiS2. This allows the system to produce and respond to feedback concerning issues dealing with the grounding of utterances.

In Chapter 4, we made a distinction between a local and a global QUD (referring to the latter as "open issues", or just "issues"). The notions of question and issue accommodation were then introduced to allow the system to be more flexible in the way utterances are interpreted relative to the dialogue context. Question accommodation allows the system to understand answers addressing issues which have not yet been raised. In cases of ambiguity, where an answer matches several possible questions, clarification dialogues may be needed.

In Chapter 5, we first extended the issue-based approach to action-oriented dialogue, and implemented a dialogue interface to a VCR where dialogue plans were based on an existing menu interface. We then proposed a view of negotiation as discussing several alternative solutions to an issue under negotiation. On our approach, an issue under negotiation is represented as a question, e.g. what flight the user wants. In general, this means viewing problems as issues and solutions as answers.

6.3 Dialogue typology

In this section, we will use some distinctions made in previous chapters as a basis for classifying dialogues and dialogue segments along various dimensions. While these dimensions can to some extent be used to classify dialogue systems according to the kinds of dialogues they can handle, they are not intended as a classification of human-human dialogues. Rather, they should be regarded as describing properties of dialogue segments.

As we have previously stated, we make a distinction between Inquiry-oriented and Action-oriented dialogue according to whether the dialogue concerns non-communicative actions to be performed by a DP. Usually, but not necessarily, AOD subsumes IOD. One example of "pure" action oriented dialogue, where no questions are asked, is Wittgenstein's simple "slab" game in Wittgenstein (1953). Another example is simple voice command systems. AOD and IOD are shown with their corresponding dialogue moves and information state components in Table 6.1.

We can also classify dialogues according to the presence or absence of general dialogue features such as grounding, question accommodation, and negotiation. This is done in

Dialogue type	Moves	IS components
IOD	ask answer	QUD ISSUES
AOD	request confirm	ACTIONS

Table 6.1: Dialogue types

Table 6.2. While grounding and accommodation is probably present in all human-human dialogue, negotiation may be less frequent.

Feature	Moves	IS component
Grounding	icm	TMP, grounding issues
Accommodation	accommodate$X$ (tacit)	-
Negotiation	propose	Question•Set(Answer)

Table 6.2: Dialogue features

Finally, we can also classify activities according to various aspects of dialogue, as in Table 6.3. Note that this classification is independent of that in Table 6.2. We believe that dialogue in all these activities may be negotiative or non-negotiative, and negotiation may be argumentative or non-argumentative.

Activity type	Dialogue type	Result type	External process	Decision rights
Database search	IOD	simple: price etc. complex: itinerary	passive	user
Ticket booking	AOD	simple: book ticket	passive	user
Simple device control	AOD	simple: action	passive or active	user shared
Offline planning, incl. itinerary planning, complex device control	AOD	complex: plan	passive	shared
Online planning, incl. rescue planning (TRIPS)	AOD	complex: plan	active	shared
Explanation	IOD	complex: explanation	passive	shared
Tutorial	IOD or AOD	complex	?	tutor

Table 6.3: Activities

We will now relate the taxonomy above to the taxonomies in Dahlbäck (1997) and Allen *et al.* (2001). It should be stressed that neither Allen not Dahlbäck have the same goals with their classifications as we do here, and though some formulations may appear critical they are mainly intended to clarify the relation between these classifications and ours.

6.3.1 Relation to Dahlbäck's dialogue taxonomy

Dahlbäck (1997) taxonomizes dialogue according to seven criteria:

- modality: spoken or written
- kinds of agents: human or computer
- interaction: dialogue or monologue
- context: spatial, temporal
- number and type of possible/simultaneous tasks
- dialogue-task distance: long or short
- kinds of shared knowledge used: perceptual, linguistic, cultural

Our typology appears to be on a different level and is independent of many of Dahlbäck's criteria, and both cover important (but for the most part distinct) dimensions of classification. In general, the interaction between the dimensions covered by Dahlbäck and the ones covered in our typology is an interesting area for future research.

Modality is not included in our typology; however, IBiS is able to use both written and spoken language. Regarding kinds of agents, we have of course been dealing mainly with human-computer interaction; however, we have based both theory and implementation on observations of human-human dialogue.

Our dialogue typology should be regarded as primarily concerning dialogue interaction; however, a version of GoDiS (the predecessor of IBiS) has been used to produce monologue output from a domain plan specification which was also used for generating dialogue plans (see Larsson and Zaenen, 2000).

We have not included aspects of spatial and temporal context in our typology; for our theory and system we have not explored the impact of any other kind of context than (pre-stored information about) the domain (activity) and the dialogue itself.

Regarding the number and type of possible and/or simultaneous tasks, the use of the ISSUES and ACTIONS stacks allows, at least in principle, an arbitrary number of simultaneous tasks. Since the simplest version of our theory and system can handle this, we have not used this as a dimension of classification.

The dialogue-task distance dimension is perhaps less obvious than the others. This is based on the observation that some kinds of dialogue have a structure closely corresponding to the task structure (e.g. planning or advisory dialogue), while some have a "longer distance" between these two structures (e.g. information retrieval dialogue). Dahlbäck argues that for dialogues with a short dialogue-task distance, intention-based methods for dialogue act recognition is both more useful and easier than for dialogues with a long dialogue-task distance. For the latter, surface-based act interpretation is easier and more appropriate, whereas intention-based methods are less useful and more difficult. Regarding this dimension, we have been mostly concerned with dialogues with a long dialogue-task distance, and if Dahlbäck is right an intention-based and context-dependent interpretation module will be needed when extending the issue-based approach to e.g. collaborative planning dialogue. While this may affect how dialogue moves are defined, we believe (although we cannot be sure) that the set of dialogue moves we have proposed in our taxonomy can still be maintained.

Finally, regarding the kinds of shared knowledge that are used, our taxonomy does not say much. We have not been concerned with the perceptual and cultural context, except to the extent that these are encoded in the static domain knowledge resources. The use of domain-specific lexicons can perhaps be regarded as a simplistic form of linguistic context.

6.3.2 Relation to Allen et. al.'s dialogue classification

The classification by Allen *et al.* (2001) appears to be closer in spirit to the one proposed here. Dialogues are classified according to the dialogue management technique (minimally) required by a dialogue system capable of handling the respective kinds of dialogue. Each class is further specified by example tasks, a degree of task complexity (ranging from least to most complex), and a set of dialogue phenomena handled.

- finite-state script
  - example task: long-distance calling
  - dialogue phenomena: user answers questions
- frame-based
  - example tasks: getting train timetable information
  - dialogue phenomena: user answers questions, simple clarifications by system
- sets of contexts
  - example tasks: travel booking agent

  - dialogue phenomena: shifts between predetermined topics

- plan-based models
  - example tasks: kitchen design consultant
  - dialogue phenomena: dynamically generated topic structures, collaborative negotiation subdialogues

- agent-based models
  - example tasks: disaster relief management
  - dialogue phenomena: different modalities (e.g. planned world and actual world)

The first thing to note about this classification is that it does not distinguish separate dimensions of classification, but rather reduce several dimensions to one; this kind of simplification and generalization does of course have its merits, but may also be confusing.

Regarding the taxonomy of technologies used in this classification, it appears that the closest corresponding dimensions in our typology is the different kinds of information states and dialogue moves used for various dialogue types, dialogue phenomena, and activities. However, the classifications are also quite different; for one thing, the finite-state-based and form-based techniques usually do not even use dialogue moves. By contrast, our classification relies on specifying dialogue moves even for very simple dialogues. We will not go into a discussion of the relative merits of these grounds of classification; suffice to say that a theory-dependent classification (which ours to some extent is) allows a greater level of detail in the classification, but its usefulness is of course dependent on the acceptance of the basic theoretical assumptions that are made.

The first two technologies listed by Allen et. al. were discussed in Chapter 1, and the distinction between them are pretty much standard. However, the classification of the remaining three technologies is more problematic.

Regarding the "sets of contexts" technology, further specified as the use of several forms, it can be regarded as ambiguous between the use of several forms of the same type and the use of several forms of different types. The example task provided is "travel booking agent", or more specifically, itinerary booking. This seems to indicate that the intended meaning of "sets of contexts" is the use of several forms of the same type (e.g. one for each leg of the itinerary). In our typology of activities in Table 6.3, this would correspond to a dialogue with a complex result. However, the use of several forms of different types seems rather to the possibility of several simultaneous tasks (e.g. asking about which channel is on while programming the VCR).

The level of plan-based technology is further specified as "interactively constructing a plan with the user" (Allen *et al.*, 2001, p.30). This specification thus says something about the result of the dialogue (a plan) and how this result is constructed (interactively). Note that this is not exactly what we referred to as the plan-based approach in Chapter 1; at least in principle (and perhaps also in practice; this is an empirical issue related to Dahlbäck's concept of dialogue-task distance) it appears to be possible for a dialogue system to engage in this kind of dialogue even if the system itself does not use complex planning and plan recognition (e.g. for dialogue act recognition). Relating this to our classification of activities, it appears that the plan-based level corresponds roughly to dialogues with complex results (plans) and distributed decision rights (interactivity). As is indicated by Table 6.3, the techniques needed to handle the "plan-based" level would also be needed for e.g. explanatory and tutorial dialogue.

Finally, regarding the level of agent-based technology, further specified as possibly involving execution and monitoring of operations in a dynamically changing world, it appears that the main difference to the plan-based model is what we refer to as (pro)activeness of the external process.

To conclude, it appears from the point of view of our typology that the classification by Allen *et al.* (2001) is based on a mix of criteria, including information state components (e.g. forms) but also activity type, result type, pro-activeness of external process, decision rights, and dialogue features such as grounding ("simple clarifications") and negotiation ("collaborative negotiation subdialogues"). The AOD/IOD distinction appears not to be included at all.

6.4 Dialogue structure

In this section we discuss the implications of issue-based dialogue management on the structure of dialogue. We discuss the dialogue model of Grosz and Sidner (1986), elaborated in Grosz and Sidner (1987), and relate it to the issue-based model. The authors present a theory of discourse structure based on three structural components:

- linguistic structure: utterances, phrases, clauses etc.
- intentional structure: intentions, related by dominance and satisfaction-precedence
- attentional state: salient objects, properties, relations and discourse intentions

The intentional structure is related to dialogue structure through *Discourse Segment Purposes (DSPs).* A dialogue can be divided into segments where each segment is engaged in for the purpose of satisfying a particular intention, designated as the DSP of that segment. This relation is used to explain the close correspondence between task structure and dialogue structure observed in collaborative planning dialogue. With regard to dialogue management, it is claimed that "a conversational participant needs to recognize the DSPs and the dominance relationships between them in order to process subsequent utterances of the discourse" (Grosz and Sidner, 1987, p. 418). The authors also sketch a process model based on the concept of a *SharedPlan*, formalized in terms of individual intentions and mutual beliefs.

There are some interesting but rough correspondences between this model and the issue-based model, and the latter can perhaps be seen (at least to some extent) as an alternative (or complement) to the SharedPlans formalization.

The simplest correspondence is that between the linguistic structure and the LU field (and perhaps also the INPUT variable) in the issue-based model, although our model of linguistic structure is extremely impoverished.

In the issue-based model, DSPs roughly correspond to the ISSUES and (in AOD) ACTIONS fields, and should thus be useful for segmenting dialogue in a manner similar to Grosz and Sidner's. Sequencing ICM, which (among other things) reflect changes in ISSUES can thus be regarded as indicating dialogue segment shifts.

The local focus of attention is partially modelled by QUD, although so far our attentional model lacks e.g. a representation of "objects under discussion". Discourse intentions seem to correspond roughly to the AGENDA field, and possibly also the PLAN field although the latter is more global in nature. Of course, our representation of dialogue plans is quite different from that of Grosz and Sidner, who use a modal logic with operators for intentions.

It should be noted that the intentional structure, modelled as SharedPlans, is part of the shared knowledge. Grosz and Sidner are primarily interested in dialogues aimed at the collaborative creation and execution of these SharedPlans, which means that their model does not trivially extend to other kinds of dialogue, e.g. simple inquiry-oriented dialogue or tutorial dialogue. For the kinds of dialogue we have dealt with so far, the kind of complex representations needed for modelling SharedPlans appear not to be needed. The closest correspondence to SharedPlans in our model is the ACTIONS field which contains domain actions to be performed by one of the DPs. It can be expected that when the issue-based model is extended to handle collaborative planning dialogue, the structure of the ACTIONS field will become more complex and more similar to SharedPlans.

6.5 Future research areas

In this section, we briefly mention some future areas for research using the issue-based approach to dialogue management. We also mention some desirable improvements to IBiS.

6.5.1 Developing the issue-based approach to grounding

Representation of utterances Starting from Ginzburg's grounding protocols, we have formalized and implemented a basic version of issue-based grounding and feedback. However, some aspects of the current solution are not completely satisfactory, and it appears that a better solution could be obtained by explicitly representing utterances in various stages of grounding to a larger extent than in the current system.

Grounding issues Also, to increase the coverage of the theory and the abilities of the system it would be useful to represent grounding issues explicitly on several levels of grounding to a larger extent than is currently done. Some of the possible grounding issues that could be represented are the following (S is the speaker, A is the addressee, u is an utterance by S).

- Contact level
  - S and A: Do I have contact with other DP?
- Perception level
  - S: Did A perceive u correctly? If not, what did A perceive?
  - A: What did S say? Did S say X? Which of $X_1, \ldots, X_n$ did S say?
  - S and A: Is u grounded on the perception level?
- Semantic understanding level
  - S: Did A understand the literal meaning of u? If not, what does A think I meant (literally)?
  - A: What does u mean (literally)? Does u mean X? Which of $X_1, \ldots, X_n$ does u mean?
  - S and A: Is u grounded on the semantic understanding level?
- Pragmatic understanding level

  - S: Did A understand the pragmatic meaning of u? If not, what does A think I meant (pragmatically)?
  - A: What did S mean by u mean, given the current context? How is u relevant in the current context? Did S mean X? Which of $X_1, \ldots, X_n$ did S mean?
  - S and A: Is u grounded on the pragmatic understanding level?

- Reaction level
  - S: Will A accept (the content of) u?
  - A: Should I accept (the content of) u? If u is an assertion, should I accept u as a fact or only as a topic for discussion? If I don't accept u, how should I indicate this? Should I accept an altered version of u? Should I accept only a part of u?

Increased coverage Our account of grounding and ICM is so far only partial in coverage; phenomena that remain to be accounted for and/or implemented include clarification ellipsis, semantic ambiguity resolution, collaborative completions and repair, and turntaking ICM. While we have included some rudimentary sequencing ICM, further investigations of the appropriateness and usefulness of these utterances are needed; here, research on discourse markers (e.g. Schiffrin, 1987) and cue phrases (e.g. Grosz and Sidner, 1986, Polanyi and Scha, 1983, and Reichman-Adar, 1984) can be of great use. We also want to explore turntaking in asynchronous dialogue management, and how this relates to turntaking ICM.

Own Communication Management has so far not been handled at all, and this is clearly an area where the system could be improved both on the interpretation and generation side. We hope that the issue-based approach could help clarify the relation between ICM and OCM aspects of grounding-related utterances.

Methods for choosing grounding and ICM strategies We have used a very basic method for choosing grounding and ICM strategies; this could be developed to include context-related aspects of the utterance to be grounded. This also goes for the strategies for choosing between several competing interpretation hypotheses on the perception and understanding levels.

Implicit issues We believe that the modelling of implicit issues, both grounding-related and others, can be very useful for accounting for the relevance of many utterances. We therefore need to develop a general way of dealing with implicit questions and the accommodation of these. We believe that such an account should be based on dynamic generation and accommodation of implicit issues when they are needed, rather than calculating all possible implicit issues available at any stage of the dialogue.

6.5.2 Other dialogue and activity types

Of course, an obvious continuation of the work presented here is to continue extending the coverage of issue-based dialogue management to other kinds of dialogues. In this section we will discuss some possibilities.

Pro-active devices To handle dialogue with pro-active devices, it is not sufficient to model the device only as a resource, since the latter are by definition passive. What is needed is a module which is connected to the active device; we can call such a module an *action manager*. Dialogue with pro-active devices will also require asynchronous dialogue processing, at least on some level. The simplest solution is to check for messages from the device when the system has the turn. To handle this, it is probably sufficient to have the whole system except the action manager running as a single process. However, it may also be necessary for the system to interrupt the user (or indeed itself) in mid-turn, to give a report on the state of some action or plan being executed. This is likely to require a more advanced asynchronous setup, where several processes are needed.

Complex results We have so far only been dealing with dialogues where the "result" (answer, action) is not very complex. In e.g. itinerary information dialogue, the result may be a more complex structure. In collaborative planning dialogue the result is a potentially complex plan with several actions related in various ways. Similarly, explanatory dialogue may require representation of complex explanations or proofs. To deal with dialogue with complex results, we need to be able to represent these complex structures, and perhaps also to incrementally construct them by successive additions and modifications. However, we believe that the essential features of inquiry-oriented, action-oriented, and negotiative dialogue are the same regardless of whether the results are complex or simple.

Argumentation To handle argumentation, which is most likely to appear in negotiative dialogue, we hope to be able to exploit previous work in this area, e.g. Mann and Thompson (1983) and Asher and Lascarides (1998), and relate it to the issue-based approach.

Use of obligations Traum and Allen (1994) propose *obligations* as a central social attitude driving dialogue. For example, if DP A asks a question Q to DP B, B will have an obligation to address Q; typically, this obligation will then give rise to an intention to address Q. In IBiS, we instead add Q to QUD (global and local), and if the system can answer a question on QUD it will do so. It has been noted that the job done by obligations and QUD overlap to a large extent (Kreutel and Matheson, 1999), and in many kinds of dialogue the choice between QUD and obligations will not result in any differences in behaviour. However, there are also differences between QUD and obligations.

For one thing, QUD does not represent who raised the question, or who should respond to it. An interesting question then becomes: given that we include a global QUD in our information state, when does it become necessary to also include obligations? It appears that one type of dialogue where QUD on its own is insufficient is tutorial dialogue, where the tutor asks the student a so-called "exam question" to which the tutor already knows the correct answer. Given the strategy used by IBiS, the system would raise the question and then immediately answer it, which is probably not a very good teaching strategy. However, in many other kinds of dialogues it appears that the strategy of answering a question regardless of who answered it is a useful strategy. For example, if a DP A (a human or perhaps a robot equipped with vision) asks another DP B where some object is located and then finds the object, it appears more felicitous for A to answer the question ("Ah, there it is!") than to wait for B to do so. More importantly, we have in the preceding chapters demonstrated several uses of a global QUD (modelling grounding issues, handling issue accommodation, representing issues under negotiation) which it may or may not be possible to handle using obligations. For these reasons, we are interested in further exploring the similarities, differences, and interaction between QUD and obligations, and possibly extend the issue-based dialogue model by adding obligations, at least for modelling some complex kinds of dialogue.

General planning A similar case applies to generalized planning. We have so far only used pre-scripted dialogue plans which are used in a flexible way by the dialogue manager to enable some degree of rudimentary replanning, but it can be expected that for sufficiently complex dialogues the number of dialogue plans that are needed will become so large that pre-scripting is no longer feasible. At this point, dynamic planning will be needed. We are interested in finding out for which kinds of dialogues dynamic general-purpose planning is needed, and in integrating dynamic planning in the issue-based approach to dialogue management.

6.5.3 Semantics

The semantics currently used in IBiS is obviously very simple, and integrating the issue-based approach to dialogue management with a more powerful semantics is likely to improve both the theoretical depth of analysis, especially regarding the semantics of questions, and the performance of the system. A relevant issue in this context is the connection between dialogue features and requirements on the semantic representation used - when does more complex semantics become necessary?

Specifically, we would like to explore and implement semantics using dependent record types (Cooper, 1998), and integrate this with issue-based dialogue management. One reason for this is that dependent record types appears to provide a computationally sound framework for implementing ideas from situation semantics; the latter has been used by Ginzburg in formulating the semantics of questions on which the issue-based approach to dialogue management is based.

It should be noted that the system itself is independent of which semantics is used; this is rather a feature of the domain-specific resources and (to some extent) the resource interfaces. Adding a more powerful semantics will therefore not require any significant modifications of update rules etc.

6.5.4 General inference

While update rules can be regarded as specialized (forward-chaining) inference rules, we have so far not dealt with general inference and backward-chaining inference. Inference could be useful even in database search dialogue to reason about the best way of dealing with search results in the form of conditionals (see Section 2.12.4). One idea here is to introduce a private issue-structure representing questions that the system is interested in resolving; this could be regarded as modelling the "wonder" attitude. A findout(Q) action on the agenda could then result in Q being added to a field /PRIVATE/WONDER, which can either lead to a database search, backward-chaining reasoning from available information, or asking the user for an answer. As an example of backward-chaining reasoning using the "wonder" attitude, if the system believes $p \rightarrow r$ and wonders about $?r$, a rule could add $?p$ to the WONDER field; this rule would then implement backward-chaining modus ponens.

6.5.5 Natural language input and output

Parsing and generation In IBiS we have so far concentrated on dialogue management and used very rudimentary modules for interpretation and generation of natural language. We need to explore the possible use of robust parsing techniques (see e.g. Milward, 2000) and "real" grammars. It would also be useful to be able to automatically generate speech recognition grammars (which are usually finite-state) from the parsing grammar. Using the same grammar for parsing and generation would further decrease the amount of work needed for porting the system to a new domain or language.

Dialogue move interpretation Since we have been dealing with simple dialogue in toy domains, we have so far been able to get away with doing dialogue move interpretation independently of the dynamic context. Instead, context dependent interpretation is performed by the dialogue move engine as a subtask of integrating moves. While we believe that this is a good strategy to use as long as it works well, it may eventually become necessary to be able to recognize indirect speech acts (in our case, indirect dialogue moves), which probably requires some form of context-dependent intention recognition to decide what move has been performed.

Focus intonation One area of research that we have not mentioned so far, but where the issue-based approach shows great promise, is the generation and interpretation of focus intonation with respect to the information state. Some work was done on this in the TRINDI project (Engdahl *et al.*, 1999), and is currently being developed further in the followup SIRIDUS project.

Speech recognition for flexible dialogue One problem for any dialogue system allowing for user initiative and flexibility is that a larger speech recognition lexicon is needed, which negatively affects the quality of speech recognition. We want to explore the use of the information state, and especially QUD, for improving recognition, e.g. by running a "global" recognizer listening for anything that the system can understand, and a "local" recognizer, listening e.g. for answers to questions on QUD, in parallel.

6.5.6 Applications and evaluation

To properly test the issue-based approach to dialogue management, we believe it is necessary to build full-scale prototype applications and evaluate these based on interactions with naive users. One possible such application is VCR control; another is local travel information.

Although we have not said much about it here, we have previously explored various ways of acquiring dialogue plans appropriate for a given domain or application. Among the options we have used is dialogue distillation (see Larsson *et al.*, 2000b), conversion of domain plans to dialogue plans (see Larsson, 2000), and conversion of menu interfaces to dialogue plans.

A further option we want to explore is the use of VoiceXML (McGlashan *et al.*, 2001) dialogue specifications as a basis for dialogue plans. We hope to be able to automatically or semi-automatically convert VoiceXML scripts into complete domain and lexicon specifications for IBiS, which we hope would allow the use of general dialogue mechanisms (e.g. grounding, accommodation, negotiation) to enable flexible dialogue given fairly simple VoiceXML scripts. This would decrease the amount of work on the part of the dialogue designer, and thus enable rapid prototyping.

6.6 Conclusion

The issue-based approach to dialogue management has proven to be very useful for formulating general and theoretically motivated accounts of important aspects of dialogue, such as inquiry-oriented dialogue interactions, dealing with multiple simultaneous tasks, sharing information between tasks, grounding, interactive communication management, question accommodation, simple belief revision, action-oriented dialogue, and simple negotiative dialogue. The model can be implemented rather straightforwardly using the TrindiKit, which has proven to be a very useful tool for exploring the issue-based approach. Some aspects of the account as presented here can be improved on, e.g. by properly dividing the tasks of utterance understanding and integration into separate rules, and improving the treatment of semantics.

To really show that the issue-based approach is a viable alternative to more complex approaches such as the plan-based approach as used e.g. in the TRIPS system (Allen *et al.* (2001)), we need to extend the coverage of the issue-based account to include more complex types of dialogue, involving e.g. collaborative planning and real-time monitoring of a dynamic environment. We believe this is feasible, and an exciting future research area.

The modularity of the IBiS system enables rapid prototyping of simple experimental applications. We believe that it will be possible to scale up the methods presented here to more realistic applications which can be evaluated on naive subjects.

Bibliography

Alexandersson, Jan and Becker, Tilman 2000. Overlay as the basic operation for discourse processing in a multimodal dialogue system. In *Proceedings of the IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems.* 8–14.

Allen, J. F. and Perrault, C. 1980. Analyzing intention in utterances. *Artificial Intelligence* 15(3):143–178.

Allen, James F.; Byron, Donna K.; Dzikovska, Myroslava; Ferguson, George; Galescu, Lucian; and Stent, Amanda 2001. Toward conversational human-computer interaction. *AI Magazine* 22(4):27–37.

Allen, J. F. 1987. *Natural Language Understanding.* Benjamin Cummings, Menlo Park, CA.

Allwood, Jens; Nivre, Joakim; and Ahlsen, Elisabeth 1992. On the semantics and pragmatics of linguistic feedback. *Journal of Semantics* 9:1–26.

Allwood, Jens 1995. An activity based approach to pragmatics. Technical Report (GPTL) 75, Gothenburg Papers in Theoretical Linguistics, University of Göteborg.

Asher, N. and Lascarides, A. 1998. The semantics and pragmatics of presupposition. *Journal of Semantics* 15(3):239–299.

Aust, H.; Oerder, M.; Seide, F.; and Steinbiss, V. 1994. Experience with the Philips automatic train table information system. In *Proc. of the 2nd Workshop on Interactive Voice Technology for Telecommunications Applications (IVTTA)*, Kyoto, Japan. 67–72.

Barwise, J. and Perry, J. 1983. *Situations and Attitudes.* The MIT Press.

Berman, Alexander 2001. Asynchronous feedback and turn-taking. ms.

Bohlin, Peter; Bos, Johan; Larsson, Staffan; Lewin, Ian; Matheson, Colin; and Milward, David 1999. Survey of existing interactive systems. Technical Report Deliverable D1.3, Trindi.

Bos, Johan and Gabsdil, Malte 2000. First-order inference and the interpretation of questions and answers. In Poesio and Traum 2000.

Boye, J.; Larsson, S.; Lewin, I; Matheson, C.; Thomas, J.; and Bos, J. 2001. Standards in home automation and language processing. Technical Report Deliverable D1.1, D'Homme.

Bäuerle, Rainer; Reyle, Uwe; and Zimmermann, Thomas Ede, editors 2002. *Presuppositions and Discourse.* Current Research in the Semantics/Pragmatics Interface. Amsterdam (Elsevier).

Carberry, S. 1990. *Plan Recognition in Natural Language Dialogue.* The MIT Press, Cambridge, MA.

Chu-Carroll, Jennifer 2000. Mimic: An adaptive mixed initiative spoken dialogue system for information queries. In *Proceedings of the 6th Conference on Applied Natural Language Processing.* 97–104.

Clark, H. H. and Schaefer, E. F. 1989a. Contributing to discourse. *Cognitive Science* 13:259 – 94.

Clark, Herbert H. and Schaefer, Edward F. 1989b. Contributing to discourse. *Cognitive Science* 13:259–294. Also appears as Chapter 5 in Clark (1992).

Clark, Herbert H. 1992. *Arenas of Language Use.* University of Chicago Press.

Clark, H. H. 1996. *Using Language.* Cambridge University Press, Cambridge.

Cohen, P. and Levesque, H. 1990. Intention is choice with commitment. *Artificial Intelligence* 42:213–261.

Cohen, P. 1981. The need for referent identification as a planned action. In *Proceedings of the 7th International Joint Conference of Artificial Intelligence, Toronto.* 31–36.

Cooper, Robin and Ginzburg, Jonathan 2001. Resolving ellipsis in clarification. In *Proceedings of the 39th meeting of the Assocation for Computational Linguistics, Toulouse.* 236–243.

Cooper, Robin and Larsson, Staffan 2002. Accommodation and reaccommodation in dialogue. In Bäuerle et al. 2002.

Cooper, Robin; Engdahl, Elisabet; Larsson, Staffan; and Ericsson, Stina 2000. Accommodating questions and the nature of qud. In Poesio and Traum 2000. 57–61.

Cooper, Robin; Ericsson, Stina; Larsson, Staffan; and Lewin, Ian 2001. An information state update approach to collaborative negotiation. In Kühnlein, Peter; Rieser, Hannes; and Zeevat, Henk, editors 2001, *BI-DIALOG 2001—Proceedings of the 5th Workshop on Formal Semantics and Pragmatics of Dialogue,* `http://www.uni-bielefeld.de/BIDIALOG`. ZiF, Univ. Bielefeld. 270–9.

Cooper, R. 1998. Information states, attitudes and dependent record types. In *Proceedings of ITALLC-98.*

Core, Mark G. and Allen, James F. 1997. Coding dialogues with the DAMSL annotation scheme. In Traum, David, editor 1997, *Working Notes: AAAI Fall Symposium on Communicative Action in Humans and Machines*, Menlo Park, California. American Association for Artificial Intelligence. 28–35.

Dahlbäck, Nils 1997. Towards a dialogue taxonomy. In Maier, Elisabeth; Mast, Marion; and LuperFoy, Susann, editors 1997, *Dialogue Processing in Spoken Language Systems*, number 1236 in Springer Verlag Series LNAI-Lecture Notes in Artificial Intelligence. Springer Verlag.

Di Eugenio, B.; Jordan, P.W.; Thomason, R.H.; and Moore, J.D. 1998. An empirical investigation of proposals in collaborative dialogues. In *Proceedings of ACL–COLING 98: 36th Annual Meeting of the Association of Computational Linguistics and 17th International Conference on Computational Linguistics.* 325–329.

The DISC consortium, 1999. Disc dialogue engineering model. Technical report, DISC, http://www.disc2.dk/slds/.

Engdahl, Elisabet; Larsson, Staffan; and Bos, Johan 1999. Focus-ground articulation and parallelism in a dynamic model of dialogue. Technical Report Deliverable D4.1, Trindi.

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

Gärdenfors, P. 1988. *Knowledge in Flux: Modeling the Dynamic of Epistemic States.* The MIT Press, Cambridge, MA.

Ginzburg, J. 1994. An update semantics for dialogue. In al, H. Buntet, editor 1994, *Proceedings of the International Workshop on Computational Semantics.* ITK. Tilburg. 111–120.

Ginzburg, J. 1996. Interrogatives: Questions, facts and dialogue. In *The Handbook of Contemporary Semantic Theory.* Blackwell, Oxford.

Ginzburg, J. 1997. Structural mismatch in dialogue. In Jaeger, G. and Benz, A, editors 1997, *Proceedings of MunDial 97*, Technical Report 97-106. Universitaet Muenchen Centrum fuer Informations- und Sprachverarbeitung, Muenchen. 59–80.

Ginzburg, J. forth. Questions and the semantics of dialogue. Forthcoming book, partly available from `http://www.dcs.kcl.ac.uk/staff/ginzburg/papers.html`.

Goffman, E. 1976. Replies and responses. *Language in Society* 5:257–313.

Grosz, B. J. and Sidner, C. L. 1986. Attention, intention, and the structure of discourse. *Computational Linguistics* 12(3):175–204.

Grosz, B. J. and Sidner, C. L. 1987. Plans for discourse. In *Symposium on Intentions and Plans in Communication and Discourse.*

Grosz, B. J. and Sidner, C. L. 1990. Plans for discourse. In Cohen, P. R.; Morgan, J.; and Pollack, M. E., editors 1990, *Intentions in Communication.* The MIT Press, Cambridge, MA. chapter 20, 417–444.

Hulstijn, J. 2000. *Dialogue Models for Inquiry and Transaction.* Ph.D. Dissertation, University of Twente.

Jennings, N. and Lesperance, Y, editors 2000. *Proceedings of the 6th International Workshop on Agent Theories, Architectures, and Languages (ATAL'1999)*, Springer Lecture Notes in AI 1757. Springer Verlag, Berlin.

Kaplan, D. 1979. Dthat. In Cole, P., editor 1979, *Syntax and Semantics v. 9, Pragmatics.* Academic Press, New York. 221–243.

Kreutel, Jorn and Matheson, Colin 1999. Modelling questions and assertions in dialogue using obligations. In Van Kuppevelt et al. 1999.

van Kuppevelt, Jan; van Leusen, Noor; van Rooy, Robert; and Zeevat, Henk, editors 1999. *Proceedings of Amstelogue'99 Workshop on the Semantics and Pragmatics of Dialogue.*

Larsson, Staffan and Traum, David 2000. Information state and dialogue management in the trindi dialogue move engine toolkit. *NLE Special Issue on Best Practice in Spoken Language Dialogue Systems Engineering* 323–340.

Larsson, Staffan and Zaenen, Annie 2000. Document transformations and information states. In *Proceeding of the 1st SigDial Workshop, Hong Kong.* ACL. 112–120.

Larsson, Staffan; Ljunglöf, Peter; Cooper, Robin; Engdahl, Elisabet; and Ericsson, Stina 2000a. Godis - an accommodating dialogue system. In *Proceedings of ANLP/NAACL-2000 Workshop on Conversational System.*

Larsson, Staffan; Santamarta, Lena; and Jönsson, Arne 2000b. Using the process of distilling dialogues to understand dialogue systems. In *Proceedings of 6th International Conference on Spoken Language Processing (ICSLP2000/INTERSPEECH2000), Volume III.* 374–377.

Larsson, Staffan 1998. Questions under discussion and dialogue moves. In *Proceedings of the Twente Workshop on Language Technology.* 163–171.

Larsson, Staffan 2000. From manual text to instructional dialogue: an information state approach. In Poesio and Traum 2000. 203–206.

Lewin, Ian; Cooper, Robin; Ericsson, Stina; and Rupp, C.J. 2000. Dialogue moves in negotiative dialogues. Project deliverable 1.2, SIRIDUS.

Lewin, I.; Larsson, S.; Ericsson, S.; and Thomas, J. 2001. The d'homme device selection. Technical Report Deliverable D5.1, D'Homme.

Lewis, D. K. 1979. Scorekeeping in a language game. *Journal of Philosophical Logic* 8:339–359.

Ljunglöf, Peter 2000. Formalizing the dialogue move engine. In *Proceedings of Götalog 2000 workshop on semantics and pragmatics of dialogue.* 207–210.

Mann, W. C. and Thompson, S. A. 1983. Relational propositions in discourse. Technical Report ISI/RR-83-115, USC, Information Sciences Institute.

McGlashan, S.; Burnett, D; Danielsen, P.; Ferrans, J.; Hunt, A.; Karam, G; Ladd, D.; Lucas, B.; Porter, B.; and Rehor, K. 2001. Voice extensible markup language (voicexml) version 2.0. Technical report, W3C. W3C Working Draft, 23 October 2001.

Microsoft, 2000. *Universal Plug and Play Device ArchitectureVersion 1.0.* URL: http://www.upnp.org/download/UPnPDA10_20000613.htm.

Milward, D. 2000. Distributing representation for robust interpretation of dialogue utterances. In *Proceedings of the 38th Annual Meeting of the Association of Computational Linguistics, ACL-2000.* 133–141.

Moore, Johanna D. 1994. *Participating in Explanatory Dialogues : Interpreting and Responding to Questions in Context.* Acl-Mit Press Series in Natural Language Processing. MIT Press.

Poesio, Massimo and Traum, David R. 1998. Towards an axiomatization of dialogue acts. In *Proceedings of Twendial'98, 13th Twente Workshop on Language Technology: Formal Semantics and Pragmatics of Dialogue.* 207–222.

Poesio, Massimo and Traum, David, editors 2000. *Proceedings of Götalog 2000*, number 00-5 in GPCL (Gothenburg Papers Computational Linguistics).

Polanyi, L. and Scha, R. 1983. On the recursive structure of discourse. In Ehlich, K. and Riemsdijk, H.van, editors 1983, *Connectedness in Sentence, Discourse and Text.* Tilburg University. 141–178.

Rao, A. S. and Georgeff, M. P. 1991. Modeling rational agents within a bdi-architecture. In Allen, James; Fikes, Richard; and Sandewall, Eric, editors 1991, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR-91)*, Cambridge, MA. 473–484.

Reichman-Adar, R. 1984. Extended man-machine interface. *Artificial Intelligence* 22(2):157–218.

Sadek, M. D. 1991. Dialogue acts are rational plans. In *Proceedings of the ESCA/ETR workshop on multi-modal dialogue.* 1–29.

van der Sandt, R. A. 1992. Presupposition projection as anaphora resolution. *Journal of Semantics* 9(4):333–377.

San-Segundo, Ruben; Montero, Juan M.; Guitierrez, Juana M.; Gallardo, Ascension; Romeral, Jose D.; and Pardo, Jose M. 2001. A telephone-based railway information system for spanish: Development of a methodology for spoken dialogue design. In *Proceedings of the 2nd SIGdial Workshop on Discourse and Dialogue.* 140–148.

Schiffrin, D. 1987. *Discourse Markers.* Cambridge University Press, Cambridge.

Severinson Eklundh, Kerstin 1983. The notion of language game – a natural unit of dialogue and discourse. Technical Report SIC 5, University of Linköping, Studies in Communication.

Sidner, Candace L. and Israel, David J. 1981. Recognizing intended meaning and speakers' plans. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vancouver, British Columbia. International Joint Committee on Artificial Intelligence. 203–208.

Sidner, Candace L. 1994a. An artificial discourse language for collaborative negotiation. In *Proceedings of the forteenth National Conference of the American Association for Artificial Intelligence (AAAI-94).* 814–819.

Sidner, Candace. L. 1994b. Negotiation in collaborative activity: A discourse analysis. *Knowledge-Based Systems* 7(4):265–267.

Stalnaker, R. 1979. Assertion. In Cole, P., editor 1979, *Syntax and Semantics*, volume 9. Academic Press. 315–332.

Stenström, Anna-Brita 1984. *Questions and Responses.* Lund Studies in English: Number 68. Lund : CWK Gleerup.

Sutton, S. and Kayser, E. 1996. The cslu rapid prototyper: Version 1.8. Technical report, Oregon Graduate Institute, CSLU.

Traum, D. R. and Allen, J. F. 1994. Discourse obligations in dialogue processing. In *Proc. of the 32nd Annual Meeting of the Association for Computational Linguistics*, New Mexico. 1–8.

Traum, D. R. and Hinkelman, E. A. 1992. Conversation acts in task-oriented spoken dialogue. *Computational Intelligence* 8(3):575–599. Special Issue on Non-literal Language.

Traum, D. R. 1994. *A Computational Theory of Grounding in Natural Language Conversation.* Ph.D. Dissertation, University of Rochester, Department of Computer Science, Rochester, NY.

Traum, David R. 1996. A reactive-deliberative model of dialogue agency. In Müller, J. P.; Wooldridge, M. J.; and Jennings, N. R., editors 1996, *Intelligent Agents III — Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL-96)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, Heidelberg. 151–157.

Wittgenstein, Ludwig 1953. *Philosophical Investigations.* Basil Blackwell Ltd.

Wooldridge, M. and Jennings, N. R. 1995. Intelligent agents: Theory and practice. *Knowledge Engineering Review* 10(2):115–152.

Appendix A

TrindiKit functionality

A.1 Introduction

In this appendix, we give a more detailed description of some parts of TrindiKit relevant to the implementation of the IBIS systems. This description refers to version 3.0a of TrindiKit[1].

Apart from the general architecture defined in 1.5, the TrindiKit provides, among other things,

- definitions of datatypes, for use in TIS variable definitions
- a format for defining datatypes
- methods (checks, queries and updates) for accessing the TIS
- a language for specifying TIS update rules
- an update algorithm language for modules
- a control algorithm language for the controller, including concurrent control
- simple default modules for input, interpretation, generation and output
- use of external resources

[1]The latest official version of TrindiKit is available from the TrindiKit webpage, `www.ling.gu.se/projekt/trindi`. Version 3.0a is available from `www.ling.gu.se/~sl/Thesis`.

We will first explain how datatypes are defined, and give specifications of some datatypes used by IBIS. We then show how these definitions relate to the syntax and semantics of conditions, queries and updates. We go on to show how conditions, queries and updates are used in formulating update rules. Two algorithm specification languages, for coordinating update rules and modules, respectively, are then introduced. Finally, we describe some of the modules included in the TrindiKit package.

A.2 Datatypes

Datatypes[2] are used extensively in the TrindiKit architecture, most importantly for modelling the TIS. Datatypes provide a natural way of formalizing information states. The TIS is specified using abstract data types, each permitting a specific set of queries to inspect the type and operations to change it.

The TrindiKit provides a number of datatype definitions, to which the user may add her own.

A.2.1 Datatype definition format

A datatype definition may include the following:

1. relations
2. functions
3. selectors
4. operations

Datatypes are implemented in the form of Prolog clauses, here represented in the form $HeadClause \leftarrow Body$ or simply $HeadClause$.

[2]An alternative term is "(abstract) data structure".

Relations

The arguments of a relation are objects, and the definition of the relation specifies between which objects the relation holds. For example, the relation in has two arguments, a set S of objects of type T and an object X of type T, and holds if X is a member of S.

The head clause of a relation definition has the form in (A.1).

(A.1) relation(Rel, $[Arg1, \ldots, Arg_n]$)

A sample definition of a relation is is shown in (A.2).

(A.2) relation(<, $[A, B]$) $\leftarrow A < B$

Relations may also be indirectly defined by functions and selectors, as will be explained below.

Functions

Functions take objects as arguments and return a new object. For example, the function fst takes a stack S of type stack(T) and gives the topmost element X (of type T) of the stack. If the stack is empty, the result of the function is undefined; that is, functions may be partial.

The head clause of a function definition has the form in (A.3).

(A.3) function(Fun, $[Arg1, \ldots, Arg_n]$, $Result$)

A sample definition of a relation is is shown in (A.4).

(A.4) function(arity, [set(Xs)], N) $\leftarrow$ length(Xs, N).

Every function corresponds to a relation according to the schema in (A.5).

(A.5) relation(Fun,$[Arg_1, \ldots, Arg_n, Result]$) $\leftarrow$
function(Fun, $[Arg1, \ldots, Arg_n]$, $Result$)

For example, given the function arity we also have a relation arity whose arguments is a set S and an integer N; this relation holds if the N is the result of applying arity to S, i.e. if N is the arity of S.

Selectors

Selectors can be regarded as a special kind of functions which can be applied to collections of objects (i.e. objects containing other objects, e.g. sets, stacks and records) to select objects in the collection.

The head clause of a function definition has the form in (A.6).

(A.6) selector(*Sel*, *Coll*, *Obj*, $CollWithHole$, $Hole$)

Here, *Coll* is a collection (e.g. a stack), *Obj* is the object in *Coll* selected by *Sel*. $CollWithHole$ is a copy of *Coll* with *Obj* replaced by a prolog variable $Hole$. The reason for this way of implementing selectors is rather technical and we will not be pursued here.

A sample definition of a selector is is shown in (A.7).

(A.7) selector(fst, stack([Fst | *Rest*]), Fst, stack([$Hole$ | *Rest*]), $Hole$
)

Every selector corresponds to a function according to the schema in (A.8).

(A.8) function(*Sel*, [*Coll*], *Obj*) $\leftarrow$
selector(*Sel*, *Coll*, *Obj*, $CollWithHole$, $Hole$)

For example, given the selector fst which selects an object *Obj* in a stack S, we also have a function fst whose argument is S and an whose result is *Obj*. The result of applying this function to S is *Obj* if *Obj* is the topmost element of S.

Since each selector corresponds to a function and each function corresponds to a relation, it follows that each selector corresponds to a relation. For example, given the selector fst which selects an object *Obj* in a stack S, we also have a relation fst which holds if fst selects *Obj* in S.

Operations

Operations take an input object and (optional) arguments and return an output object. The objects in TrindiKit are *immutable*, which means they cannot be changed. What operations do is to replace an object with another object which is the result of applying the operation to the original object.

The head clause of an operation definition has the form in (A.9).

(A.9) operation(Opr, Obj_{in}, [$Arg_1, \ldots, Arg_n$], Obj_{out})

A sample definition of an operation is is shown in (A.10).

(A.10) operation(push, stack(Xs), [X], stack([$Xl \mid Xs$]))

Every operation corresponds to a relation according to the schema in (A.11).

(A.11) relation(Opr, [Obj_{in}, $Arg_1, \ldots, Arg_n$, Obj_{out}]) $\leftarrow$
operation(Opr, Obj_{in}, [$Arg_1, \ldots, Arg_n$], Obj_{out})

For example, given the operation push which pushes an object X on a stack $Stack_{in}$, resulting in a stack $Stack_{out}$, we also have a relation push whose arguments are $Stack_{in}$, X, and $Stack_{out}$. This relation holds if $Stack_{out}$ is the result of pushing X on $Stack_{in}$.

Some datatypes used by IBiS

In this section, we list the definitions of some of the datatypes used in the implementation of IBiS. Relations, functions, selectors and operations are here represented as they appear in update rules, rather than how they appear in the datatype definitions. The relation between these representation is the following:

- $Rel(Arg_1, \ldots, Arg_n)$ given relation(Rel, [$Arg_1, \ldots, Arg_n$])
- $Fun(Arg_1, \ldots, Arg_n)$ given relation(Fun, [$Arg_1, \ldots, Arg_n$], $Result$)
- $Coll/Sel$ given selector(Sel, $Coll$, Obj, $CollWithHole$, $Hole$)
- $Opr(Arg_1, \ldots, Arg_n)$ given relation(Opr, Obj_{in}, [$Arg_1, \ldots, Arg_n$], Obj_{out})

Relations are described in terms of truth conditions, i.e. what has to be true for the relation to hold. For functions, there is a description of the result of applying the function to its arguments. For selectors, the selected object is described. For operations, the description explains how the operation modifies the object to which it is applied.

Some of the descriptions below are partial in the sense that not all relations, functions, selectors and operations are included. We have mainly included those used in IBiS.

Set

Simple unordered set.

TYPE: set

$$\textsc{rel}: \left\{ \text{in}(Set, X) : \textit{X is unifiable with an element of Set} \right.$$

$$\textsc{fun}: \left\{ \text{arity}(Set) : \textit{the number of elements in Set} \right.$$

$$\textsc{sel}: \left\{ Set/\text{elem} : \textit{an element (member) of Set} \right.$$

$$\textsc{opr}: \begin{cases} \text{add}(X) : \textit{adds an element X} \\ \text{del}(X) : \textit{deletes an element unifiable with X, fails if no element is unifiable with X} \\ \text{extend}(Set) : \textit{add all elements of Set} \end{cases}$$

Stack

Simple stack.

TYPE: stack

$$\textsc{rel}: \left\{ \text{fst}(Stack, X) : \textit{X is unifiable with the topmost element of Stack} \right.$$

$$\textsc{fun}: \left\{ \text{arity}(Stack) : \textit{the number of elements in Stack} \right.$$

$$\textsc{sel}: \left\{ Set/\text{fst} : \textit{the topmost element of Set} \right.$$

$$\textsc{opr}: \begin{cases} \text{push}(X) : \textit{make X the topmost element} \\ \text{pop} : \textit{pop the stack, i.e. remove the topmost element} \end{cases}$$

Open stack (“stackset”)

Stack with some set-like properties. Non-topmost elements can be accessed. Open stacks cannot contain two unifiable elements.

TYPE: openstack

$$\textsc{rel}: \begin{cases} \text{fst}(Stack, X) : \textit{X is unifiable with the topmost element of Stack} \\ \text{in}(Stack, X) : \textit{X is unifiable with an element of Set} \end{cases}$$

$$\textsc{fun}: \left\{ \text{arity}(Stack) : \textit{the number of elements in Stack} \right.$$

$$\textsc{sel}: \left\{ Set/\text{fst} : \textit{the topmost element of Set} \right.$$

$$\textsc{opr}: \begin{cases} \text{push}(X) : \begin{array}{l} \textit{if no element is unifiable with } X\textit{, make } X \textit{ topmost;} \\ \textit{if } X \textit{ is unifiable with an element } Y\textit{, make } Y \textit{ the topmost element} \end{array} \\ \text{raise}(X) : \begin{array}{l} \textit{make an element unifiable with } X \textit{ the topmost element;} \\ \textit{fails if } X \textit{ is not unifiable with any element} \end{array} \\ \text{pop} : \begin{array}{l} \textit{pop the stack, i.e. remove the topmost element;} \\ \textit{fails if the stack is empty} \end{array} \\ \text{del}(X) : \begin{array}{l} \textit{deletes an element unifiable with } X\textit{;} \\ \textit{fails if no element is unifiable with } X \end{array} \end{cases}$$

Queue

FIFO queue.

$\textsc{type}$: queue

$$\textsc{rel}: \begin{cases} \text{none}^3 \end{cases}$$

$$\textsc{fun}: \begin{cases} \text{arity}(\textit{Queue}) : \textit{the number of elements in Queue} \end{cases}$$

$$\textsc{sel}: \begin{cases} \textit{Queue}/\text{fst} : \textit{the first (closest to end top) element of Set} \\ \textit{Queue}/\text{lst} : \textit{the last (closest to the end) element of Set} \end{cases}$$

$$\textsc{opr}: \begin{cases} \text{push}(X) : \textit{make } X \textit{ the last element} \\ \text{pop} : \textit{pop the queue, i.e. remove the first element} \end{cases}$$

Open Queue

FIFO queue with some set-like properties and a “shift” operation.

$\textsc{type}$: openqueue

$$\textsc{rel}: \begin{cases} \text{fst}(\textit{Queue}, X) : X \textit{ is unifiable with the topmost element of Queue} \\ \text{in}(\textit{Queue}, X) : X \textit{ is unifiable with an element of Set} \\ \text{fully\_shifted}(\textit{Queue}) : \begin{array}{l} \textit{Queue has been shifted one cycle,} \\ \textit{i.e. all elements have been shifted once} \end{array} \end{cases}$$

$$\textsc{fun}: \begin{cases} \text{arity}(\textit{Queue}) : \textit{the number of elements in Queue} \end{cases}$$

$$\textsc{sel}: \begin{cases} \textit{Queue}/\text{fst} : \textit{the first (closest to end top) element of Set} \\ \textit{Queue}/\text{lst} : \textit{the last (closest to the end) element of Set} \end{cases}$$

OPR: $\left\{\begin{array}{l} \text{push}(X) : \begin{array}{l} \textit{if no element is unifiable with } X\textit{, make } X \textit{ the last element;} \\ \textit{if } X \textit{ is unifiable with an element } Y\textit{, make } Y \textit{ the last element} \end{array} \\ \text{pop} : \textit{pop the queue, i.e. remove the first element; fails if the stack is empty} \\ \text{del}(X) : \begin{array}{l} \textit{deletes an element unifiable with } X\textit{;} \\ \textit{fails if no element is unifiable with } X \end{array} \\ \text{shift} : \begin{array}{l} \textit{pop and push first element to the end;} \\ \textit{requires shifting enabled} \end{array} \\ \text{init\_shift} : \textit{initialize queue for shifting} \\ \text{cancel\_shift} : \textit{disable shifting} \end{array}\right.$

pair

Simple pair of objects, possibly of different types. We will sometimes use the notation *Fst-Snd* for pairs.

TYPE: pair
REL: $\{$ none
FUN: $\{$ none
SEL: $\left\{\begin{array}{l} \textit{Pair}/\text{fst} : \textit{the first element of Pair} \\ \textit{Pair}/\text{snd} : \textit{the second element of Pair} \end{array}\right.$
OPR: $\left\{\begin{array}{l} \text{set\_fst}(X) : \textit{set the first element to } X \\ \text{set\_snd}(X) : \textit{the second element to } X \end{array}\right.$

record

Recursive record structure.

TYPE: record
REL: $\{$ none
FUN: $\{$ none
SEL: $\{$ *Record/Label* : *the value of Label in Record*
OPR: $\{$ add_field(*Label*, *Obj*) : *add a field with label Label and value Obj*

Type independent

Apart from the type-specific relations, functions, selectors and operations, there are some that apply to objects of all types.

$$\begin{array}{l}
\textsc{type:}\ \text{(any type)}\\
\textsc{rel:}\ \begin{cases}\text{is\_set} : \textit{the object is set (it is not nil)}\\ \text{is\_unset} : \textit{the object is not set (it is nil)}\\ \text{is\_empty} : \textit{the object is empty (only useful for collections)}\end{cases}\\
\textsc{fun:}\ \begin{cases}\text{(none)}\end{cases}\\
\textsc{sel:}\ \begin{cases}\text{(none)}\end{cases}\\
\textsc{opr:}\ \begin{cases}\text{set}(Obj) : \textit{the variable is set to Obj}\\ \text{unset} : \textit{the variable is unset (set to nil)}\\ \text{clear} : \begin{array}{l}\textit{the value of the variable is set to be an empty object of type T}\\ \textit{(only useful for collections)}\end{array}\end{cases}
\end{array}$$

Resource objects, types, and variables

Each resource must be defined as a *resource object* of a certain type, a *resource type*. The definition of this type can be regarded as a interface to whatever input/output facilities are available for the resource itself, enabling the resource to be accessed by TrindiKit. Resources are hooked up to the TIS using resource interface variables, and the value of a resource interface variable is a resource object.

A sample interface type definition for a lexicon (much like the one used by IBiS) is shown in (A.12).

$$\text{(A.12)}\quad \begin{array}{l}
\textsc{type:}\ \text{lexiconT}\\
\textsc{rel:}\ \begin{cases}\text{input\_form}(Lexicon, Phrase, Moves) : \begin{array}{l}\textit{Phrase is interpreted as}\\ \textit{Moves by Lexicon}\end{array}\\ \text{output\_form}(Lexicon, Move, Phrase) : \begin{array}{l}\textit{Move is generated as}\\ \textit{Phrase by Lexicon}\end{array}\end{cases}\\
\textsc{fun:}\ \begin{cases}\text{(none)}\end{cases}\\
\textsc{sel:}\ \begin{cases}\text{(none)}\end{cases}\\
\textsc{opr:}\ \begin{cases}\text{(none)}\end{cases}
\end{array}$$

There may be several resources of each type; for example, there may be lexicons for several languages which all are of the type lexiconT. For each object, a type declaration such as those in (A.13) is needed.

(A.13) lexicon_travel_english : lexiconT
lexicon_travel_swedish : lexiconT
lexicon_vcr_english : lexiconT
lexicon_vcr_swedish : lexiconT

To hook up a resource to the TIS, we need a resource interface variable type declaration and an assignment of a resource object to that variable, as shown in (A.14). (See Section A.3.4 for an explanation of the assignment syntax.)

(A.14) LEXICON : lexiconT
LEXICON := lexicon_travel_english

A note on the difference between modules and resources: Resources are declarative knowledge sources, external to the information state, which are used in update rules and algorithms. Modules, on the other hand, are agents which interact with the information state and are called upon by the controller. Of course, there is a procedural element to all kinds of information search, which means among other things that one must be careful not to engage in extensive time-consuming searches. Conversely, modules can be defined declaratively and thus have a declarative element. There is no sharp distinction dictating the choice between resource or module; for example, it is possible to have the parser be a resource. However, it is important to consider the consequences of choosing to see something as a resource or module.

A.3 Methods for accessing the TIS

The TIS can be accessed in three ways: *conditions*, *queries*, and *updates*. Checking and querying are ways to find out what the information state is like, and they can bind Prolog variables. Updates change the information state, but can not bind Prolog variables. Conditions are true or false, whereas query and apply-calls fail or succeed.

A.3.1 Objects, functions, locations, and evaluation

The information state in TrindiKit consists of variables whose values are objects of different types. When explaining the syntax for conditions in Section A.3.2 we will use *Obj* as a variable ranging over all objects; for example, *Obj* can be a stack or an integer. TIS variables are evaluated using the variable evaluation operator “$”, which can be regarded as a function from TIS variables to objects. The syntax rule for *Obj* allowing objects to be specified using evaluation of a TIS variable $TISvar$ is shown in (A.15).

(A.15) $Obj \rightarrow \$TISvar$

Objects can also be specified using evaluation of functions; the function evaluation operator is denoted "$$". Given a function *Fun* taking arguments $Arg_1, \ldots, Arg_n$, the syntax rule in (A.16) allows specifying an object by applying *Fun* to $Obj_1, \ldots, Obj_n$.

(A.16) $Obj \rightarrow \$\$Fun(Obj_1, \ldots, Obj_n)$

By using *paths*, built up by selectors, it is possible to "point" at an object embedded at the corresponding location inside a (complex) object and inspect or manipulate it. Paths thus appear in two contexts: inspection, where they specify objects, and manipulation, where they specify locations.

An object X can be specified by a complex object and a selector *Sel* pointing out X inside the complex object. The syntax rule for pointing out embedded objects using selectors is shown in (A.17).

(A.17) $Obj \rightarrow Obj/Sel$

This recursive definition allows selectors to be iteratively applied to objects, using expressions of the form $Obj/Sel_1/Sel_2 \ldots /Sel_n$; this is equivalent to $(\ldots((Obj/Sel_1)/Sel_2) \ldots /Sel_n)$.

Another basic concept in TrindiKit is that of *locations* in objects. The general syntax for locations is shown in (A.18); here, *Sel* is a selector and *Obj* is a complex object (a collection).

(A.18) $Loc \rightarrow Loc/Sel$
$Loc \rightarrow TISvar$

Again, the recursive definition allows selectors to be iteratively applied, using expressions of the form $TISvar/Sel_1/Sel_2 \ldots /Sel_n$; this is equivalent to $(\ldots((TISvar/Sel_1)/Sel_2) \ldots /Sel_n)$.

For example, assume we have a TIS where the information state proper (the IS variable) has the type given in (A.19) and the value given in (1.19.) (This examples assumes there are definitions of the types Proposition and Topic.)

(A.19) $$\textsc{is} : \begin{bmatrix} \textsc{beliefs} & : & \text{Set(Proposition)} \\ \textsc{topics} & = & \text{Stack(Topic)} \end{bmatrix}$$

(A.20) $$\textsc{is} = \begin{bmatrix} \textsc{beliefs} & = & \{\ \textbf{happy(sys)}, \textbf{frustrated(usr)}\ \} \\ \textsc{topics} & = & \langle\ \textbf{the_weather}, \textbf{foreign_politics}\ \rangle \end{bmatrix}$$

Then, the following holds:

- the location pointed to by IS/TOPICS contains the object ⟨ weather, foreign_politics ⟩
- \$IS/TOPICS/FST is equivalent to **the_weather**
- the location IS/BELIEFS/ELEM contains (indeterministically) some member of the set { **happy(sys)**, **frustrated(usr)** }
- ⟨**the_weather**, **foreign_politics**⟩/fst is equivalent to **the_weather**
- \$\$arity(\$IS/BELIEFS) is equivalent to 2

TrindiKit offers a shortcut representation for paths in the information state proper:

- the object \$/*Path* is equivalent to \$IS/*Path*
- the location /*Path* is equivalent to IS/*Path*

A.3.2 Conditions

The basic syntax rules for conditions is shown in (A.21). Arguments ($Arg_i, 1 \leq i \leq n$) are either objects or (if allowed by the relation definition) prolog variables.

(A.21) a. $Cond \rightarrow Rel(Arg1_1, \ldots, Arg_n$
e.g. fst(\$/TOPICS, Q)

b. $Cond \rightarrow Arg1 :: Rel(Arg_2, \ldots, Arg_n)$
e.g. \$/SHARED/QUD :: fst(Q)

c. $Cond \rightarrow Obj_1 = Obj_2$
This holds if Obj_1 and Obj_2 are unifiable.

d. $Cond \rightarrow Obj_1 == Obj_2$
This holds if Obj_1 and Obj_2 are identical.

Given conditions *Cond*, $Cond_1$ and $Cond_2$, the following constructs are also possible:

- $Cond_1$ and $Cond_2$
  This is true if $Cond_1$ is true and $Cond_2$ is true.

- $Cond_1$ or $Cond_2$
  This is true if $Cond_1$ is true or $Cond_2$ is true. $Cond_1$ will be tested first, and only if it is false will $Cond_2$ be tested.

- not $Cond$
  This is true if $Cond$ is false.

- forall($Cond_1$, $Cond_2$)
  This is equivalent to not ($Cond_1$ and (not $Cond_2$)).

- setof(Obj, $Cond$, $ObjSet$)
  $ObjSet$ is the set of objects Obj satisfying $Cond$.

Conditions and First Order Logic

In terms of First Order Logic (FOL), a condition can be seen as a proposition (which is true or false of the TIS), existentially quantified over all Prolog variables occuring in the condition - except for variables occuring only within the scope of negation or universal quantification.

An illustration is shown in (A.22) ($C_1(X)$, $C_2(X,Y)$ and $C_3(Y,Z)$ are conditions; X, Y and Z are Prolog variables, and x, y and z are the corresponding FOL variables).

(A.22) ($C_1(X)$ and $C_2(X,Y)$ and $C_3(Y,Z)$) $\sim$
$\exists x, y, z(C_1(x) \wedge C_2(x,y) \wedge C_3(y,z))$

Negation

When evaluating "not $Cond$", some Prolog variables appearing in $Cond$ may already have become bound (when evaluating a previous check). Any previously bound variables appearing in $Cond$ will still be bound when evaluating "not $Cond$.

(A.23) ($C_1(X)$ and (not $C_2(X)$) and $C_3(Y,Z)$) $\sim$
$\exists x, y, z(C_1(x)) \wedge \neg C_2(x) \wedge C_3(y,z)$

Any previously *unbound* variables appearing in $Cond$ will not be bound by checking "not $Cond$". They are interpreted as existentially quantified within the scope of the negation, as illustrated in (A.24). Any occurrences of these variables occuring in following conditions will be independent, i.e. they can be regarded as separate variables.

(A.24) $(C_1(X)$ and (not $C_2(X,Y)$) and $C_3(Y,Z)$) $\sim$
$\exists x, y, z(C_1(x)) \wedge \neg\exists y'(C_2(x,y')) \wedge C_3(y,z)$

Universal quantification

The binding behaviour of Prolog variables inside the scope of "forall" is similar to that for "not", which is natural since "forall" is defined using "not".

If $X_1, \ldots, X_n$ are variables in $Cond_1$ which are not previously bound, and $Y_1, \ldots, Y_m$ are variables in $Cond_2$ which are not previously bound *and* do not occur in $Cond_1$, the FOL interpretation is as shown in (A.25).

(A.25) forall$(Cond_1, Cond_n) \sim$
$\forall x_1, \ldots, x_n(Cond_1 \rightarrow \exists y_1, \ldots, y_m(Cond_2))$

Any previously bound variables appearing in $Cond_1$ or $Cond_2$ will still be bound when evaluating "forall$(Cond_1, Cond_2)$". Any previously *unbound* variables appearing in $Cond_1$ or $Cond_2$ will not be bound by checking "forall$(Cond_1, Cond_2)$". Any occurrences of these variables occuring in following conditions will be independent, i.e. they can be regarded as separate variables.

A.3.3 Queries

Queries are similar to conditions in that they do not modify the information state; however, they are also similar to updates in that they do not backtrack, and if they fail they produce an error message. The syntax for queries is shown in (A.26).

(A.26) $Query \rightarrow\ !Cond$

For example, a query "! in($/TOPICS, Q)" will bind Q to the topmost element of the stack in /TOPICS. However, if the stack is empty an error message will be reported.

A.3.4 Updates

Updates modify TIS, and if an update fails, an error message is reported. The basic syntax for updates is shown in (A.27).

(A.27) a. $\mathit{Update} \rightarrow \mathit{Opr}(\mathit{Loc}, \mathit{Obj}_1, \ldots, \mathit{Obj}_n)$
e.g. push(/topics, sports)

b. $\mathit{Update} \rightarrow \mathit{Loc} :: \mathit{Obj}_1, \ldots, \mathit{Obj}_n)$
e.g. /topics :: push(sports)

c. $\mathit{Update} \rightarrow \mathit{Loc} := \mathit{Obj}$
Equivalent to set($\mathit{Loc}, \mathit{Obj}$).

Given updates *Update*, $\mathit{Update}_1, \ldots, \mathit{Update}_n$, the constructions in (A.28) are also possible.

(A.28) a. $\mathit{Update} \rightarrow [\ \mathit{Update}_1, \ldots, \mathit{Update}_n\ ]$
Execute $\mathit{Update}_1, \ldots, \mathit{Update}_n$ in sequence.

b. *Update* → if_do(*Cond*, *Update*)
If *Cond* holds, execute *Update*.

c. *Update* → if_then_else(*Cond*, Update_1, Update_2)
If *Cond* holds, execute Update_1; otherwise, execute Update_2.

d. *Update* → forall_do(*Cond*, *Update*)
(See below for explanation)

It is possible to apply an operation repeatedly using a single update call with the syntax "forall_do(*Cond*, *Update*)", where *Cond* is a condition and *Update* is an update.

Let $X_1, \ldots, X_n$ be all unbound Prolog variables in *Cond* (i.e. those which are not bound when the update is applied). Now, the interpretation of "forall_do(*Cond*, *Update*)" goes as follows: "For all bindings of $X_1, \ldots, X_n$ which make *Cond* true, apply *Update*."

As an example, for all A such that A is in the set /BELIEFS, (A.29) will push A on the stack at /TOPICS.

(A.29) forall_do(in($/BELIEFS, A), push(/TOPICS, A))

A.4 Rule definition format

Update rules are rules for updating the TIS. They consist of a rule name, a precondition list, and an effect list. Preconditions are conditions, and effects can be queries or update-calls.

If the preconditions of a rule are true for the TIS, then the effects of that rule can be applied to the TIS. Rule may also belong to a class.

The rule definition format is shown in (A.30).

(A.30) RULE: *RuleName*
CLASS: *RuleClass*
PRE: $\left\{ \; PrecondList \right.$
EFF: $\left\{ \; EffectsList \right.$

Here, *PrecondList* is a list of conditions and *EffectsList* is a list of queries and updates. The *RuleClass* may be used in defining DME algorithms (Section A.5).

The precondition list $Cond_1, \ldots, Cond_2$ in a rule is equivalent to a conjunction "$Cond_1$ and ... and $Cond_n$". Any variables that become bound while checking the preconditions will still be bound when executing the effects.

A.4.1 Backtracking and variable binding in rules

When a rule is applied to the TIS, the preconditions will be evaluated in the order they appear. Since conditions may be nondeterministic, conditions containing Prolog variables may have several possible results. For example, checking "member(X)" on a set {a, b, c} has X=a, X=b and X=c as possible results.

The first time this check is made, the first solution will be returned, i.e. X will become bound to a. If a later precondition which uses X is not true for X=a, TRINDIKIT will use Prolog's backtracking facility to go back and get the second solution X=b. In this way, the TRINDIKIT rule interpreter will try to find a way to bind the Prolog variables appearing in the precondition list[4] so that all the preconditions hold.

Once this has succeeded, the effects of the rule will be applied using the variable bindings obtained when checking the preconditions. Continuing the example above, if all preconditions succeed with X bound to a, this binding will "survive" to the effects and any appearance of X in the effects will be equivalent to an appearance of a.

A.4.2 Condition and operation macros

In addition to the conditions and operations provided by the datatype definitions, it is also possible to write *macros*. Macros define sequences of conditions (for *condition macros*)

[4]apart from those appearing in the scope of a negation or universal quantification, see A.3.2 and A.3.2 respectively

or operations (for *operation macros*). Like conditions and operations (but unlike rules), macros can take arguments.

Macros are defined by associating a macro with a list of TIS conditions or a list of TIS updates.

(A.31) macro_cond(belief_and_topic(X), [in(\$/BELIEF, X), in(\$/TOPIC, X)])

(A.32) macro_update(add_to_belief_and_topic, [add(/BELIEF, X), push(/TOPIC, X)])

A.4.3 Prolog variables in the TIS

TRINDIKIT does not prohibit Prolog variables as part of the TIS, i.e. objects can contain Prolog variables. This has the advantage that it is possible to have partially uninstantiated objects (*non-ground terms* in Prolog terminology) which may become fully instantiated at a later point.

However, this also makes it possible for checks and queries to temporarily change the information state by unifying a partially instantiated object in the TIS with a more specific object. For example, if the value of the TIS variable LATEST_MOVE is ask(**?happy(**X**)**), checking \$LATEST_MOVE = ask(**?happy(john)**) will have the effect that LATEST_MOVE now has the value ask(**?happy(john)**).

If this check is part of the preconditions list of a rule, and a later precondition fails, TRINDIKIT may backtrack which may result in X becoming unbound again. These bindings "survive" within the scope of an update rule or a sequence of conditions. After the rule or sequence of checks is done, any Prolog variables in the TIS will again be unbound.

A.5 The DME-ADL language

DME-ADL (Dialogue Move Engine Algorithm Definition Language) is a language for writing algorithms for updating the TIS. Algorithms in DME-ADL are expressions of any of the following kinds (C is a TIS condition; R, S and T are algorithms, *Rule* is the name of an update rule, and *RuleClass* is a rule class):

1. *Rule*
   apply the update rule *Rule*
2. *RuleClass*
   apply an update rule of class *RuleClass*; rules are tried in the order they are declared
3. $[R_1, \ldots, R_n]$
   execute $R_1, \ldots, R_n$ in sequence
4. if C then S else T
   If C is true of the TIS, execute S; otherwise, execute T
5. while C do R
   while C is true of the TIS, execute R repeatedly
6. repeat R until C
   execute R repeatedly until C is true of the TIS
7. repeat R
   execute R repeatedly until it fails; report no error when it fails
8. repeat+ R
   execute R repeatedly, but at least once, until it fails; report no error when it fails
9. try R
   try to execute R; if it fails, report no error
10. R orelse S
    Try to execute R; if it fails, report no error and execute S instead
11. test C
    if C is true of the TIS, do nothing; otherwise, halt execution of the current algorithm
12. apply *Op*
    apply operation *Op*
13. *SubAlg*
    execute subalgorithm *SubAlg*

Subalgorithms are declared using $\Rightarrow$, which is preceded by the subalgorithm name and followed by the algorithm, as in (A.33).

(A.33) main_update $\Rightarrow \langle$ grounding,
repeat+ (integrate orelse accommodate) $\rangle$

A sample DME-ADL algorithm is shown in (A.34).

```
(A.34) if $LATEST_MOVES == failed
       then repeat refill
       else ⟨ grounding,
              repeat+ ( integrate orelse accommodate ),
              if $LATEST_SPEAKER == usr
              then ⟨ repeat refill,
                     try database ⟩
              else store
              ⟩
```

A.6 The Control-ADL language

The control algorithm specifies whether a system should be run in serial or asynchronously. Serial algorithms are simpler than asynchronous ones, and the syntax used for asynchronous algorithms subsumes the syntax for serial algorithms. Since IBiSuses only serial control, we will not introduce the syntax for asynchronous control here.

A.6.1 Serial control algorithm syntax

The serial Control-ADL language is similar to the DME-ADL language, except that it calls module algorithms instead of rules, and it can include the "print_state" instruction. Each algorithm has a name, and each module may define one or more algorithms.

A sample serial Control-ADL algorithm is shown in (A.35).

```
(A.35) ⟨ reset,
         repeat ⟨ select,
                  generate,
                  output,
                  update,
                  print_state,
                  test( $PROGRAM_STATE == run ),
                  input,
                  interpret,
                  update,
                  print_state ⟩
       ⟩
```

A.7 Provided modules

The TrindiKit package includes a couple of simple modules which can be used to quickly build prototype systems.

- **input_simpletext**: a simple module which reads text input from the user and stores it in the TIS
- **output_simpletext**: a simple text output module
- **intpret_simple1**: an interpretation module which uses a lexicon of key words and phrases to interpret user utterances in terms of dialogue moves
- **generate_simple1**: a generation module which uses a lexicon of mainly canned sentences to generate system utterances from moves

A.7.1 Simple text input module

The input module **input_simpletext** reads a string (until new-line) from the keyboard, preceded by the printing of an input prompt. The system variable INPUT is then set to be the value read.

A.7.2 Simple text output module

The output module **output_simpletext** takes the string in the system variable OUTPUT and prints it on the computer screen, preceded by the printing of an output prompt. The contents of the OUTPUT variable is then deleted. The module also moves the contents of the system variable NEXT_MOVES to the system variable LATEST_MOVES. Finally it sets the system variable LATEST_SPEAKER to be the system.

A.7.3 A simple interpretation module

The interpretation module **interpret_simple1** takes a string of text, turns it into a sequence of words (a "sentence") and produces a set of moves. The "grammar" consists of pairings between lists whose elements are words or semantically constrained variables. Semantic constraints are implemented by a set of semantic categories (**location**, **month**,

means_of_transport etc.) and synonymy sets. A synonymy set is a set of words which all are regarded as having the same meaning.

The simplest kind of lexical entry is one without variables. For example, the word "hello" is assumed to realize a greet move.:

(A.36) input_form([hello], greet)

The following rule says that a phrase consisting of the word "to" followed by a phrase S constitutes an answer move with content **to**(C) provided that the lexical semantics of S is C, and C is a `location`. The lexical semantics of a word is implemented by a coupling between a synset and a meaning; the lexical semantics of S is C, provided that S is a member of a synonymy set of words with the meaning C.

(A.37) input_form([to| S], answer(to(C)) $\leftarrow$ lexsem(S, C), location(C).

To put it simply, the parser tries to divide the sentence into a sequence of phrases (found in the lexicon), covering as many words as possible.

A.7.4 A simple generation module

The generation module **generate_outputform** takes a sequence (list) of moves and outputs a string. The generation grammar/lexicon is a list of pairs of move templates and strings.

(A.38) output_form(greet, "Welcome to the travel agency!").

To realize a list of Moves, the generator looks, for each move, in the lexicon for the corresponding output form (as a string), and then appends all these strings together. The output strings is appended in the same order as the moves.

Appendix B

Rules and classes

This appendix lists rule classes used by the various versions of IBiS. Rules are listed in the order they are tried when the corresponding rule class is called in a module algorithm.

The IBiS systems and TrindiKit can be downloaded from:
`http://www.ling.gu.se/~sl/Thesis`.

The size of the systems range from approximately 1,200 lines of code (32kbyte) for IBiS1 to about 2,500 lines (75 kbyte) for IBiS4, excluding domain-specific resources.

B.1 IBiS1

B.1.1 IBiS1 update module

- grounding
  - **getLatestMove** (RULE 3.1) (p. 41)
- integrate
  1. **integrateUsrAsk** (RULE 3.3) (p. 44)
  2. **integrateSysAsk** (RULE 3.2) (p. 43)
  3. **integrateAnswer** (RULE 3.4) (p. 47)
  4. **integrateGreet** (RULE 3.6) (p. 48)
  5. **integrateSysQuit** (RULE 3.8) (p. 49)

6. **integrateUsrQuit** (RULE 3.7) (p. 48)

- downdate_qud

  1. **downdateQUD** (RULE 3.5) (p. 48)
  2. **downdateQUD2** (RULE 3.16) (p. 63)

- load_plan

  1. **recoverPlan** (RULE 3.17) (p. 64)
  2. **findPlan** (RULE 3.9) (p. 49)

- exec_plan

  1. **removeFindout** (RULE 3.10) (p. 50)
  2. **removeRaise** (RULE 3.19) (p. 66)
  3. **exec_consultDB** (RULE 3.11) (p. 50)

B.1.2 IBiS1 select module

- select_action

  1. **selectRespond** (RULE 3.14) (p. 53)
  2. **selectFromPlan** (RULE 3.12) (p. 51)
  3. **reraiseIssue** (RULE 3.18) (p. 65)

- select_move

  1. **selectAnswer** (RULE 3.15) (p. 54)
  2. **selectAsk** (RULE 3.13) (p. 52)
  3. **selectOther**

B.2 IBiS2

B.2.1 IBiS2 update module

- grounding

  - **getLatestMoves** (RULE 4.16) (p. 130)

- integrate
    1. **integrateUsrAsk** (RULE 4.1) (p. 110)
    2. **integrateSysAsk** (RULE 4.18) (p. 132)
    3. **integrateNegIcmAnswer** (RULE 4.7) (p. 115)
    4. **integratePosIcmAnswer** (RULE 4.8) (p. 116)
    5. **integrateUsrAnswer** (RULE 4.4) (p. 113)
    6. **integrateSysAnswer** (RULE 4.19) (p. 132)
    7. **integrateUndIntICM** (RULE 4.6) (p. 115)
    8. **integrateUsrPerNegICM** (RULE 4.20) (p. 133)
    9. **integrateUsrAccNegICM** (RULE 4.21) (p. 135)
    10. **integrateOtherICM** (RULE 4.10) (p. 121)
    11. **integrateGreet**
    12. **integrateSysQuit**
    13. **integrateUsrQuit**
    14. **integrateNoMove**
- downdate_qud
    - **downdateQUD**
    - **downdateQUD2**
- load_plan
    1. **recoverPlan** (RULE 4.24) (p. 143)
    2. **findPlan** (RULE 4.23) (p. 143)
- exec_plan
    1. **removeFindout**
    2. **exec_consultDB**
- (none)
    - **irrelevantFollowup** (RULE 4.22) (p. 141)
    - **unclearFollowup**

B.2.2 IBiS2 select module

- select_action
    1. **rejectIssue** (RULE 4.15) (p. 129)
    2. **rejectProp** (RULE 4.14) (p. 127)
    3. **selectIcmUndIntAsk** (RULE 4.3) (p. 112)
    4. **selectIcmUndIntAnswer** (RULE 4.5) (p. 114)
    5. **selectRespond** (RULE 4.26) (p. 147)
    6. **selectFromPlan**
    7. **reraiseIssue** (RULE 4.25) (p. 144)
- select_icm
    1. **selectIcmConNeg** (RULE 4.9) (p. 120)
    2. **selectIcmPerNeg** (RULE 4.11) (p. 121)
    3. **selectIcmSemNeg** (RULE 4.12) (p. 122)
    4. **selectIcmUndNeg** (RULE 4.13) (p. 123)
    5. **selectIcmOther** (RULE 4.2) (p. 111)
- select_move
    1. **selectAnswer** (RULE 4.27) (p. 147)
    2. **selectAsk**
    3. **selectOther**
    4. **selectIcmOther** (RULE 4.2) (p. 111)
- (none)
    - **backupShared** (RULE 4.17) (p. 131)

B.3 IBiS3

B.3.1 IBiS3 update module

- grounding
    - **getLatestMoves**

- integrate
  1. **retract** (RULE 5.7) (p. 180)
  2. **integrateUsrAsk**
  3. **integrateSysAsk**
  4. **integrateNegIcmAnswer** (RULE 5.10) (p. 184)
  5. **integratePosIcmAnswer** (RULE 5.11) (p. 189)
  6. **integrateUsrAnswer**
  7. **integrateSysAnswer**
  8. **integrateAccommodationICM**
  9. **integrateUndPosICM**
  10. **integrateUndIntICM**
  11. **integrateUsrPerNegICM**
  12. **integrateUsrAccNegICM**
  13. **integrateOtherICM**
  14. **integrateGreet**
  15. **integrateSysQuit**
  16. **integrateUsrQuit**
  17. **integrateNoMove**
- accommodate
  1. **accommodateIssues2QUD** (RULE 5.2) (p. 169)
  2. **accommodateQUD2Issues** (RULE 5.9) (p. 183)
  3. **accommodatePlan2Issues** (RULE 5.1) (p. 166)
  4. **accommodateCom2Issues** (RULE 5.6) (p. 179)
  5. **accommodateCom2IssuesDependent** (RULE 5.8) (p. 182)
  6. **accommodateDependentIssue** (RULE 5.4) (p. 172)
- downdate_issues
  - **downdateISSUES**
  - **downdateISSUES2** (similar to downdateQUD2 in IBiS1)
- downdate_qud
  - **downdateQUD**

- load_plan
    1. **recoverPlan**
    2. **findPlan**
- exec_plan
    1. **removeFindout**
    2. **exec_consultDB**
- select_action
    1. **selectIcmUndIntAsk**
    2. **selectIcmUndIntAnswer**
    3. **selectIcmUndIntRequest**
    4. **rejectIssue**
    5. **rejectAction**
    6. **rejectProp**
- none
    - **irrelevantFollowup**
    - **unclearFollowup**
    - **failedFollowup**
    - **noFollowup** (RULE 5.12) (p. 190)
    - **backupSharedUsr** (RULE 5.13) (p. 193)

B.3.2 IBiS3 select module

- select_action
    1. **clarifyIssue** (RULE 5.3) (p. 170)
    2. **clarifyDependentIssue** (RULE 5.5) (p. 176)
    3. **selectRespond**
    4. **selectFromPlan**
    5. **reraiseIssue**
- select_icm
    1. **selectIcmConNeg**

2. **selectIcmPerNeg**
3. **selectIcmSemNeg**
4. **selectIcmUndNeg**
5. **selectIcmOther**

- select_move
  1. **selectQuit**
  2. **selectAnswer**
  3. **selectAsk**
  4. **selectGreet**
  5. **selectIcmOther**
- none
  - **backupSharedSys**

B.4 IBiS4

B.4.1 IBiS4 update module

- grounding
  - **getLatestMoves**
- integrate
  1. **retract**
  2. **integrateUsrAsk**
  3. **integrateSysAsk**
  4. **integrateUsrRequest** (RULE 6.1) (p. 216)
  5. **integrateConfirm** (RULE 6.6) (p. 218)
  6. **integrateNegIcmAnswer**
  7. **integratePosIcmAnswer**
  8. **integrateUsrAnswer**
  9. **integrateSysAnswer**
  10. **integrateAccommodationICM**

11. **integrateUndPosICM**
12. **integrateUndIntICM**
13. **integrateUsrPerNegICM**
14. **integrateUsrAccNegICM**
15. **integrateOtherICM**
16. **integrateGreet**
17. **integrateSysQuit**
18. **integrateUsrQuit**
19. **integrateNoMove**

- accommodate
  1. **accommodateIssues2QUD**
  2. **accommodateQUD2Issues**
  3. **accommodatePlan2Issues**
  4. **accommodateCom2Issues**
  5. **accommodateCom2IssuesDependent**
  6. **accommodateDependentIssue**
  7. **accommodateAction** (RULE 6.8) (p. 221)
- downdate_issues
  1. **downdateISSUES**
  2. **downdateISSUES2**
  3. **downdateISSUES3** (downdates resolved action-issue)
  4. **downdateActions** (RULE 6.7) (p. 218)
- downdate_qud
  - **downdateQUD**
- load_plan
  1. **findPlan**
  2. **findActionPlan**
- exec_plan
  1. **recoverPlan**
  2. **recoverActionPlan**

3. **removeFindout**
4. **exec_consultDB**
5. **exec_dev_get**
6. **exec_dev_set**
7. **exec_dev_do** (RULE 6.3) (p. 217)
8. **exec_dev_query**

- select_action

1. **selectIcmUndIntAsk**
2. **selectIcmUndIntAnswer**
3. **selectIcmUndIntRequest**
4. **rejectIssue**
5. **rejectProp**
6. **rejectAction** (RULE 6.2) (p. 216)

- (none)

  – **backupSharedUsr**
  – **irrelevantFollowup**
  – **irrelevantFollowup**
  – **unclearFollowup**
  – **failedFollowup**
  – **noFollowup**
  – **declineProp**

B.4.2 IBiS4 select module

- select_action

1. **clarifyIssue**
2. **clarifyIssueAction** (RULE 6.9) (p. 221)
3. **selectConfirmAction** (RULE 6.4) (p. 217)
4. **selectRespond**
5. **reraiseIssue**
6. **selectFromPlan**

- select_icm
  1. **selectIcmConNeg**
  2. **selectIcmPerNeg**
  3. **selectIcmSemNeg**
  4. **selectIcmUndNeg**
  5. **selectIcmOther**
- select_move
  1. **selectQuit**
  2. **selectAnswer**
  3. **selectAsk**
  4. **selectConfirm** (RULE 6.5) (p. 217)
  5. **selectGreet**
  6. **selectIcmOther**
- (none)
  - **backupSharedSys**