A model of dialogue moves and information state revision

David Traum Johan Bos Robin Cooper Staffan Larsson
Ian Lewin Colin Matheson Massimo Poesio

Distribution: PUBLIC

Task Oriented Instructional Dialogue
LE4-8314

Deliverable D2.1

November 1999

LE4-8314 TRINDI

Task Oriented Instructional Dialogue

Gothenburg University
Department of Linguistics

University of Edinburgh
Centre for Cognitive Science and Language Technology Group, Human Communication Research Centre

Universität des Saarlandes
Department of Computational Linguistics

SRI Cambridge

Xerox Research Centre Europe

For copies of reports, updates on project activities and other TRINDI-related information, contact:

The TRINDI Project Administrator
Department of Linguistics
Göteborg University
Box 200
S-405 30 Gothenburg, Sweden
trindi@ling.gu.se

Copies of reports and other material can also be accessed from the project's homepage, http://www.ling.gu.se/research/projects/trindi.

Responsibility for authorship is divided as follows. David Traum was the overall editor. Chapter 3 was written by Staffan Larsson and David Traum. Chapter 4 was written by Staffan Larsson and Robin Cooper. Chapter 5 was written by Massimo Poesio and David Traum, with contributions from Colin Matheson and Johan Bos on formalizations 2 and 3, respectively. Chapter 6 was written by Ian Lewin.

Contents

Chapter 1

Introduction

The term INFORMATION STATE of a dialogue represents the information necessary to distinguish it from other dialogues, representing 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". Generally, although not necessarily, we will also talk about the information state of participants of the dialogue, representing the information that those participants have at a particular point in the dialogue - what they brought with them to the dialogue, what they pick up, and how they are motivated to act in the (near) future.

In this document we will be concerned with presenting a particular view toward the formalization of the notion of information state in such a way that allows specific theories and variants to be formalized, implemented, tested, and compared. Key to this approach will be a notion of UPDATE of information state, with most updates related to the observation and performance of DIALOGUE MOVES. Rather than confine our investigations to a single theory of information state, which we feel is premature at this point, given the range of candidate approaches and limited knowledge of what works best in practice, we instead present a more general framework for developing and testing such theories, along with several instantiated candidate theories which appear promising.

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

- a description of the informational components of the information state (e.g., participants, beliefs, common ground, intentions, etc.)
- formal representations of the above components (e.g., as typed feature structures, lists, sets, 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 behavior will also require rules for recognizing and realizing the performance of these moves, e.g., with traditional 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. Some of these rules will also *select* particular dialogue moves for the system to perform (in the case of a system participating in a dialogue rather than just monitoring one)

- a control 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.

It is important to distinguish information state approaches to dialogue modeling from other, structural dialogue state approaches. These latter approaches conceive a "legal" dialogue as behaving according to some grammar, with the states representing the results of performing a dialogue move in some previous state, and each state licensing a set of allowable next dialogue moves. The "information" is thus implicit in the state itself and the relationship it plays to other states. It may be difficult to transform an information state view to a dialogue state view, since there's no finiteness restriction on information states (depending on the type of information modeled), and the motivations for update and picking a next dialogue move (using update rules, selection rules, and control strategy) may rely on only a part of the information available, rather than the whole state. On the other hand, it is very easy to model dialogue state as information state: the information is the dialogue state, itself. This is easily modeled as a register indicating the state number (for finite state models), or a stack (for recursive transition networks). The dialogue moves will be the same set from the dialogue state theory, the update and selection rules will be the transitions in the dialogue state theory, formulated as an update to a new state, given the previous state and performance of the action, and the control strategy will be much the same as in the transition network (i.e., deterministic or non-deterministic, etc.)

Structural dialogue state approaches have often been contrasted with plan-based approaches to dialogue modeling (e.g., by Cohen (1996); Sadek and De Mori (1998)). Structure-based approaches are usually viewed as viable for simple, scripted dialogues, while plan-based approaches, though more complex and difficult to embed in practical dialogue systems, are seen as more amenable to flexible dialogue behavior. Plan-based approaches are also criticized as being more opaque, especially given the large amount of procedural processing and lack of a well-founded semantics for plan-related operations. An information-state approach allows one to fruitfully combine the two approaches, using the advantages of each. The information state may include aspects of dialogue state as well as more mentalistic notions such as beliefs, intentions, plans, etc. Moreover, casting the updates in terms of update and selection rules provides for a more transparent, declarative representation of system behavior.

In the next chapter, we describe how popular theories of dialogue can be cast in the framework of "information state". In chapter 3, we describe the uses to be made of information state,

namely analysis of dialogues (as described in Cooper *et al.* (1999)), and participation in dialogue, using a DIALOGUE MOVE ENGINE described in more detail in Larsson *et al.* (1999). In the following chapters, we present the views of information state in dialogue that we are currently exploring. More details on implementations of these theories can be found in the appendices to Bos *et al.* (1999).

Chapter 2

Specifying Information State

In this chapter we describe in a little more detail what is meant by the various components of information state, outlining some of the choices available to the theorist. We also present a quick overview of some popular accounts of dialogue, viewed from this conceptual framework.

2.1 Aspects of Information State

2.1.1 Informational Components

Information state is usually conceived of not as a monolithic node in a transition network (as with dialogue state), but as consisting of several interacting components. There is a wide range of possibilities for what kinds of components should be modeled to track the dialogue. The first choice point comes as to whether to model the participants' internal state, or more external aspects of the dialogue. To use the information state as a resource for participating in a dialogue (using selection rules to pick the next action to perform) one must obviously model the internal state of that participant. However, for other aspects of dialogue analysis, such as calculating the "winner" of a debate, one might choose to take an external point of view, based more on what was said than what was in the minds of the participants when things were said. There are also many ways of modeling the internal state of a participant. One can choose to model the mental state of the agent (attitudes such as belief, desire, intention, along with social correlates such as mutual belief, joint intention, and obligation), or one can take a more structural view of the dialogue, concentrating on the performance of actions and various sorts of accessibility relationships. It may also be useful to distinguish components of information state into *static* and *dynamic* aspects. The former are those aspects of information state that are not expected to change during the course of a dialogue, but are still very useful to modeling the progression of the dialogue. Examples of static information state components could include things like domain knowledge, or knowledge of dialogue conventions. It depends on the type of dialogue being modeled, and the scope of the conversation as to which aspects will be assumed to be static vs. dynamic (e.g., contrasting

knowledge acquisition system vs. a question answering system - the former may want to treat domain knowledge as dynamic, while the latter would see it as static). Marking some information as static may have some advantages in efficient implementation, since various compilation shortcuts and memory allocation could be performed.

2.1.2 Formal Representations

Given a choice of what aspects of the dialogue structure and the participants internal state to model, the question then arises as to *how* to model them. There are a wide number of choices, from simple abstract data types, to more complex informational systems, such as logics (with associated inference systems) and statistical systems of various flavors. These choices will be related to the particular theory of accessibility of these elements, and will also affect other processing issues, such as comprehensiveness and efficiency. As an example, consider an aspect of information state such as actions to be performed in the dialogue (e.g., an agenda, plan, bligations or intentions). There's a choice as to whether to represent tokens separately (e.g., with some sort of list) or just types, not distinguishing between multiple tokens of the same type (e.g., with a set). Given a choice of representing a list, there is still the question of accessibility - should it be a FIFO queue, a LIFO stack, or some more open structure, allowing access to the whole list, for at least some of the operations of addition and deletion. Likewise, if an agent's beliefs are represented, should this be a set, some sort of ordered list, or a complete logical inference system, in which implicit beliefs are also said to hold given some configuration of explicit beliefs.

2.1.3 Dialogue Moves

Dialogue moves are meant to serve as an abstraction between the large number of different possible messages that can be sent (especially in a natural language) and the types of update to be made on the basis of performed utterances. Dialogue moves can also provide an abstract level for content generation. There are also a number of dialogue move taxonomies to choose from. In Cooper *et al.* (1999), we discussed several extant taxonomies. Some principles regarding this issue are also outlined in Traum (1999). There must be at least sufficient types of dialogue moves to provide the different kinds of updates desired. The set of dialogue moves to choose is also influenced by the task of language interpretation – how easy will it be to (reliably) determine that one move vs. another has been performed? Another complicating issue is how to capture the inherent multi-functionality of utterances – with complex moves and move taxonomies, where each move has multiple functions, but a single utterances is described by a single move, or multiple strata of act taxonomies, where each move has a more or less simple function, but multiple moves are required to describe an utterance.

2.1.4 Update Rules

Update rules formalize the way that information state is updated. Each rule consists of a set of applicability conditions, and effects. The applicability conditions are related to aspects of the information state which must be present for the rule to be appropriate. These may include the performance of a particular type of move as one of the conditions. Other update rules may govern other changes to the information state, as a result of internal deliberation or inference. Other rules, called *selection rules* have, as one effect, the selection of a particular dialogue move, which should be performed by the participant being modeled.

2.1.5 Control Strategy

Along with the set of update rules, a strategy for how to apply the rules is needed. This is, in many cases, going to be crucial for the design of the rules, themselves. Given a particular control strategy, one may need to make adaptations to the rules, and perhaps also aspects of the information state itself, in order to guarantee a particular sequence of rule applications. There's also a question of whether to have separate strategies for choosing different types of rules, such as some for update and others for dialoguie move selection rules, and whether these processes can be ordered or asynchronously applied. Some of the types of control strategies we have considered include:

1. Take the first rule that applies (iteratively until no rules apply)
2. Apply each rule (if applicable) in sequence
3. Choose among applicable rules using probabilistic information
4. present choices to user to decide

There is thus a synergy between choices for the components of information state: the conceptual notions, formal representations, dialogue moves, update and selection rules, and control strategy. A complete theory of dialogue update will need to include a smoothly interacting combination of these aspects. However, it is still very possible to hold some of them constant while trying out other possibilities for the others. E.g., different ways of representing a concept, different rules for doing updates, or different control strategies for applying rules. This experimentation is a central part of our current work. In the next chapter, we present an architecture for implementing and testing this approach to information state update, with several example instantiations in the following chapters. In the rest of this chapter, we show how this information state approach to dialogue can be used to view several other popular views of dialogue.

2.2 Dialogue Theories Viewed From The Information State Approach

The approach to formalizing dialogue modeling as a combination of information state, dialogue moves, and update, as outlined in the previous section can be fruitfully used to describe and compare extant theories of dialogue. In this section, we begin such a study, with a particular focus on those aspects related to question answering.

2.2.1 Finite-state Dialogue Models

In the spoken language processing community, it is currently very popular to build dialogue systems using finite state dialogue models, indicating the course of legal conversations. There have been numerous simple systems built using this framework, for simple tasks such as query answering and template filling. There are also a number of toolkits available for building such systems by wiring together dialogue states with expected recognitions and actions. Some of these are described and evaluated in Bohlin *et al.* (1999).

In viewing this kind of dialogue model as information state, the dynamic information state is essentially a state variable (or a stack of such variables, for a recursive network). The static information state would include the transition network, outlining possible transitions from each state. Dialogue moves would be the classifications on input allowing choice of transitions (the choice of input varies widely in such systems, from keywords to more abstract sentence mood or illocutionary labels). Update rules would look at the last dialogue move observed and include whatever processing the system needed to do, but crucially including a transition to a new state. Selection rules would also include some sort of output to the user. A control strategy would mediate which of multiple applicable rules to apply, in the same manner that the network would have to decide which transition to choose in the case of multiple applicable transitions. For most of these simple systems, there is only one rule at a particular point.

In terms of question answering, usually a quite simplistic model is employed, consisting of two states, one in which the question is posed, and another in which the answer is provided (or sometimes multiple result states, depending on the answer). For user questions, the s-rule that provides a transition from question state to answer state also includes code to look up the answer and produce it as a next move. For system queries, there will be one or more u-rules, which provide a transition given particular input. Occasionally this network will be augmented with other states and transitions to cover non-understandings and un-expected answers, often by repeating (or perhaps rephrasing) the question. Generally this kind of sub-network will be embedded in a larger dialogue model, allowing only the questions and answers anticipated by the system designer.

2.2.2 Classical Plan Based Models

The plan-based approach (e.g., works following the framework outlined in Cohen (1978); Cohen and Perrault (1979); Allen (1979); Allen and Perrault (1980b); Perrault and Allen (1980); Allen (1983) views dialogue from the perspective of cooperating agents. The information state is thus a model of the mental state of the participant, crucially also including within that a model of the other participant, as well. The dynamic mental state usually consists of three basic attitudes: *belief* (also called knowledge, especially when regarding the beliefs of another agent thought to be true), concerning an agent's view of the world (including the mental state of other agents), *desires* (or wants, or goals), indicating the basic motivation for action, and *intentions* (or plans), indicating the way the agent has decided to act. Static aspects of the information state include speech act definitions, based more or less on Searle's work Searle (1969). Also included were cooperativity assumptions, such as a "cause-to-want" operator, in which an agent would adopt a new desire on the basis of a belief that the other agent has such a desire.

The dialogue moves were mainly *request* and *inform*. The latter, in particular, could be parametrized by the type of information. E.g., *inform-if* for telling the truth value of a proposition, and *inform-ref* for telling a parameter that provides a missing description making a lambda abstracted proposition true. The moves were also recursive, in the sense that one could request another move, such as an inform.

Update operators concern changes to the basic attitudes - adopting new beliefs on the basis of observation, adopting new plans on the basis of desires and beliefs, and adopting new desires on the basis of the cooperativity assumptions. Also important is inferring the plans held by the other participant, on the basis of performed actions and other beliefs. Selection involves picking the right speech act on the basis of the current plan structure (one for which all pre-conditions. Other rules and control strategies were highly dependent on the particular implementation. For instance, the system in Allen and Perrault (1980b) would also have possible plans, and "ratings" associated with partial plans. Heuristic rules would change the weightings of these plans based on different factors, such as truth of preconditions and effects, and rules to specify new plans, identify referents, and accept plans. The control strategy decides on which rules to apply based on the rating of partial plans in the information state, and heuristics on the type of task, itself.

Chapter 3

Implementing and testing: TRINDIKIT

In this chapter we present a way of implementing the approach to information state outlined in the previous chapters. The implementation is in the form of a DIALOGUE MOVE ENGINE (DME), that, together with some connective material, forms the dialogue management and discourse tracking aspects of a dialogue system. TRINDIKIT, described more fully in Larsson *et al.* (1999) is a toolkit for building and experimenting with *dialogue move engines* and *information states*. It thus allows straightforward implementation of a computational theory of information state update as part of a dialogue system. Apart from proposing a general system architecture, thie TRINDIKIT package also specifies formats for defining information states, update rules dialogue moves and associated algorithms. It further provides a set of tools for experimenting with different formalizations of implementations of information states, rules, and algorithms.

To build a dialogue move engine, one needs to provide the computational theory of information state, as described in the previous chapter. This includes definitions of update rules, moves and algorithms, as well as the internal structure of the information state. One may also add inference engines, planners, plan recognizers, dialogue grammars, dialogue game automata etc..

The DME forms the core of a complete dialogue system. Simple interpretation, generation, input and output modules are also provided by the TRINDIKIT, to simulate a end-to-end dialogue system.

3.1 The TRINDIKIT architecture

The general architecture we are assuming is shown in Figure 3.1.

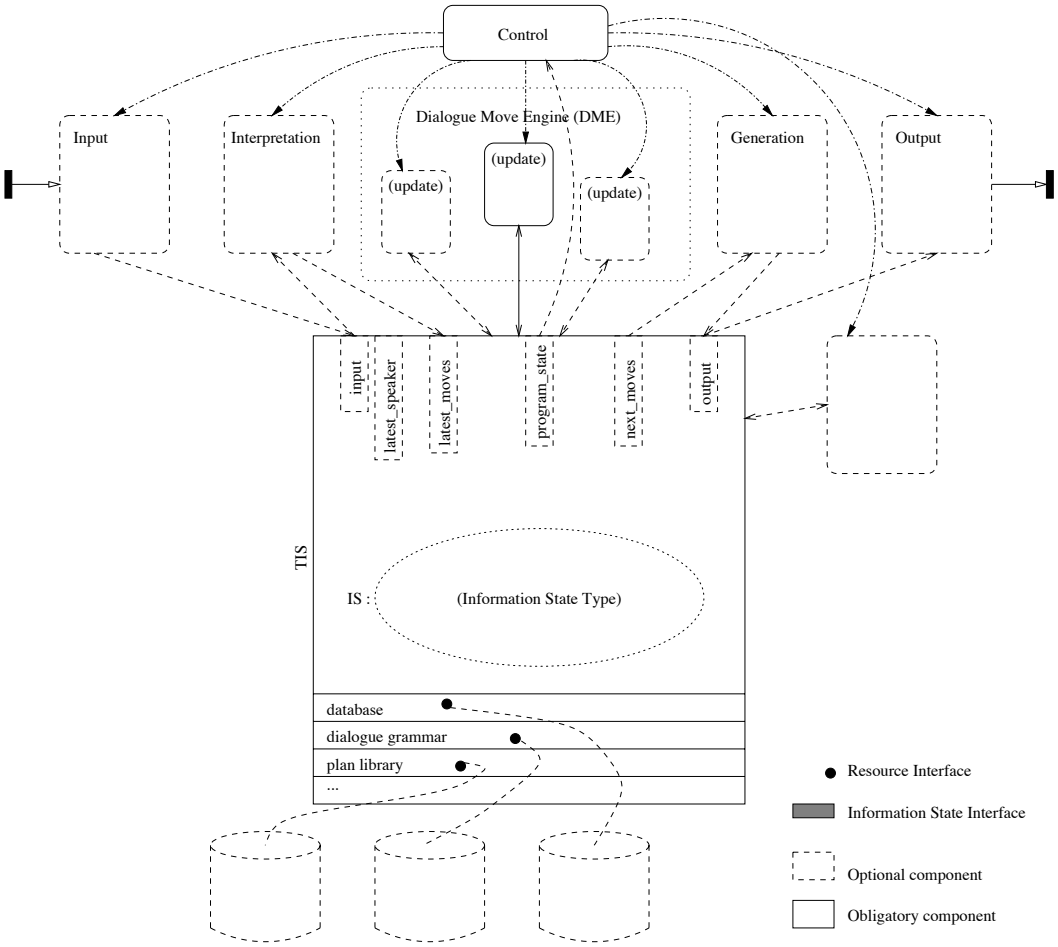


Figure 3.1: The TRINDIKIT architecture

The component of the architecture are the

- the Total Information State (TIS), consisting of
  - the Information State (IS)
  - module interface variables
  - resource interfaces
- the Dialogue Move Engine, consisting of one or more DME modules
- non-DME modules
- a control module, wiring together the other modules, either in sequence or through some asynchronous mechanism.

DME modules are responsible for updating the IS based on observed moves, and selecting moves to be perfpormed by the system. DME modules are defined using update rules and and update algorithm, while non-DME modules can be any piece of code.

Some of the components are obligtory, and others are optional or user-defined. To build a system, one must minimally supply

- An information state type.
- At least one DME module, consisting of TIS update rules and an algorithm.
- A control module, operating according to a control algorithm

The TIS is accessed by modules through conditions and operations. The types of the various components of the TIS determine which conditions and operations are available. DME modules have full access to all parts of the TIS.

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.
- 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.
- 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.

One possible setup of DME-external modules, indicated by dashed lines in figure 3.1, is the following:

- Input: Receives input utterances from the user and stores it in the input interface variable.
- Interpretation: Takes utterances (stored in input) and gives interpretations in terms of dialogue moves (including semantic content). The interpretation is stored in the interface variable latest_moves.
- Generation: Generates output moves based on the contents of next_moves and passes these on to the output interface varible.
- Output: Produces system utterances based on the contents of the output variable

Corresponding to these modules, one could have the following interface variables:

- input: the input utterance(s)
- lastest_moves: list of moves currently under consideration
- latest_speaker: the speaker of the latest move
- next_moves: list of moves for system's next turn
- output: the system's output utterance(s)
- program_state: current state of the DME (used for control)

A possible setup of the DME is to divide it into two modules, one for updating the TIS based on the latest move and one for selecting the next move:

- Update: Applies update rules to the DIS according to the update engine algorithm (also specified in SIS)
- Selection: Selects dialogue move(s) using the selection rules and the move selection algorithm. The resulting moves are stored in next_moves.

Note that the illustrated module setup is just an example. The TRINDIKIT provides methods for defining any number of both DME-modules and DME-external modules, with associated interface variables.

3.2 Toolkit provided functionality

Apart from the general architecture defined above, the TRINDIKIT provides

- definitions of datatypes, for use in TIS variable definitions
- a language and format for specifying TIS update rules
- methods for accessing the TIS
- an algorithm definition language for DME and control modules
- default modules for input, interpretation, generation and output
- methods for converting items from one type to another
- methods for visually inspecting the TIS
- debugging facilities

Chapter 4

Instantiation I: Focus on Questions Under Discussion

The overall structure of the GoDiS system is illustrated in Figure 4.1.

In addition to the control module, there are six modules in GoDiS: input, interpret, generate, output, update and select. The last two are DME modules, which means that they together make up the DME in GoDiS. There are six module interface variables, three resources and a record structure for the information state.

The question about what should be included in the information state is central to any theory of dialogue management. The notion of information state we are putting forward here is basically a version of the dialogue game board which has been proposed by Ginzburg. We are attempting to use as simple a version as possible in order to have a more or less practical system to experiment with. The main emphasis is on the following points:

1. Build on previous work in Ginzburg (1996) and Cooper and Larsson (1999) investigating information-seeking dialogues using a representation of Questions Under Discussion (QUD)
2. Extend this work to a processing model of dialogue, and implement it
3. Explore the notions of grounding and question and task accommodation within the QUD framework

4.1 Informational components

Here we present the information state, the resources and the module interface variables which together make up the total information state (TIS) in GoDiS.

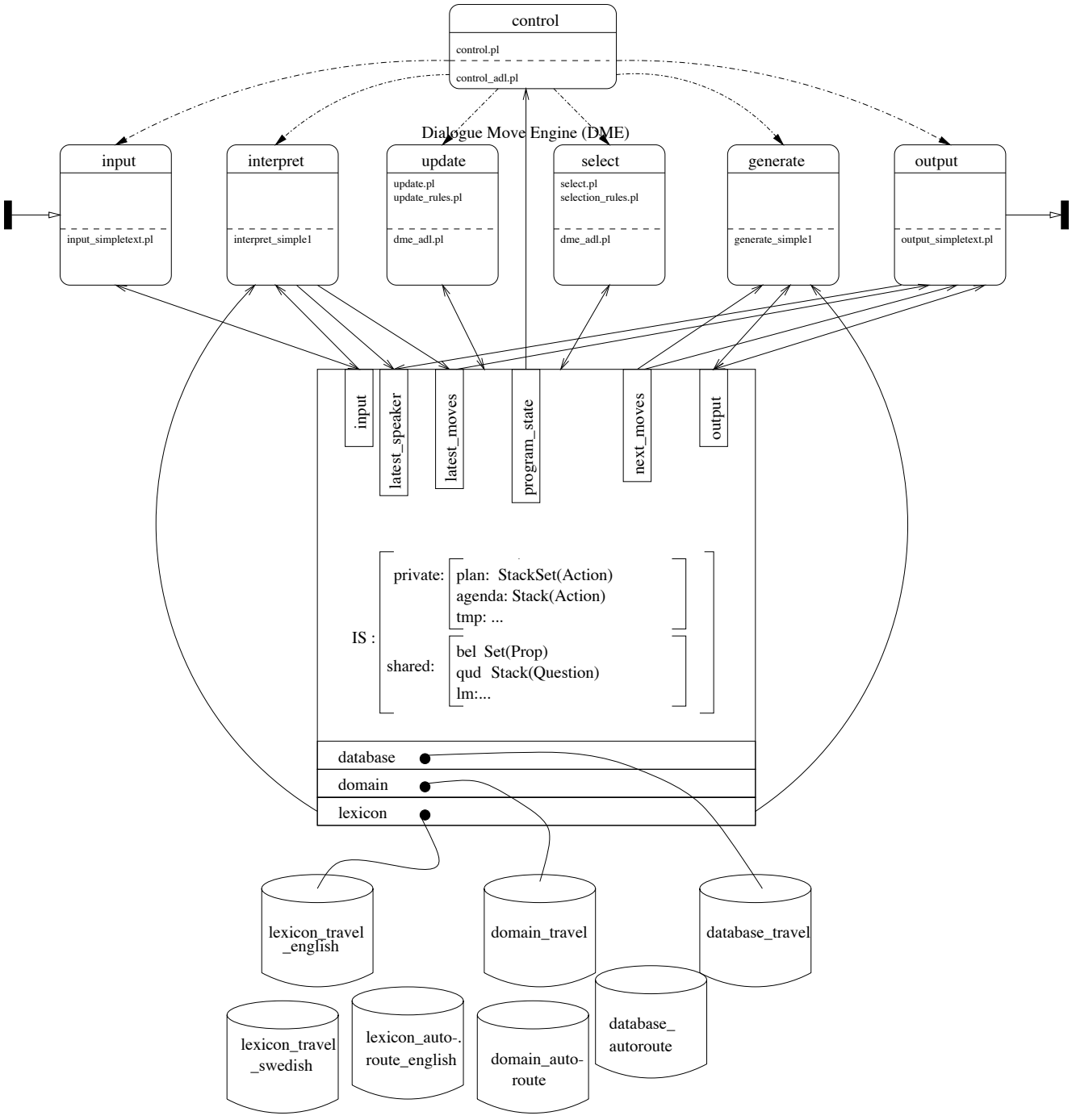


Figure 4.1: Overall structure of GoDiS

4.1.1 Information State

The main division in the information state is between information which is private to the agent and that which is shared between the dialogue participants. What we mean by shared information here is that which has been established (i.e. grounded) during the conversation, akin to what Lewis in Lewis (1979) called the "conversational scoreboard".

The private part of the information state includes a dialogue *plan*, i.e. is a list of dialogue actions that the agent wishes to carry out. The plan can be changed during the course of the conversation. For example, if a travel agent discovers that his customer wishes to get information about a flight he will adopt a plan to ask her where she wants to go, when she wants to go, what price class she wants and so on. The *agenda*, on the other hand, contains the short term goals or obligations that the agent has, i.e. what the agent is going to do next. For example, if the other dialogue participant raises a question, then the agent will normally put an action on the agenda to respond to the question. This action may or may not be in the agent's plan.

The private part of the IS also includes "temporary" shared information that has not yet been grounded, i.e. confirmed as having been understood by the other dialogue participant[1]. In this way it is easy to delete information which the agent has "optimistically" assumed to have become shared if it should turn out that the other dialogue participant does not understand or accept it. If the agent pursues a cautious rather than an optimistic strategy then information will at first only be placed in the "temporary" slot until it has been acknowledged by the other dialogue participant whereupon it can be moved to the appropriate shared field.

The (supposedly) shared part of the IS consists of three subparts. One is a set of propositions which the agent assumes for the sake of the conversation. The second is a stack of questions under discussion (QUD). These are questions that have been raised and are currently under discussion in the dialogue. The third contains information about the latest utterance (speaker, moves and integration status).

4.1.2 Module interfaces

In GoDiS, there are six module interface variables handling the interaction between modules:

- 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.

[1] In discussing grounding we will assume that there is just one other dialogue participant.

- 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: The state of GoDiS; either "run" or "quit". Read by the Control module.

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

4.1.3 Resources

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

The Lexicon resource The lexicon and its relation to the interpretation and generation modules will be described in Bos *et al.* (1999). Apart from providing relations between moves and strings, the lexicon defines the notion of a yes/no answer[2].

The Domain resource In our implementation, the domain resource includes a set of *dialogue plans* which contain information about what the system should do in order to achieve its goals. Traditionally Allen and Perrault (1980a), 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. Our approach is 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.

Typically, the system has on the agenda an action to respond to a question. However, the move for answering the question cannot be selected since the system does not yet have the necessary information to answer the question. The system then tries to find a plan which will allow it to answer the question, and this plan will typically be a list of actions to raise questions; once these questions have been raised and the user has answered them, the system can provide an answer to the initial question. This behaviour is similar to that of many natural language database interfaces, but the difference is that the architecture of our system allows us to improve the conversational behaviour of the system simply by adding some new rules, such as the accommodation rules described above. A sample plan for a travel agency domain is shown in (1).

[2]This is used to prevent accommodation of answers to yes/no answers.

```
(1)  plan( get_trip_info, [ raise(X1^(how=X1)),
                            raise(X3^(to=X3)),
                            raise(X2^(from=X2)),
                            raise(X4^(return=X4)),
                            raise(X5^(month=X5)),
                            raise(X6^(class=X6)),
                            respond(X7^(price=X7)) ] ).
```

The action which may appear in a plan are currently `raise(Q)` and `respond(Q)`, where `Q` is a question.

The domain resource also defines a concept of question-answer relevance, and a related notion of relevance of a dialogue move in relation to a goal and a plan. The former relation is not much more than a list enumerating questions and their possible answers. The latter builds on the former in the sense that a move is relevant to a task T if it gives an answer to a question Q such that an action to raise Q is a part of a plan to achieve T.

The Database resource The database is accessed by giving a list of propositions ("known facts") and a question, and returns an answer which is either a proposition or a message indicating failure to find an answer.

4.2 Formalization

4.2.1 Information State

We represent information states of dialogue participants as records of the following type:

is :

$$\left[\begin{array}{ll} \textsc{private} : & \left[\begin{array}{ll} \textsc{plan} : & \textsc{StackSet}(\textsc{Action}) \\ \textsc{agenda} : & \textsc{Stack}(\textsc{Action}) \\ \textsc{bel} : & \textsc{Set}(\textsc{Prop}) \\ \textsc{tmp} : & \left[\begin{array}{ll} \textsc{bel} : & \textsc{Set}(\textsc{Prop}) \\ \textsc{qud} : & \textsc{Stack}(\textsc{Question}) \\ \textsc{lu} : & \left[\begin{array}{ll} \textsc{speaker} : & \textsc{Participant} \\ \textsc{moves} : & \textsc{assocSet}(\textsc{Move},\textsc{Bool}) \end{array}\right] \end{array}\right] \end{array}\right] \\ \textsc{shared} : & \left[\begin{array}{ll} \textsc{bel} : & \textsc{Set}(\textsc{Prop}) \\ \textsc{qud} : & \textsc{StackSet}(\textsc{Question}) \\ \textsc{lu} : & \left[\begin{array}{ll} \textsc{speaker} : & \textsc{Participant} \\ \textsc{moves} : & \textsc{assocSet}(\textsc{Move},\textsc{Bool}) \end{array}\right] \end{array}\right] \end{array}\right]$$

As any abstract datatype, this type of information state is associated with various conditions and operations which can be used to check and update the information state. For example, fstRec(SHARED.QUD,Q) succeeds if Q is unifiable with the topmost element on the shared QUD stack, and popRec(SHARED.QUD) will pop the topmost element off the stack.

The PLAN and AGENDA fields contain the dialogue plan (a stackset[3] of actions) and the short term goals or obligations (a stack of actions), respectively. The TMP field (a record) is the "temporary" slot mirroring the shared field. The SHARED field is a record divided into three subfields: BEL, the set of shared propositions which the agent assumes for the sake of the conversation, QUD, a stack of questions under discussion and LU, which is a record containing information about the latest utterance. The MOVES subfield is an association set, where each move is associated with a boolean indicating whether (the effects of the) the move has been integrated or not.

4.2.2 Interface variables

The GoDiS 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

TYPE: PARTICIPANT
OBJECTS: { **Usr**, **Sys** }

TYPE: PROGRAMSTATE
OBJECTS: { **Run**, **Quit** }

TYPE: MOVE
OBJECTS: { **ask(Question)**, **answer(Proposition)**, **requestRepeat**, **repeat(Move)**, **greet**, **quit** }

[3] A "stackset" is a data-structure which is a stack with the additional membership condition **in** and the additional operation delete which removes an element from the stackset even if it is not at the top.

4.2.3 Resources

There are three resource interface variables, with corresponding types.

- lexicon : LEXICON
- domain : DOMAIN
- database : DATABASE

All resources in GoDiS are static, which means that there are no operations available for objects of these types.

TYPE: LEXICON
COND: { input_form(+Phrase, -Move)
output_form(-Phrase, +Move)
yn_answer(?A)
OP: {

TYPE: DOMAIN
COND: { relevant_to_task(+Moves, -Task, -Plan)
relevant_answer(-Query,+Answer)
plan(+Goal, -Plan)
OP: {

TYPE: DATABASE
COND: { consultDB(+PropList, +Question, -Answer)
OP: {

4.3 Dialogue moves

The following dialogue moves are used in GoDiS:

- **ask**(Q), where Q is a question
- **answer**(A), where A is a (possibly partial[4]) proposition
- **requestRepeat**
- **repeat**(M), where M is a move
- **greet**

[4] This may happen if an answer is elliptical.

- **quit**

Of these, **ask** and **answer** are the most prominent, which is natural given that GoDiS is intended for information-seeking dialogue and is based on the idea of Questions Under Discussion. The **requestRepeat** and **repeat** moves facilitate some minimal grounding behaviour; specifically, the possibility of "backtracking" when the optimistic assumption goes wrong. The final two move types are conventional moves, expressed stereotypically as "welcome" and "bye", respectively.

4.4 Rules

Six classes of rules are used by the Update algorithm:

- **Grounding**
- **Integrate**
- **Accommodate**
- **Refill**
- **Database**
- **Store**

In addition to these, there is one class **Select** which is used by the select module[5].

4.4.1 Grounding and Store rules

Optimism means that participants assume that their contributions have been understood and entered in both participants' common beliefs or QUDs as soon as they have been uttered. As soon as a participant A has uttered something as a response to a question, she enters the response into the shared beliefs. As soon as A raises a question, the question is entered into QUD. A cautious strategy would wait until there is some kind of feedback (which may involve simply continuing with another relevant utterance) before entering the common information. If optimistic grounding fails, for example because there is a clarification request, the system has to retract. A consequence of this is that you need to keep information from previous turn(s) in order to reinstate previous information.

[5]While this has not yet been done, one could arrange the rule classes in a hierarchy where update and select are daughters of the root note, and update has seven daughters (one for each rule class used by the update module).

The rule for optimistically assuming that a system move is grounded is shown in (2). It creates an association set where each move in latest_moves is associated with a boolean flag indicating whether the move has been integrated, with value "false".

(2) RULE: **assumeSysMovesGrounded**
CLASS: **grounding**
PRE: { latest_speaker $== sys
! (latest_moves $= *MoveSet*)
EFF: { setRec(SHARED.LU.SPEAKER, sys)
moveSet2MoveAssocSet(*MoveSet*, false, *MoveAssocSet*)
setRec(SHARED.LU.MOVES, *MoveAssocSet*)

The macro moveSet2MoveAssocSet is a macro which takes a set of moves and outputs a association set where each move is associated with a boolean (in this case with value "false").

The rule for storing the current SHARED field in TMP is shown in (3).

(3) RULE: **saveShared**
CLASS: **store**
PRE: { latest_speaker $= sys
EFF: { copyRec(SHARED, PRIVATE.TMP)

4.4.2 Integration rules

The purpose of the integration rules is to integrate (the effects of) a move into the information state. These rules may look different depending on whether the user or the system itself was the agent of the move. As an illustration, in (4) we see the update rule for integrating an **answer** move when performed by the user, and in (5) the converse rule for the case when the latest move was performed by the system. When the system gives an answer it does not need to be checked for relevance, nor does the system give elliptical answer. Note that the system optimistically assumes that the user accepts the resulting proposition P by adding it to SHARED.BEL.

(4) RULE: **integrateUsrAnswer**
CLASS: **integrate**
PRE: { valRec(SHARED.LU.SPEAKER, usr)
assocRec(SHARED.LU.MOVES, answer(R), false)
fstRec(SHARED.QUD, Q)
domain :: relevant_answer(Q, R)
reduce(Q, R, P)
EFF: { popRec(SHARED.QUD)
addRec(SHARED.BEL, P)
setAssocRec(SHARED.LU.MOVES, answer(R), true)

(5) RULE: **integrateSysAnswer**
CLASS: **integrate**
PRE: { valRec(SHARED.LU.SPEAKER, sys)
assocRec(SHARED.LU.MOVES, answer(P), false)
EFF: { popRec(SHARED.QUD)
addRec(SHARED.BEL, P)
setAssocRec(SHARED.LU.MOVES, answer(P), true)).

4.4.3 Accommodation rules

Accommodating a question onto QUD

Dialogue participants can address questions that have not been explicitly raised in the dialogue. However, it is important that a question be available to the agent who is to interpret it because the utterance may be elliptical. Here is an example from a travel agency dialogue[6]:

(6) `$J: vicken månad ska du åka`
(what month do you want to go)
`$P: 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)

The strategy we adopt for interpreting elliptical utterances is to think of them as short answers (in the sense of Ginzburg Ginzburg (1998)) to questions on QUD. A suitable question here is *What kind of price does P want for the ticket?*. This question is not under discussion at the point when P says "as cheap as possible". But it can be figured out since J knows that this is a relevant question. In fact it will be a question which J has as an action in his plan to raise. On our analysis it is this fact which enables A to interpret the ellipsis. He finds the matching question on his plan, accommodates by placing it on QUD and then continues with the integration of the information expressed by *as cheap as possible* as normal. Note that if such a question is not available then the ellipsis cannot be interpreted as in the dialogue in (7).

(7) A. What time are you coming to pick up Maria?
B. Around 6 p.m. As cheap as possible.

This dialogue is incoherent if what is being discussed is when the child Maria is going to be picked up from her friend's house (at least under standard dialogue plans that we might have for such a conversation).

[6]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.

Update rules can be used for other purposes than integrating the latest move. For example, one can provide update rules which accommodate questions and plans. One possible formalization of the **accommodateQuestion** move is given in (8). When interpreting the latest utterance by the other participant, the system makes the assumption that it was a **reply** move with content A. This assumption requires accommodating some question Q such that A is a relevant answer to Q. The check operator "answer-to(A, Q)" is true if A is a relevant answer to Q given the current information state, according to some (possibly domain-dependent) definition of question-answer relevance.

(8) RULE: **accommodateQuestion**
CLASS: **accommodate**
PRE:
- valRec(SHARED.LU.SPEAKER, usr)
- moveInRec(SHARED.LU.MOVES, **answer**(A))
- not (lexicon :: **yn_answer**(A))
- valIntegrateFlag(**answer**(A), false)
- inRec(PRIVATE.PLAN, raise(Q))
- domain :: relevant_answer(Q, A)

EFF:
- delRec(PRIVATE.PLAN, **raise**(Q))
- pushRec(SHARED.QUD, Q)

Accommodating the dialogue task

After an initial exchange for establishing contact the first thing that P says to the travel agent in our dialogue is:

(9) `$P: flyg ti paris`
< flights to Paris >

This is again an ellipsis which on our analysis has to be interpreted as the answer to a question (two questions, actually) in order to be understandable and relevant. As no questions have been raised yet in the dialogue (apart from whether the participants have each other's attention) the travel agent cannot find the appropriate question on his plan. Furthermore, as this is the first indication of what the customer wants, the travel agent cannot have a plan with detailed questions. We assume that the travel agent has various plan types in his domain knowledge determining what kind of conversations he is able to have. Each plan is associated with a task. E.g. he is able to book trips by various modes of travel, he is able to handle complaints, book hotels, rental cars etc. What he needs to do is take the customer's utterance and try to match it against questions in his plan types in his domain knowledge. When he finds a suitable match he will accommodate the corresponding task, thereby providing a plan to ask relevant question for flights, e.g. when to travel?, what date? etc. Once he has accommodated this task and retrieved the plan he can proceed as in the previous example. That is, he can accommodate the QUD with the relevant question and proceed with the interpretation of ellipsis in the normal fashion.

This example is interesting for a couple of reasons. It provides us with an example of "recursive" accommodation. The QUD needs to be accommodated, but in order to do this the

dialogue task needs to be accommodated and the plan retrieved. The other interesting aspect of this is that accommodating the dialogue task in this way actually serves to drive the dialogue forward. That is, the mechanism by which the agent interprets this ellipsis, gives him a plan for a substantial part of the rest of the dialogue. This is a way of capturing the intuition that saying *flights to Paris* to a travel agent immediately makes a number of questions become relevant.

Task accommodation and plan retrieval is taken care of by two rules: **accommodateTask** and **retrievePlan**, respectively. The first matches the move(s) in SHARED.LU.MOVES with a task to which it is relevant and stores this task in SHARED.BEL, and the second puts the plan to achieve that task in the PRIVATE.PLAN field[7].

(10) RULE: **accomodateTask**

CLASS: **accommodate**

PRE:
- not(inRec(SHARED.BEL, task=_))
- valRec(SHARED.LU.SPEAKER, usr)
- ! movesInRec(SHARED.LU.MOVES, $Moves$)
- not(in($Moves$, **answer**(A)) and (lexicon :: **yn_answer**(A)))
- domain :: relevant_to_task($Moves$, $Task$, _)

EFF:
- addRec(SHARED.BEL, (task=Task))

(11) RULE: **retrievePlan**

CLASS: **accommodate**

PRE:
- emptyRec(PRIVATE.PLAN)
- inRec(SHARED.BEL, task=E)
- !(domain :: plan(E, $Plan$))

EFF:
- setRec(PRIVATE.PLAN, $Plan$)

4.4.4 Refill and Database moves

In (12), a rule for refilling the agenda with an action from the plan is shown. The condition lu_integrated is a macro which checks that all moves in SHARED.LU.MOVES have been integrated.

(12) RULE: **refillAgenda(plan)**

CLASS: **refill**

PRE:
- emptyRec(PRIVATE.AGENDA)
- fstRec(PRIVATE.PLAN, $Action$)
- lu_integrated

EFF:
- popRec(PRIVATE.PLAN)
- pushRec(PRIVATE.AGENDA, $Action$)

[7] The notion of "plan accommodation", which has appeared in earlier GoDiS work, has been replaced by task accommodation and plan retrieval. The reason is that the traditional notion of accommodation implies that something is added to the shared information of dialogue participants; since the plan in GoDiS is assumed to be private, it cannot be subject to accommodation.

The database query rule (there is only one) is shown in (13). If there is an action on the agenda to respond to a question, and there is no answer in the private belief set, the database is consulted and the result is stored in the private belief set.

(13) RULE: **queryDB**(Q, R)
CLASS: **database**
PRE: { fstRec(PRIVATE.AGENDA, **respond**(Q))
valRec(SHARED.BEL, $SharedBel$)
not(inRec(PRIVATE.BEL, R) and domain :: relevant_answer(Q, R))
EFF: { ! (database :: consultDB(Q, $SharedBel$, R))
addRec(PRIVATE.BEL, R)

4.4.5 Select rules

A sample **select** rule is shown in (14). Most **select** moves have the same simple structure.

(14) RULE: **select(ask)**
CLASS: **select**
PRE: { fstRec(PRIVATE.AGENDA, **raise**(Q))
EFF: { set(next_moves, **ask**(Q))

4.5 Algorithms

The algorithms in GoDiS are defined using the TRINIDKIT Algorithm Definition Language (ADL). There are two variants: DME-ADL, which uses rule types to trigger update rules, and Control-ADL, which calls modules. ADL is further described in the TRINDIKIT manual (D2.2).

4.5.1 Control algorithm

```
repeat ( [reset,
          repeat (  [ select,
                      generate,
                      output,
                      update,
                      print_state,
                      test( program_state:val(run) ),
                      input,
                      interpret,
                      print_state,
                      update,
                      print_state
```

```
                ] )
            ] )
```

4.5.2 Update algorithm

```
if (latest_moves $== failed)
   then (repeat refill)
   else ( [ grounding,
            repeat ( integrate or accommodate ),
            if (latest_speaker $== usr)
               then [ repeat refill,
                      try database ]
               else store ] ) ).
```

This algorithm can be read as follows: If move interpretation failed, then refill the agenda. Otherwise, start by taking care of grounding issues. Then, integrate the latest utterance (possibly using accommodation) as much as possible. If the latest utterance was performed by the user, refill the agenda and do a database search[8] if necessary (e.g. if the first item on the agenda is to answer a question to which there is currently no answer known). If the latest move was performed by the system, store the current SHARED record in case backtracking should become necessary (i.e. in case the optimistic grounding assumption should prove wrong).

4.5.3 Selection algorithm

The selection algorithm uses a single rule class **select**, and the "algorithm" is to apply a rule of that type. In effect, it takes the first **select** rule it can find whose preconditions are fulfilled and applies it. All current **select** rules set next_moves to be a singleton set of moves (i.e. a single move).

4.6 Discussion

To give a more concrete impression of the behaviour that results from the rules and algorithms described above, we will show an example of how GoDiS can handle accommodation of questions and plans. The example is actual (typed) dialogues with the system, supplemented with information about dialogue moves, rule applications, and (partial) illustrations of the systems dynamic information state at different stages of the dialogue[9].

[8] The difference between "try α" and "α" is that if the former fails, there is no error; if the latter fails there is an error.

[9] The illustrations have been simplified slightly for easier reading.

```
# select(greet)
    > set(next_moves,set([greet]))

$S> Welcome to the travel agency!

$U> flights to paris

:  is =        private =   agenda =   < raise(_54350^(task=_54350)) >
:                          plan =     [  ]
:                          bel =      {  }
:                          tmp =       bel =      {  }
:                                      qud =      <  >
:                                      lu =        speaker =  sys
:                                                  moves =    { (greet,true) }
:              shared =    bel =      {  }
:                          qud =      <  >
:                          lu =        speaker =  sys
:                                      moves =    { (greet,true) }

# assumeUsrMovesGrounded
    > setRec(shared^lu^speaker,usr)
    > moveSet2moveAssocSet(set([answer(plane),answer(to=paris)]),false,B)
    > setRec(shared^lu^moves,B)
# accomodateTask
    > addRec(shared^bel,task=get_trip_info)
# retrievePlan
    > setRec(private^plan,stackset([raise(A^(how=A)),
                                    raise(B^(to=B)),
                                    raise(C^(from=C)),
                                    raise(D^(return=D)),
                                    raise(E^(month=E)),
                                    raise(F^(class=F)),
                                    respond(G^(price=G))]))
# accommodateQuestion
    > delRec(private^plan,raise(A^(how=A)))
    > pushRec(shared^qud,A^(how=A))
# integrateUsrAnswer
    > popRec(shared^qud)
    > addRec(shared^bel,how=plane)
    > setIntegrateFlag(answer(plane),true)
# accommodateQuestion
    > delRec(private^plan,raise(A^(to=A)))
    > pushRec(shared^qud,A^(to=A))
# integrateUsrAnswer
    > popRec(shared^qud)
    > addRec(shared^bel,to=paris)
    > setIntegrateFlag(answer(to=paris),true)
# downdateAgenda(raise)
    > popRec(private^agenda)
# refillAgenda(plan)
    > popRec(private^plan)
    > pushRec(private^agenda,raise(A^(from=A)))

:  is =        private =   agenda =   < raise(_150042^(from=_150042)) >
:                          plan =     [ raise(_150025^(return=_150025)),
                                       raise(_150015^(month=_150015)),
                                       raise(_150005^(class=_150005)),
```

```
                                    respond(_149995^(price=_149995)) ]
:                          bel =      {  }
:                          tmp =       bel =      {  }
:                                      qud =      <  >
:                                      lu =        speaker =  sys
:                                                  moves =    { (greet,true) }
:              shared =    bel =      { to=paris, how=plane, task=get_trip_info }
:                          qud =      <  >
:                          lu =        speaker =  usr
:                                      moves = { (answer(to=paris), true),
                                                 (answer(plane), true) }

# select(ask(A^(from=A)))
    > set(next_moves,set([ask(A^(from=A))]))

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

After interpreting the users utterance as two **answer** moves with respective contents (how=plane) and (to=paris), the update module is called. After applying a **grounding** rule to ground the user's utterance, the update algorithm tries to find an **integrate** or accommodate rule. Following the ordering of the rules given in the list of rule definitions, it first checks if it can perform **integrateUsrAnswer**. However, this rule requires that the content of the answer must be relevant to the topmost question on QUD. Since the QUD is empty, the rule does not apply. It then tries to apply the **accommodateQuestion** rule, but since the plan is empty this rule does not apply either. However, **accommodateTask** [10] does apply, since there is (in the domain resource) a plan such that the latest move is relevant to that plan. More precisely, the latest move provides an answer to a question Q such that raising Q is part of the plan. . As a consequence of task accommodation, the preconditions of the **retrievePlan** rule will be true and that rule will trigger and put the plan to achieve the accommodated task in the PRIVATE.PLAN field[11].

Once this rule has been executed, the update algorithm tries again to find an **integrate** or accommodate rule. This time, it turns out the preconditions of **accommodateQuestion** hold, so the rule is applied. As a consequence of this, the preconditions of **integrateUsrAnswer** now hold, so that rule is applied. Actually, it turns out that a move performed in the latest utterance is also relevant to a second question (concerning the destination) in the plan, so that question is also accommodated and its answer integrated. Since no additional **integrate** or accommodate rules apply, the update module refills the agenda from the plan and stops, allowing the control module to call the select module which selects the next move on the agenda: asking where the user wants to travel from.

[10] In the case where a move is relevant to several plans, this rule will simply take the first one it finds. This clearly needs further work.

[11] This requires that there is only one plan for each task. This assumption is feasible if one allows conditional, or "algorihmic" plans, which can result in different action sequences depending e.g. on how the information state is changed by future dialogue.

Chapter 5

Instantiation II: Focus on Obligations and Grounding

The second model of information states is based on the dialogue model of Poesio and Traum Poesio and Traum (1997, 1998). This work can be roughly characterized by emphasis on the following concerns:

1. Building upon previous work by Traum (1994), is a central focus on the GROUNDING process, by which common ground is established (Clark and Schaefer, 1989; Traum and Hinkelman, 1992). Poesio and Traum view the public information state as including both the material that has already been grounded, indicated as G here, and the material that has been introduced but hasn't yet been grounded; the ungrounded part consists of a specification of the current 'contributions,' or DISCOURSE UNITS, as they are called in (Traum and Hinkelman, 1992).

2. a view of the information state as, first and foremost, a record of the dialogue acts that take place during a conversation.

3. an orientation toward *incremental processing*, which include updates closer to the level of words, rather than full propositional or sentence level contributions

4. a focus on the *social* effects of dialogue acts, particularly obligations and commitments, rather than the more traditional mental states of belief and intention (the latter are deducible only as default inferences, using additional rationality and sincerity assumptions).

5. an exploration of the accessibility conditions of pragmatic processes, such as reference and scoping, within the context of multi-party dialogue rather than monologic discourse.

The information state (for each participant) consists of the following three main parts:

- A private part, with information available to the participant, but not introduced into the dialogue. This part consists of private beliefs and intentions, as well as potentially hypotheses about the beliefs and intentions of other agents
- A public part, with information assumed to be part of the common ground (called G). This will contain obligations and options, social commitments to propositions, as well as facts about which dialogue acts have occurred.
- A semi-public part, with information introduced into the dialogue, but not yet grounded, and not assumed (yet) to be part of the common ground, yet still accessible. This part is divided into a set of DISCOURSE UNITS (DUs), which represent packets of information that can be added to common ground together. In addition, there is a structure of these DUs, UDUs, representing the accessibility of these units. Information within each DU will be of the same sort as within G.

Within the TRINDI Project, we have been extending this work in several ways, as described in the next three sections of this chapter:

1. Extension of the formalization based on compositional DRT (Muskens, 1995), directly extending and improving the preliminary work in Poesio and Traum (1997, 1998).
2. a formalization sharing many of the aspects of the theory in the previous chapter, using records to represent the information state, as well as G and the DUs, with lists representing the other components (intentions, UDUs, dialogue acts, obligations, etc.). This formalization was described in detail in Cooper *et al.* (1999). We have used this formalization for coding information states, and are currently building a DME library for this formalization. In this document we will only touch briefly on this aspect, referring the reader to Cooper *et al.* (1999) for more details.
3. a formalization using classical DRT, with heavy emphasis on first-order theorem proving. This is implemented as a DME library, comprising the system MIDAS, described in more detail in Bos *et al.* (1999).

5.1 Formalization 1

We first present a more discursive overview of formalization 1, building directly on the work presented in Poesio and Traum (1997, 1998).

Grounded and Ungrounded Information

Poesio and Traum propose that the information state of each conversational participant CP at any given time has the structure in (5.1). Poesio and Traum view the conversational

information state as a DRS which specifies information about G and the discourse units; this DRS gets updated over time as a result of dialogue acts. G and the discourse units are also DRSs; we are shortly going to see what kind of information they contain. There are two reasons for these decisions: first of all, the grounding acts refer back to the DUs, as we will see shortly; and second, the modifications to G and to the discourse units can be easily modeled as modifications to discourse markers in an extended version of Compositional DRT (Muskens, 1995) with markers denoting DRSs. The CIS also contains information about the currently pending discourse units, which are put together in a list UDUS. The top of UDUs is the Current Discourse Unit CDU– the Discourse Unit to which new material gets added. We write below CDU for **first**(*UDUS*).

(5.1)

G UDU CDU DU1 DU2 DU3
G = ...
DU1 = ...
DU2 = ...
DU3 = ...
UDUS = $\langle DU3, DU1 \rangle$
(CDU = **first**(*UDUS*) = DU3)

The picture of dialogue assumed by Poesio and Traum is one in which each act leads to an update of the CIS. All new information gets first added to a DU; this results in obligations of various types and possibly in the responder coming to some conclusions about the intentions of the initiator. Information moves from UDUS into G as the result of acknowledgments.

The Conversational Score as a Record of the Discourse Situation

As such, it can be characterized in terms of the language introduced in DRT to characterize other types of situations. For example, the utterances in (5.2), if interpreted as an **Assert**, and **Accept**, and an **Assert**, respectively, result in the conversational participants sharing the information in (5.3), that includes a record of the occurrence of three locutionary acts and three core speech acts generated by them (we have omitted from (5.3) all information about smaller locutionary acts such as the uttering of *there*):

(5.2) a. A: There is an engine at Avon.

b. B: Okay.

c. A: *It* is hooked to a boxcar.

(5.3) G=

u1-6 u7 u8-13 ce1 ce3 s s′ s″ K1 K2

---

u1-6 : **Utter**(*A*, *"There is an engine at Avon"*)
ce1 : **Assert**(*A*,*B*,*K1*)

K1 =
x w e
engine(*x*)
Avon(*w*)
e : [**at**(*x*,*w*)]

K1(*s*)(*s′*)
generate(*u1-6*,*ce1*)
u7 : **Utter**(*B*, *"Okay"*)
ce3 : **Accept**(*B*,*ce1*)
generate(*u7*,*ce3*)
u8-13 : **Utter**(*A*, *"It is hooked to a boxcar"*)
ce4 : **Assert**(*A*,*B*,*K2*)

K2 =
y u e′
boxcar(*y*)
e′ : [**hook**(*y*,*u*)]
u **is** *x*

K2(*s′*)(*s″*)
generate(*u8-13*,*ce4*)
satisfaction-precedes(*ce*,*ce4*)

This hypothesis about the conversational score plays two important roles in what follows. First of all, we can assume that agents can reason about the occurrence of dialogue acts and draw some conclusions; most of the updates we will see below are originated by observations of this type. Secondly, we can assume that agents can refer back to dialogue acts just like they do with other events; in this way we can handle the implicit anaphoric reference to events in backward-looking acts.

Compositional DRT

Poesio and Traum's formalization is based on DRT–most specifically, on Muskens' formulation of DRT in terms of the theory of types, Compositional DRT (Muskens, 1995). The crucial properties of CDRT to understand what follows are that assignments are treated as first-class objects– of type *s*–and that discourse entities are viewed as functions from assignments to entities in the domain. DRSs can then be defined as relations between assignments, i.e., objects of type $\langle s,\langle s,t\rangle\rangle$: the DRS $[u_1, \ldots, u_n|\varphi_1, \ldots \varphi_m]$ is defined as follows:

$$[\mathrm{u}_1, \ldots, \mathrm{u}_n \mid \varphi_1, \ldots, \varphi_m] = \lambda\, i.\quad \lambda\, j.\quad i[\mathrm{u}_1, \ldots, \mathrm{u}_n]j \wedge\ \ \varphi_1(j), \ldots, \varphi_m(j)$$

where $i[u]j$, the UPDATE OPERATOR, is short for (simplifying somewhat):

- $\forall v\ (u \neq v) \rightarrow (v(i) = v(j))$

For example, the DRS which is the value of the marker K1 in (5.3) has the following value:

(5.4) $[\![[x\ w\ e|\mathbf{engine}(x),\mathbf{Avon}(w),\ e : \mathbf{at}(x,w)]\ ]\!] = \{\langle$i,j$\rangle$ | i and j are states, j differs from i at most over x, w and e, and the values assigned by j to x , w and e satisfy $[\![\mathbf{engine}(x)]\!]$,$[\![\ \mathbf{Avon}(w)]\!]$, and $[\![e : \mathbf{at}(x,w)]\!]\}$

Poesio and Muskens (1997) proposed to extend the standard version of CDRT in order to allow for discourse markers of two new types: ranging over assignments, and ranging over relations between assignments (DRSs).

5.1.1 The Update Effects of Dialogue Acts

Cohen and Levesque (1990b) argued that illocutionary acts are not an essential ingredient of a theory of communication; they can be 'defined away' by capturing their effect in terms of intentions and beliefs. Our goals here are more modest. The axiomatization of dialogue acts that we propose below specifies for each dialogue act the update to the conversational score that results when an occurrence of that act is recorded; e.g., what gets grounded as the result of an acknowledgment, or the attitudes that become public (i.e., the corresponding states are recorded in G) as a result of a core speech act. However, we feel it is too early to claim that the update properties we specify completely define the DRI dialogue acts, and that therefore these can be dispensed with.

The update effects are specified using the format:

Name:	**Act**
Condition on update:	Φ
Update:	Ψ

In the simplest cases, the update simply depends on the occurrence of the dialogue act being recorded in one of the DRS that constitute the CIS; in more complex cases, additional conditions on the CIS are involved. The update condition may also depend on the condition holding in a specific DRS among those that constitute the CIS: e.g., the update resulting from a core speech act being added to a discourse unit are typically different from that that results from that speech act being added to G. We use the shorthand K::φ to specify that condition φ must hold in DRS K, meaning:

$$\text{K::}\varphi =_{def} \forall ijK(i)(j) \rightarrow \varphi(j)$$

We use the notation X += K to indicate the operation of DRS update in which the value of X is updated by concatenating K to it by means of the CDRT ; operator:

$$\text{X} +\!\!= \text{K} =_{def} \lambda i \lambda j X(j) = (X(i); K)$$

Finally, we use two operators for doing list manipulation, **push** and **remove**, defined in turn in terms of a concatenation operator | and a deletion operator / on lists:

$$\mathbf{push}(X, Y) =_{def} \lambda i \lambda j X(j) = \langle Y | X(i) \rangle$$

$$\mathbf{remove}(X, Y) =_{def} \lambda i \lambda j X(j) = X(i)/Y$$

Primitives

First of all, a brief introduction to the terminology we use to talk about events and types. We use the term EVENTUALITY TYPE to refer to abstracts over conditions describing events or states of type $\langle \epsilon, \langle s, t \rangle \rangle$, such as $\lambda e.\lambda i.e(i) : \mathbf{Accept}(x(i), e'(i))$ where **Accept** is an event type, x, e and e' are discourse markers[1] and e' is the event being accepted; or $\lambda e.\lambda i.e(i) : \mathbf{Bel}(x(i), K(i))$, where **Bel** is a state type and K a discourse marker taking values over DRSs. We also refer to event types as ACTION TYPES. We use the symbol α to refer to action types, and the symbol σ to refer to state types.

Our characterization of the effects of DAs on the CIS makes use, first of all, of the event types **Try**, **Achieve** and **Address**, informally described as follows:

- e : **Try**(A,α) means that e is an event of A trying to perform an act of type α .[2]
- e : **Achieve**(A,σ) means that e is an event of A bringing about the satisfaction of state type σ.
- e : **Address**(A,e') means that e is an event of A considering and responding to e$'$.

Secondly, we assume that the conversational score can include information about agents being in the state having one of the following mental attitudes:

- s : **Bel**(A,K): s is a state of agent A believing the proposition expressed by DRS K.
- s : **Int**(A,τ), where τ is either an action type that agent A intends to perform or a state type that A intends to achieve.
- s : **Option**(A,α): action type α is one that A is aware that she can perform.

[1] We recall that in CDRT discourse markers are functions from assignments to objects in the domain.

[2] **Try** expresses the notion of present-directed intention (Cohen and Levesque, 1990a, pg. 35) and is related to Cohen and Levesque's **ATTEMPT**.

In addition to 'private' attitudes such as **Bel** and **Int**, which are traditional ingredients of formalizations of speech acts (Allen, 1983; Cohen and Levesque, 1990b), our formalization also relies on some *social* attitudes, which relate an agent not only to a course of events or action, but also to a social group. These include:

- s : **SCP**(A,B,K): this stands for Socially Committed to a Proposition. It is the public counterpart to individual belief. It means that A is committed to B to K being the case (whether or not she actually privately believes it).[3]
- s : **Obliged**(A,B,α) state s is one of A having the obligation to B to perform an act of type α(whether or not she actually intends to) Traum and Allen (1994).

Typically these states cease to hold after a while, either because e.g., the obligation has been addressed, or because an intention has been dropped. Current states are those whose associated time interval $\varphi(s)$ properly contains the indexical time point **now** (**now** $\subseteq \varphi(s)$, in Muskens' notation); some of the acts below update the temporal duration of some of these states making them not current anymore.

In general, an agent's obligations are satisfied when that agent performs the action that (s)he is obliged to do:

$$\forall i, s, s', e', A, B, \overline{x}[s : \textbf{Obliged}(A,B,\lambda s'.P(A,\overline{x}))](i) \wedge [e : \mathrm{P}(\mathrm{A},\overline{x})](i) \rightarrow [\neg \exists t t \subseteq s \wedge e < t]$$

We assume that obligations are ranked by priority, and that is a partial order, more or less in the way that questions are ranked in QUD in Ginzburg's model. We will write $s \prec s'$ to indicate that obligation s precedes obligation s'.

Locutionary Acts

As a new utterance is perceived, the current discourse unit is updated with the corresponding locutionary act. This update rule specifies a sort of default co-presence assumption - everything that gets uttered is by default recorded as part of the conversational score. This case differs from the others in that there are no update conditions–the act is not recorded anywhere in the CIS prior to this update. u is a new discourse marker.

Name:	**Utter**
Condition on update:	
Update:	CDU += $[u \mid u : \textbf{Utter}(A, \text{“...”})]$

[3] A default inference can generally be drawn in the case of an honest agent between SCP and actual belief, as follows:

$$\forall \mathrm{a,b,K,s,i} \quad \boxed{s : \textbf{SCP}(a,b,K)}(\mathrm{i}) \Rightarrow \exists\ \mathrm{s'}\ \boxed{s' : \textbf{Bel}(a,K)}(\mathrm{i})$$

For example, as soon as the first word in (5.2a), *There*, is perceived, the update to the CDU in (5.5a) takes place; assuming that the initial constituents of the CIS are empty, and the initial CDU is DU1, the result is the CIS in (5.5b), where *u1* is a new discourse marker. The same update takes place after each locutionary act.[4]

(5.5) a. CDU += $[u1|u1 : \mathbf{Utter}(A, \text{“}There\text{”})]$

b.

G UDUS CDU DU1
G = [|]   DU1 = $[u1\|u1 : \mathbf{Utter}(A, \text{“}There\text{”})]$ UDUS = $\langle DU1 \rangle$ (CDU = DU1)

Core Speech Acts

A fundamental property of forward looking acts is that they impose an obligation on the responder to perform an **Understanding-act** (e.g., acknowledge them) when she recognizes their occurrence. Let F be any forward looking acts, with arguments $A, B, \overline{x}$; then the occurrence of an action of that type in K (a DU) results in the following update :

Name:	**F**
Condition on update:	K::$[e :$ $\mathbf{F}(A,B,\overline{x})$]
Update:	G += $[s\|s :$ **Obliged**$(B,\lambda s'.s' :$ **Understanding-act**$(B, K))]$

Some forward-looking actions also impose an obligation on the responder to address them. This is certainly the case for **directives** – arguably, it holds for **Statement**s and **Offer**s, as well. Let D be a forward-looking action of this class, with arguments $A, B, \overline{x}$; then its occurrence in K (DU or G) results in the following update:

Name:	**D**
Condition on update:	K::$[e :$ $\mathbf{D}(A,B,\overline{x})$]
Update:	K += $[s\|s :$ **Obliged**$(B,\lambda s'.s' :$ **Address**$(B, e))]$

The specific update effects for some of the forward-looking acts are shown in Table 5.1. These formalizations are fairly direct implementations of the specifications in (Allen and Core, 1997). As mentioned above, we assume that the occurrence of an act such as **Assert** that specializes another act (**Statement**) results in the updates associated both with the more general and with the more specific act.

What distinguishes an assertion from a garden variety statement is the intention to get the responder to believe the claim (one could make a statement in the case where one knows

[4] This is actually a simplification, in reality one often can't tell to which DUs various parts of an input utterance will belong. In an extended version of this paper, we will give more details on how to handle this kind of update using **continue** grounding acts to merge new input with existing DUs. For now, the assumption that all new material from a current utterance gets put into the CDU will suffice.

Name:	**Statement**
Condition on update:	G::[e : **Statement**(A,B,K)]
Update:	G += [s|s : **SCP**(A,B,K)]
Name:	**Assert**
Condition on update:	G::[e : **Assert**(A,B,K)]
Update:	G += [e'|e' : **Try**(A,$\lambda s'.s'$: **Bel**(B,K)), [e"|e" : **Accept**(B,e)] ⇒ [s|s : **SCP**(B,A,K)]]
Name:	**Influencing-addressee-future-act**
Condition on update:	G::e : **IAFutA**(A,B,$\lambda e'.e' : \varphi$)
Update:	G+=[s|s : **Option**(B,$\lambda e'.e' : \varphi$)]
Name:	**Open-option**
Condition on update:	G::[e : **OpOp**(A,B,$\lambda e'.e' : \varphi$)]
Update:	G+=[|¬[e"|e" : **Try**(A,$\lambda s'.s'$: **Achieve**(A,$\lambda e'.e'$:φ))]]
Name:	**Directive**
Condition on update:	G::[e : **Dir**(A,B,$\lambda e'.e' : \varphi$)]
Update:	G += [| [e"|e" : **Accept**(B,e)] ⇒ [s|s : **Obliged**(B,A,$\lambda e'.e' : \varphi$)]]
Name:	**Committing-speaker-future-action**
Condition on update:	G::[e : **CSFA**(A,B,$\lambda e'.e' : \varphi$)]
Update:	G+=[s|s : **Option**(A,$\lambda e'.e' : \varphi$)]
Name:	**Commit**
Condition on update:	G::[e : **Commit**(A,B,$\lambda e'.e' : \varphi$)]
Update:	G += [s|s : **Obliged**(A,B,$\lambda e'.e' : \varphi$)]
Name:	**Offer**
Condition on update:	G::[e : **Offer**(A,B,$\lambda e'.e' : \varphi$)]
Update:	G += [| [e"|e" : **Accept**(B,e)] ⇒ [s|s : **Obliged**(A,B,$\lambda e'.e' : \varphi$)]]

Table 5.1: Forward-Looking Act Definitions

the responder won't believe it, or already believes it). But the achievement of that belief (a successful assertion) is too strong a condition, that defines the perlocutionary act of **convince**: that only results if the responder explicitly **Accepts** the act, which results in a further inference because of the conditional originated from the update. For an example of the consequences of an **Assert**, consider again (5.2a). The situation after all of the locutionary acts have been processed is as in (5.6) (we omit here all information about the locutionary acts derived from incremental syntactic and semantic interpretation):

(5.6)

G UDUS CDU DU1

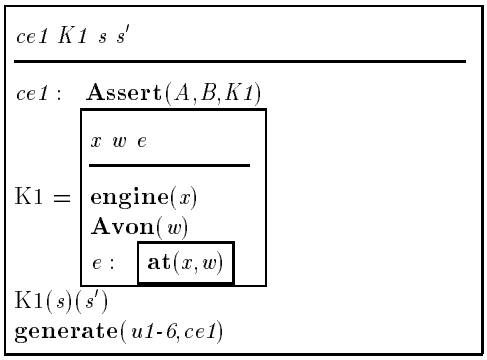
G = []

DU1 = *u1 ... u6*

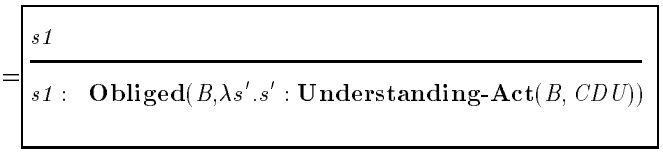
u1 : **Utter**(A, "*There*")
.....
u6 : **Utter**(A, "*Avon*")

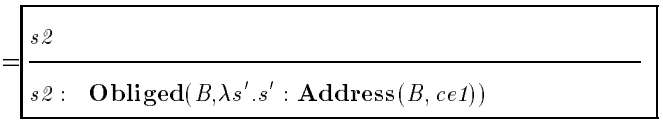
UDUS = ⟨*DU*1⟩ (CDU = DU1)

In the meantime, intention recognition takes place. Assuming that the utterance unit consisting of *u1 ... u6* is interpreted as an **Assert**, the following update of the CDU takes place (the occurrence of other acts such as a **Release-turn** and perhaps other core speech acts are also possibly inferred):

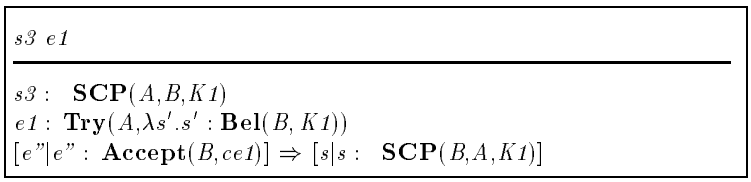
CDU +=


(we have glossed over how precisely the semantic interpretation of the utterance unit is computed - see Poesio and Traum (1997) for some details). This update results in an obligation to signal understanding or misunderstanding with respect to *ce1* and (possibly) in an obligation to address it, which result in the following updates of the conversational score:

(5.7) G+=


(5.8) CDU+=


As we mentioned earlier and we will see in more detail shortly, acknowledging a DU has the effect of updating G with the information in that DU. The utterance in (5.2b), *Okay*, has a dual purpose: it serves as an acknowledgment of *ce1*, as well as an acceptance. The acknowledgment leads to the occurrence of *ce1* being grounded, which in turn leads to the updates associated with an **Assert** act according to Table 5.1, namely:

G +=


The crucial property of backward-looking acts is that they remove the obligation to **Address** an act. For example, if **F** is a forward-looking act imposing an obligation to address, and **B** is either an **Accept** or a **Reject**, both of which are types of addressing, then performing an act *e* of type **B** with respect to the occurrence of act *e'* of type **F** removes the obligation to address the event.

The specific updates resulting from backward-looking acts are described in Table 5.2.

Name:	**Agreement**
Condition on update:	G::[e : **Agreement**(A,ce)]
Update:	_
Name:	**Accept**
Condition on update:	G::[e : **Accept**(A,ce)]
Update:	[effect specified by conditional in update for **Assert/Directive**]
Name:	**Answer**
Condition on update:	G::[e : **Answer**(A,ce)]
Update:	_
Name:	**Reject**
Condition on update:	G::[e : **Reject**(A,ce)]
Update:	[as in the case of Accept, work specified by conditional clauses in the specification of the updates for the core speech acts (not included here)]

Table 5.2: Backward-Looking Act Definitions

The DRI scheme includes a single **Accept** act that may be used to address acts of different types; we hypothesize that the act-specific consequences of acceptance are part of the definition of the forward acts themselves, in the conditionals introduced as the result of the performance of the forward acts. Thus, for example, B's acceptance of *ce1* by the *Okay* in (5.2a) leads to G being updated with the information [$s|s$: **SCP**(B,A,*K1*)].

Grounding Acts

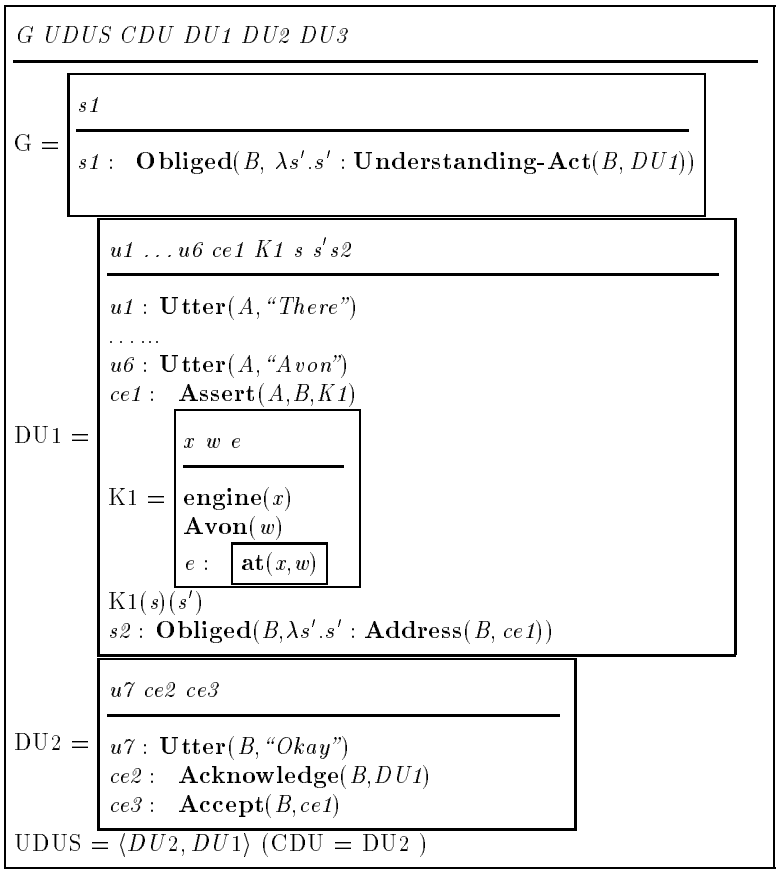
Of the grounding acts, we only consider here **Acknowledge**, that we treat as a predicate ce : **Acknowledge**(A,*DU1*) relating a CP A to a DU DU1. The occurrence of an acknowledgment of DU1 results in G being updated with that discourse unit, which is then removed from UDUS. Grounding acts do not seem to ever get added to G; we hypothesize that they are included in their own DUs that also get removed after they update the conversational score.[5]

Name:	**Acknowledge**
Condition on update:	CDU::[u : **Acknowledge**(A,*DU1*)]
Update:	G += DU1;G+=CDU

For example, we hypothesize that the *Okay* in (5.2b) works as follows. At the end of the first turn in (5.2) the CIS is as after the update in (5.7). As the turn is taken by B a new DU is initiated, DU2. (This is an effect of the implicit **Release-turn** performed at the end of (5.2)a.) The locutionary act *u7* of uttering *Okay* is added to DU2, as are the **Acknowledge** act *ce2* and the **Accept** act *ce3* as soon as they are recognized. The result is the situation in (5.9).

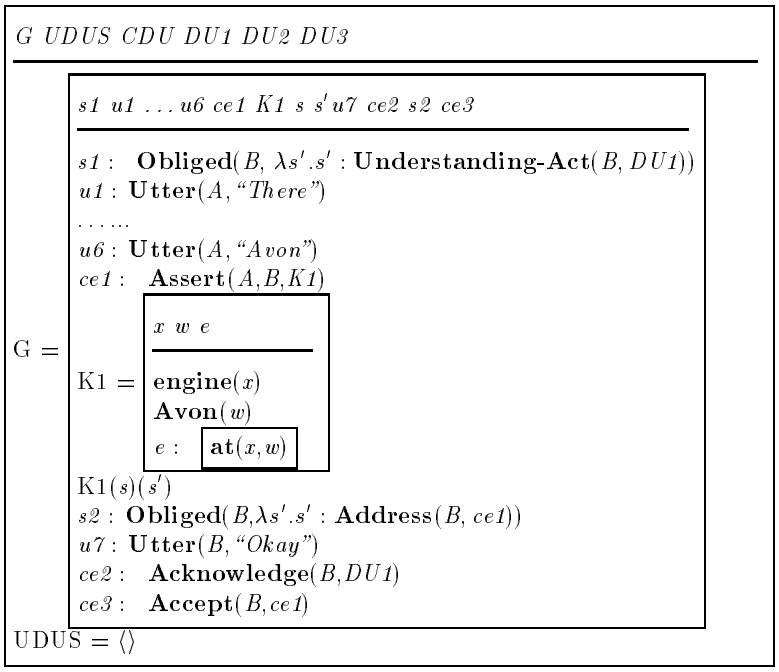
[5]The importance of grounding acts is not that they occur and are objects of discussion, but their effect on restructuring parts of CIS.

(5.9)



At this point, as a result of the acknowledgment, G is updated with DU1 as well as with DU2:

(5.10)



Adding to G the fact that an understanding act and an addressing act were performed results in the removal of the obligations *s1* and *s2*.

We will assume here for simplicity that each new turn results in a new DU being created, and that whatever is inferred from the content of that Du gets also added to the DU, except for forward looking acts that have to be acknowledged and will therefore result in new DUs. (In general, each new bit of information results in a new DU that may or may not be merged with previous DUs.)

Summary of Update Procedure

Updating the Poesio-Traum framework information states due to the performance of a new action can be conceived of as a multi-step process. While we do not advocate that it is necessary or desirable (or perhaps even possible) to proceed exactly in this fashion during actual processing, it does result in consistent new states in the theory. There should be one new DU for each grounding act, and one grounding act each for (1) operations on a previous DU or (2) initiation of new material that will have to be grounded. The updates can thus be summarized as adherence to the following procedures.

For each new utterance:

1. create a new DU, (e.g., DU1) and push this on the top of UDUs (=CDU).
2. put in CDU the locutionary act (e.g., for each utterance X by speaker A, create a new discourse marker u, and update CDU with [u|u:utter(A,X)]
3. if this new material in CDU requires grounding (currently somewhat underspecified, in lieu of further empirical evidence), AND there is a backward looking grounding act performed by the utterance (acknowledge, continue, repair, request repair, request acknowledge), create a new DU (e.g., DU2) and push it on the top of UDUs.
4. update CDU (which might now be DU1 or DU2, depending on step 2) by including the backward-looking grounding act (if any) (e.g., acknowledge), and other "backward looking" acts that will not require separate acknowledgement from the unit to which the backward-looking grounding act refers to.
5. perform updates based on the observance of the backward grounding act i.e., for ack(DU0): G+= DU0;CDU , pop(CDU), remove(DU0,UDUs)
6. put forward looking acts and backward acts that did not fall into those described in (4) (i.e., those that still need acknowledgement to be grounded) (if any) into CDU (i.e., DU1)
7. apply update rules to all DRS's which contain newly added forward and backward acts (from steps 4, 6) (note that this may also involve using the taxonomic rules from 8 to determine which acts have been performed, so, perhaps that step should come before,...)

8. apply other inference rules (e.g., modus ponens, taxonomic reasoning about subclasses)

At this point the utterance interpretation phase can be considered complete. However there is still the part about what the agent should do about it to update further mental states (particularly intentions) that would lead to the agent engaging in further action. Some of this deliberation could actually be formed as inference rules and applied in (8), as well.

9. deliberate and perform an utterance, and then begin again at (1), usually with a new agent as the speaker.

This can be instantiated with the following subrules (A is the speaker, B the addressee, CDU= top(UDUs), DU0 is some pre-existing DU, probably but not necessarily on UDUs, CE0 is some conversational event, Q0 is a previous question (info request), S0 a previous statement, K is some DRS, AT, an act type, Q is a question-object, ans is a function from question and answers to (structured?) propositions - basically the derived proposition which is the answer to the question):

1. create DU1, push(DU1,UDUs)
2. CDU += [u|u:utter(A,X)]
3. if needed, create DU2, push(DU2,UDUs)
4. if appropriate act has been performed:
   (a) CDU += [ga | ga : acknowledge(DU0)]
   (b) CDU += [acc | acc : accept(CE0)]
   (c) CDU += [rej | rej : reject(CE0)]
   (d) CDU += [ans | ans : answer(Q0,K)]
   (e) CDU += [agr | agr : agree(S0)]
   (f) CDU += [adr | adr : address(CE0)]

   5) if rule 4a performed in previous step, update on the basis of observed acknowledgement: CDU:: ack(DU0) → G+= DU0;CDU , pop(CDU) , remove(DU0,UDUs)

   6) if appropriate act has been performed:

   (a) CDU += [sa | sa : statement(A,B,K)]
   (b) CDU += [di | di : IAFutA(A,B,AT)]
   (c) CDU += [di | di : Direct(A,B,AT)]
   (d) CDU += [cs | cs : CSFA(A,B,AT)]
   (e) CDU += [co | co : Commit(A,B,AT)]
   (f) CDU += [of | of : Offer(A,B,AT)]
   (g) CDU += [qu | qu : Info-req(A,B,Q)]

(h) CDU += [ch | ch : Check(A,B,K)]

Likewise, each act should be added to DH of CDU. Also, see updates from (4), some of which would be appropriate now.

5. as appropriate depending on acts newly added to G (in 5) or CDU (in 4 or 6):

   (a) K:: e : ForwardAct(A,B,X) → G+=[s|s:Obliged(B,A,UnderstandingAct(K))]

   (b) K:: s: statement(A,B,K') → K+=[sa|s:SCP(A,B,K')]; [[e|e : Accept(B,s)] → [s'|s':SCP(B,A,K')]]

   (c) K:: e : IAFutA(A,B,AT) → K+=[o|o:Option(B,AT)]; [[e'|e' : Accept(B,e)] → [s'|s':Obliged(B,A,AT)]]

   (d) K:: di: Direct(A,B,AT) → K+=[s|s:Obliged(B,A,Address(di))]

   (e) K:: CSFA(A,B,AT) → K+=[o|o:Option(B,AT)]

   (f) K:: Commit(A,B,AT) → K+=[o|o:Obliged(A,B,AT)]

   (g) K:: o : Offer(A,B,AT) → K+= [[e|e : Accept(B,o)] → [s'|s':Obliged(B,A,AT)]]

   (h) K:: q : Info-req(A,B,Q) → K+=[s|s:Obliged(B,A,Answer(q))]

   (i) K:: q : Check(A,B,K') → K+=[s|s:Obliged(B,A,Answer(q))]; [[e|e : Agree(B,s)] → [s'|s':SCP(A,B,K')]]

   (j) K:: Agree(A,B,K') → K+=[o|o:SCP(A,B,K')]

   (k) K:: Answer(A,B,q,K') → K+=[o| o:SCP(A,B,ans(q,K'))]

6. as appropriate apply the following inference rules:

   - taxonomic rules:

     (a) e : Ack(K) → e : UnderstandingAct(K)
     (b) e : Accept(e') → e : Address(e')
     (c) e : Reject(e') → e : Address(e')
     (d) e : Accept(e') → e : BackwardAct(e')
     (e) e : Address(e') → e: BackwardAct(e')
     (f) e : Agree(e') → e : BackwardAct(e')
     (g) e : Statement(e') → e : ForwardAct(e')
     (h) e : IAFutA(e') → e : ForwardAct(e')
     (i) e : CSFA(e') → e : ForwardAct(e')
     (j) e : Info-req(e') → e : ForwardAct(e')
     (k) e : Check(e') → e : Info-req(e')
     (l) e : Direct(e') → e : IAFutA(e')
     (m) e : Commit(e') → e : CSFA(e')
     (n) e : Offer(e') → e : CSFA(e')

   - Obligation resolution:

     K :: [o : Obliged(A,B,AT), AT(A)] → K+= [o < now]

- Modus ponens:
  K :: [phi → psi, phi] → K+= [phi]
- Intention Resolution:
  K :: [i : Intend(A,AT), Bel(Done(AT(A)))] → K+= [i < now]

7. Some example Intention Updating rules:

   (a) G :: [Obliged(A, AT)], Option(AT) → Intend(At)

   (b) Intend(A, AT), Bel(Subact(AT, at1)) → Intend(at1)

   (c) Intend(A, AT), Bel(precondition(AT, pre1) → Intend(achieve(pre))

8. Action performance rule: Intend(A,AT), Turn(A), Cando(A,AT) → Perform(A,AT)

5.2 Formalization 2

In the record-based formalization, the information state of a dialogue participant is represented as in (5.11).

(5.11)
$$\begin{bmatrix} G & : & \text{PT-record} \\ DU_1 & : & \text{PT-record} \\ \dots & & \\ DU_n & : & \text{PT-record} \\ UDUs & : & \text{List} \\ INT & : & \text{List} \end{bmatrix}$$

Where PT-record is of the type shown in (5.12) (abbreviated PT-record in (5.11)). INT represents the participant's intentions, currently a prioritized list of actions. DH is an ordered list of dialogue acts that have been performed. OBL is a list of the obligations of the dialogue participants, while OPT is a list of options mentioned. SCP represents commitments to propositions (rather than obligations to perform actions).

(5.12)
$$\begin{bmatrix} DH & : & \text{List} \\ OBL & : & \text{List} \\ SCP & : & \text{List} \\ OPT & : & \text{List} \end{bmatrix}$$

The set of dialogue moves used is the same set of augmented DRI/Damsl moves described in Cooper *et al.* (1999), and above in section 5.1.

Update rules follow the guidelines for update described in Cooper *et al.* (1999). In the current implementation there are three types of update rules, the first to do with DU creation, the second dealing with backward grounding function, and the third handling updates based on

observation of other acts (there are also some other update rules for inference and other minor operations).

A couple of examples of a rules of the first type are the following:

```
urule( makeNewDU, ruletype1,
       [ w^cdu^id:is_set,
         valRec(w^cdu,CDU) ],
       [ setRec(w^pdu,CDU) ]
     ).

urule( makeNewDU, ruletype1,
       [ latest_moves: not( empty ),
         next_du_id:val(ID)],
       [ setRec(w^cdu, record([tognd=record([scp=stack([]),
                                             obl=stack([]),
                                             dh=stackset([]),
                                             lm=stack([])]),
                               id=ID])),
         pushRec(w^udus,ID) ]
     ).
```

An example of the second type of rule is the following, which updates G on the basis of a performed acknowledgement:

```
urule( doBGActsUsr, ruletype2,
       [ latest_moves: not( empty ),
         latest_moves: fst(ack(DU)),
         latest_speaker:val(c),
         next_dh_id:val(ID) ,
         valRec(w^pdu^id,PID) ],
       [ pePathRec(w^gnd,w^pdu^tognd),
         delRec(w^udus,PID),
         pushRec(w^gnd^dh,
                  record([atype=
                           record([pred=ack,
                                   dp=w,
                                   args=stackset([record([item=DU] )])]),
                          id=ID])) ]
     ).
```

An example of a rule from the third type is the following, which implements the updates on the basis of an infoRequest performed by the conversational participant w:

```
urule( doOtherActSys, ruletype3,
   [ latest_moves: not(empty),
     latest_moves: fst(inforeq(_^(Req=_))),
     latest_speaker: val(w),
     next_du_id:val(ID),
     next_dh_id:val(HID) ],
   [ pushRec(w^cdu^tognd^dh,record([atype=record([pred=inforeq, dp=w]),
                                    id=HID ])),
     pushRec(w^cdu^tognd^lm,record([pred=utter,
                                    dp=w,
                                    args=stackset([record([item=Req])])])),
     pushRec(w^cdu^tognd^obl,record([pred=answer,
                                     dp=c,
                                     args=stackset([record([item=HID])])])),
     pushRec(w^gnd^obl,record([pred=uact,
                               dp=c,
                               args=stackset([record([item=ID])])])) ]
     ).
```

Selection rules are still a topic of current work in formalization 1. However, the general approach is to include rules that adopt new intentions (these can be either update or selection rules), and other rules that pick particular dialogue acts on the basis of these intentions (and other aspects of the information state).

The control strategy differs from that presented in the previous chapter. The u-rule engine applies each type of update rule in turn, rather than applying the first rule that is matched. There is also a question of the best way to interleave update and selection, in order to achieve robust, mixed-initiative interaction.

5.3 Formalization 3

5.3.1 Informational components

In the third formalization, the system MIDAS, a more minimal approach is taken toward the aspects of information state described above. The information state consists of background knowledge and a Discourse Representation Structure (DRS, the representations used in DRT) to represent the dialogue. The former can be seen as the private part, whereas the latter resembles the public part.

Intention is represented as a plan-DRS, which is loaded into the main-DRS when a specific task has to be completed. A plan-DRS consists of a series of questions MIDAS should ask to the user, and operations it should perform. Dialogue acts are part of the main-DRS. The aspects of social state are not represented, but is taken care of by the update rules.

DUs are represented as Sub-DRSs within the main DRS G (grounded information is literally copied from the DUs to G). The one-place predicate symbol UNDER-DISCUSSION represents an item under discussion. In addition, there is a predicate symbol PREV-UNDER-DISCUSSION, which is applied to items that were under discussion in the previous turn (and, in a sense, still are). Note that UNDER-DISCUSSION and PREV-UNDER-DISCUSSION are dynamic—they are removed and replaced at each turn, in contrary to the rest of the DRS (which information is monotonically extended).

5.3.2 Formal representations

Background knowledge is represented as a set of axioms of first-order logic. This covers ontological knowledge (*inheritance*, like 'every city is a location', and 'every location is an abstraction'; and *disjointness*, like 'no city is a country') and task-specific axioms ('if you go to a city, then this city is the destination of your journey').

The dialogue is represented as a *set* of DRSs. Contributions to the dialogue can be ambiguous, which leads to more than one potential DRS ("reading", following the linguistic terminology). The linguistic phenomena that lead to more than one readings are pronouns and other anaphoric phrases, scope ambiguities, or ambiguous dialogue acts.

The MIDAS system not only generates DRSs as part of the information state, it also performs inference on these DRSs. This is done by using a translation approach to first-order logic, and exploring off-the-shelf theorem provers (Bliksem, Spass, Otter, FDPLL) and model builders (Mace, Kimba, Satchmo). First-order reasoning is applied to the following tasks:

1. resolve natural language ambiguities;
2. eliminate logical equivalent readings;
3. embed old utterances into the dialogue context;
4. plan new utterances on the basis of the current dialogue state

Task 1 involves resolution of quantifier and operator scope, pronouns, and presuppositions (such as definite description), and induce consistency checks Blackburn *et al.* (1999). Task 2 involves comparing readings by checking entailment between all possible combinations Gabsdil and Striegnitz (1999). Task 3 and 4 requires entailment checks to interpret the user's last utterance (for instance, to determine whether the user responded to a question or not) or to decide on a next move of the system (the system shouldn't ask for information already supplied by the user).

5.3.3 Dialogue moves

Dialogue acts are represented as a pair of the illocutionary force label and a DRS for the content. Dialogue moves are one of the following set:

```
reply, greeting, thanks, assert, query, check.
```

Dialogue acts are explicitly put into the main-DRS. The associated DRS is the result of parsing the input (in the case of analysis) or a sub-DRS taken from the plan-DRS or the main-DRS (in case of generation). In the first case, the DRS still contains unresolved ambiguous information, which is only resolved when integrating it with the main DRS. In the second case, there are actually two DRS involved: the content-DRS (representing the information that has to be conveyed) and a context-DRS (representing the information of the dialogue, used to generate context-specific information, such as definite descriptions).

5.3.4 Update rules

There are several different sets of update moves. The first set updates the information state on the basis of performed moves. A sample rule is the following, which updates on the basis of an assertion:

```
urule(
      assert,
      [
       fstRec(lastmoves,move(assert,U))
      ],
      [
       integrateRec(readings,assert,U),
       popRec(lastmoves)
      ]
     ).
```

There are only one pre-conditions to fire this rule: the dialogue act should be an assert (this information is provided by `getLastMoves`). The effects are caused by `integrateRec/2`, which integrates the dialogue act (move and content represented as DRS) with the main DRS (G), and by popping `lastmoves`, which updates the list of moves that need to be analysed.

The second set of rules update the information state and involve grounding. Update rules represent three different strategies for grounding:

optimistic in which the system assumes grounding without any clarification,

cautious in which the system engages in clarification when the parser is uncertain, and

pessimistic in which the system always clarifies any non yes-no answer.

The grounding strategy is implemented as a parameter the user can set, which then will be handled as a default value. When encountering extragrammatical input, the system switches to the cautious mode.

The third set of rules concern selection of the next move(s) for the system to perform (this implementation does not use selection rules, but instead uses the same urule strategy to govern both types of rule application).

An example rule shows introducing a check question in pessimistic mood:

```
urule(
      pessimistic_grounding,
      [
       valRec(strategy,pessimistic),
       fstRec(readings,drs(D,C)),
       member(bel(U,B),C),
       strict_member(under_discussion(U),C)
      ],
      [
       resource(compose(C1,the_system,X)),
       resource(member(C1,C)),
       resource(compose(C2,the_user,Y)),
       resource(member(C2,C)),
       emptyRec(readings),
       addRec(readings,drs(D,[check(X,Y,U)|C])),
       resource(generate(check,B,drs(D,C),Output)),
       pushRec(output,Output)
      ]
     ).
```

Pre-conditions for the pessimistic check are `valRec(strategy,pessimistic)` (the pessimistic strategy for grounding has been selected), checking if there is still something under discussion (by taking the first reading, and inspecting its content on an occurrence of a DRS still under discussion. The effects are an update of the main-DRS with a dialogue act `check` associated with the DRS that is under discussion, calling the generator, and store the result in the interface-variable 'output'.

5.3.5 Control strategy

The control algorithm is the same as that described in the previous chapter, applying rules until no more are applicable.

5.4 Discussion

The three formalizations in the previous section shows the advantage of the 'Trindi view' of information state for theory development and testing. In some ways, these are notational variants of the same basic ideas about dialogue modeling and update, expressed in three different formalizations: record structures, classical DRS structures, and DRS structures based on the semantics of CDRT. However, there are some significant differences in the details of the theories, using these particular formalizations.

One key aspect is the role of inference, particularly on calculating whether there is a current obligation. In the first formalization, using records, all inference must be explicitly specified via update rules manipulating the structures, so that, e.g., an obligation is taken off the stack when it has been satisfied. In formalization 3, however, obligations are not "removed" but their extent has been specified to be ending before the previous time. However, it can be a more difficult (i.e., time consuming) process to determine, for a given DRS, which obligations now hold. In some senses, the inference process has been shifted from an update on the information state to a condition. Likewise, In formalization 2, inference is done via conversion to first order theorem proving, rather than via explicit update rules, as in Formalization 1.

- differences between three formalizations
  - notational variants
  - semantics
- future formalization plans
- future implementation plans

Chapter 6

Instantiation III: Focus on Conversational Game Theory

6.1 Overview

The SRI Autoroute demonstrator possesses a dialogue capability based upon the notion of "Conversational Game" Power (1979) Houghton (1986) Carletta *et al.* (1997). An earlier version of the Autoroute demonstrator (and some of its theoretical work) predates the Trindi project. A very brief overview and some evaluation of that earlier work is given in the Trindi deliverable Bohlin *et al.* (1999). Our objective here is to update and recast the Autoroute theory and implementation within the framework of the Trindi Dialogue Move Engine (TDME), thereby providing some "proof of concept" for TDME. The concepts to be proved include theoretical and descriptive adequacy as well as implementational goals.

The structure of the description follows the general structure outlined earlier in Chapter 2. We describe first the informational components of CGT information states and then define formal representations for these components using the facilities made available in the Trindi Toolkit. Then we describe the dialogue moves of CGT. In subsequent sections, update and selection rules are described and, finally, the general control strategy for applying the rules is defined.

In order to make the description more concrete, some examples are presented from an implementation inside the Trindi Dialogue Move Engine software. We begin with some motivating comments for developing Conversational Game Theory as an approach to understanding dialogue.

The underlying theoretical objective for the SRI Autoroute Demonstator has been the development of a clearer picture of what the central notions of Conversational Game Theory (henceforth 'CGT') actually are and how they can help in dialogue system reconfiguration. By "clearer picture", we mean a formal account of CGT notions with a well defined syntax

and, we believe, intuitive semantics. In this way, the work represents a contribution to the development of CGT. As is usually remarked upon in formalized accounts, we hope that, even if *our* formalization misrepresents or oversimplifies aspects of CGT, then alternative proposals will sufficiently clearly articulated so as to be equally formalizable. In this chapter, the notions are introduced and then cast in terms of Information States and updates upon them.

There is also a more practical objective represented in the work. CGT is intended to provide a fully general account of conversation and dialogue: how and why it occurs and what it achieves when it does occur. In this most general form, CGT subjects itself to the test of linguistic adequacy: any actual or possible dialogue ought to be accountable for within the theory. However, following this line of thought, theory development can find itself driven by an agenda of amendments designed to cope with ever more difficult and increasingly rare phenomena. The very clarity of formalized accounts make them especially likely to undertake this development path. From a practical perspective however, it may be very worthwhile not to concentrate on handling 'difficult cases' but rather simply to explore the habitability of the territory that the simpler versions define. Good tools are often ones whose use is the simpler to comprehend, even if that use is more circumscribed that other possible tools.

6.2 Aspects of CGT Information State

6.2.1 CGT Informational components

In general, dialogues result when rational agents, planning to satisfy their own goals, decide to invoke the help of other rational agents. Starting a conversation is one way amongst others of, for example, finding something out or getting something done. There are usually other ways of achieving one's objectives (to find something out one might consult an encyclopaedia) but, for a variety of reasons, an agent may prefer to engage other agents in a joint enterprise.

A formalized CGT account will therefore consist, at least in part, of a formal account of rational agency, in which there are `agents` who have `goals` and can undertake `actions` in order to achieve them. Amongst these actions will be `conversational games`. The informational state of such a rational agent participating in a dialogue will be specified by beliefs about the state of the world, goals specifying how the world should be, and further beliefs about how to change the world through action. How can conversation change the world? In our development of CGT, the "world" includes a conversational scoreboard consisting of publically agreed propositions. If I make a proposal and you agree to it, then the proposal is, as it were, inscribed on the scoreboard and it becomes part of the world. It may take a substantive effort to remove or alter the inscription. The point bears comparison with making moves in a game of chess. Once a move is made, it is made and the board is updated. Changing the board requires further moves. The point is worth stressing only as a partial corrective to views of dialogue which see conversational acts as essentially being "mental-state" transformers, and in which reasoning about the effects of conversation immediately turns into reasoning about other people's mental states.

The central topic of Conversational Game Theory however is the internal structure of the games. Dialogues generally consist of many individual exchanges between the participants. In the Autoroute corpus of dialogues, there are generally exchanges concerning the origin, destination and time of the user's planned journey. These exchanges are conceived of as being games with mutually known and understood rules governing which moves conversational participants can make. In Houghton (1986), the games are thought of as shared pieces of program code. When one player initiates a game, the other "loads" the relevant piece of program and a conversational exchange can continue by running the program to completion. Here, we think of the games rather as grammars, indeed very simple ones.

The informational components required for this aspect of CGT therefore consist of knowledge of the rules of the games and an ability to play them. A formalized CGT account will define (at least) the notions of `game`, `conversational rule` and `move`.

6.2.2 CGT Formal Representations

What should a CGT information state look like? There are two main notions to capture. First, there is the state of the rational agent who plans and executes conversational games. Secondly, there is the state of the agent in his role of game-player. Conversational games are atomic actions for the rational agent. That is, playing a game is akin to executing a module or a library function. In general, at the level of rational agency, how a game is played, or how it is progressing is not an item of interest. Only the result of the game is of interest to the rational agent.

CGT Rational Agency

A rational agent possesses a set of beliefs about the world (here including what conversation may have established so far - called a scoreboard), a set of actions for changing the world and a goal. These three objects can be regarded as arguments to a planning function whose output is a plan: a (partially ordered) set of actions to carry out. In the current Trindi DME instantiation, we represent the state of the rational agent simply by the plan that he currently has and his current conversational scoreboard

$$\begin{bmatrix} \textsc{plan} & stack(action) \\ \textsc{scoreboard} & set(propn) \end{bmatrix}$$

A plan is a stack of actions, which currently include only conversational games. An action is denoted by

$$\begin{bmatrix} \text{TYPE} & gameid \\ \text{CONTENT} & propn \\ \text{INITIATOR} & person \\ \text{RESPONDER} & person \\ \text{GOAL} & propn \end{bmatrix}$$

type identifies the type of the game to be played. Its value is of type *gameid*, an atomic datatype including `qw`,`cnf`, and `pardon`. *content* identifies the content or issue of the game. The issue raised by the game is a proposition. The initiator is the person who makes the first move in the game and the responder is the person who takes the next turn. Turn-taking is enforced within games. Each takes a value of type *person*, an atomic datatype including just `p1` and `p2`. The goal of the game is also a proposition to be established by the game.

Propositions in the autoroute scenario are highly restricted. There is a stock of proper names, 6 unary predicates (`from, to, time, starttime, endtime, mode`), lambda abstraction and one binary predicates (`knows`).

CGT Conversational Game Player

Conversational Games are defined using recursive transition network notation and a conversational game player is consequently thought of as a network traverser, or equivalently, conversational game parser. Consequently, the information state required for playing a game includes sufficient information for traversing such a network, which includes notions of a current game state, an indicator of how much of the input string has currently been parsed, and a pushdown stack of game states. By "input string" we understand a sequence of semantic representations which are the meanings of the utterances.

Game states are defined as follows

$$\begin{bmatrix} \text{GAME} & gameid \\ \text{NODE} & \begin{bmatrix} \text{NAME} & nodeid \\ \text{PLAYER} & person \end{bmatrix} \\ \text{PROPVALUE} & stackset(propn) \\ \text{WHOAMI} & person \end{bmatrix}$$

gameid is an atomic datatype including `qw`, `inf`, `pardon` and `interrupt`, corresponding to querying games, information-giving games, pardon games and interruptions. A `node` identifies a state within a network and consists of a node name `s0, s1, s2 ...` and a `person` (p1 or p2) indicating whose turn it is to make the next move. (The latter information is strictly redundant since, by convention, p1 always makes the first move and turn-taking within games is enforced). `propvalue` identifies the current propositions under discussion. It is currently

implemented as a "stackset". `whoami` indicates whether the system is playing the role of p1 or p2 in the current game.

The informational state of a game player includes not just the current game state he is in but also the state of the current game parse. An Agenda Item defines the possible states of game parsing as follows

$$\begin{bmatrix} \text{GAMESTATE } state \\ \text{CURRTOKEN } token \\ \text{GAMESTACK } stackset(state) \end{bmatrix}$$

`gamestate` is the current state of the conversational game as defined above. `currtoken` defines how many tokens in the input string have been parsed in the current state. `gamestack` is the stack of game states required to handle recursion in the conversational games properly. (This is a 'stackset' so that we can check for possible loops in the parsing process – when a new possible state is generated, we check all possible earlier states to ensure that we are not generating a state that we have already encountered).

Finally, the state of a game player includes not just the state of the current game parse but also the other possible parses available. (This is a necessity unless game parsing is deterministic and fully incremental). We define the Information State of a Conversational Game Player as follows

$$\begin{bmatrix} \text{AGENDA} & stack(agendaitem) \\ \text{CURRTOKEN} & token \\ \text{ALLTOKENS} & stack(set(propn)) \end{bmatrix}$$

The `agenda` is a stack of possible game parses. In the general control strategy (see also below), after any addition to the input string, the agenda is updated and then sorted according to a preference mechanism. For example, after the system makes an utterance "When do you want to go?", there will be at least two possible parses of the dialogue so far. One will include the system having issued a query concerning the destination. The other will include the system having made a move which was not publically decipherable. The latter possibility enables one to make sense of a subsequent "pardon" move by the user. The preference mechanism may initially prefer the first interpretation over the second, but, if the user should indeed say "pardon" then the first will be discarded and the second will become the most preferred interpretation.

`currtoken` is the index into the string of tokens generated so far `alltokens` is the string of tokens themselves.

6.2.3 CGT Dialogue Moves

Conversational Game Theory is a *two-level* theory. There are games and there are moves. Games are functions from scoreboards to scoreboards. That is, their essential nature is to update the set of propositions agreed. Moves are functions between sets of propositions under discussion. Their essential function is to modify a set of propositions until a set that can be agreed upon is reached.

The games and moves defined in the Trindi CGT implementation are shown in figure 6.1. A query game opens with a query move (qw) or a "restricted query move" (qw-r) which should be followed by a reply. The querier may then either terminate the game with an acknowledgment or make a confirmation move. Possible responses to a confirmation include a "reply-yes" move, a "reply-no" move and a "reply-modifier" move. The latter move encodes a correction such as "No, to Leicester" following a confirmation of "You want to go to Bicester. Is that correct?".

An information-giving game consists simply of an information move followed by an acknowledgment.

A pardon game consists of one player making a move that is (deemed to be) indecipherable and the other player saying "pardon".

An interruption game consists of one player saying something that is (deemed to be) unimportant and the other player playing an information game.

Although not shown in the diagram, pardon and interruption games can link any node in any game back to itself.

Figure 6.1 shows the general structure of games and moves, in terms of their game and move types. Games and moves also have contents. (The distinction resembles that in speech-act theory between the force of an utterance –querying, commanding, promising etc.– and its content). Being properly formal about the matter, we may define the syntax and semantics of games and moves as follows. Suppose there are finite sets of game types G and an infinite set of propositions P. Then we may write $g : \langle \gamma, p \rangle$ to denote a game g of type $\gamma \in G$ and content $p \in Powerset(P)$. This defines the syntax of game names corresponding to locutions such as "a query game concerning the time of the journey", and an "information-giving game for the proposition that the destination is London".

Similarly, given a finite set of move types M, then $m : \langle \beta, q \rangle$ denotes a move of type $\beta \in M$ and content $q \in Powerset(P)$. Again, these correspond to locutions such as "a denial that the destination is London" and "a reply that the arrival time is 3 p m".

For the semantics, if we let $Pd = Powerset(P)$ be the possible sets of propositions under discussion, then we may define the semantics of move types as:

$$I_1(\beta) : Pd \times Pd \rightarrow Pd$$

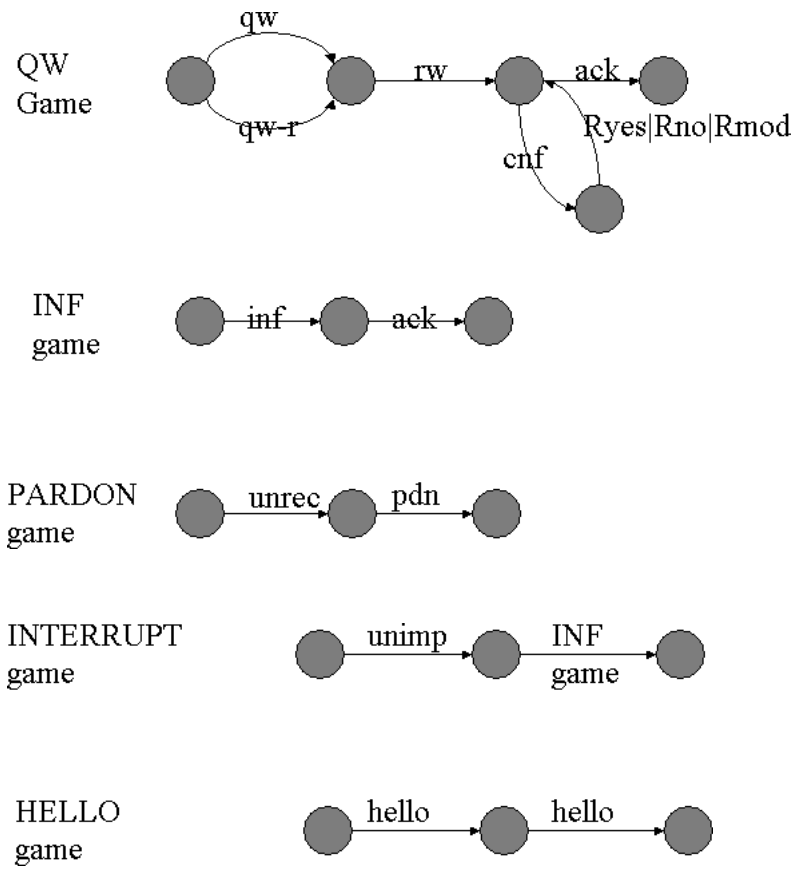


Figure 6.1: Game Definitions

and the semantics of moves as

$$[\![\, m{:}\langle \beta, q \rangle \,]\!]_1 = \lambda x_{Pd}.[I_1(\beta)(x, q)]$$

That is, the move type tells you how to update the current set of propositions under discussion and the content of the move tells you what to update it with. Here are some examples from the autoroute demonstrator (p and q are variables over sets of propositions).

Move Type	Operation	
ack	copy	$\lambda(p, q).p$
cnf	promote	$\lambda(p, q).[promote(p, q)]$
rno	delete	$\lambda(p, q).[p - q]$

The table shows that an acknowledgment leaves the current set of propositions under discussion unaltered. A confirm move promotes a member of the set into the specially marked position. A "reply-no" deletes a proposition from the set.

A move sequence which constitutes a game determines by functional composition a function

$\emptyset \rightarrow Pd$. $\emptyset$ indicates that nothing is under discussion before the first move. The function's range indicates the propositions to become commitments at the completion of the game.

The semantics of games themselves has a very similar flavour. Let Cn denote the set of propositions publically committed to. The propositions negotiated in a game determine what to update Cn with and the game type determines how to update it. There is a small complication in that games may occur within other games (e.g. confirmation games) so the total commitment effect of a game must include the possible effects of nested games.

$$I_2(\gamma) = Fn : Cn \times Pd \rightarrow Cn$$
$$[\![\ [_{g\langle\gamma,p\rangle}\ m_0, \ldots, m_n]\ ]\!]_2 = \lambda x_{Cn}.[I_2(\gamma)([\![\ m_0, \ldots, m_n\ ]\!]_2(x), [\![\ m_0, \ldots, m_n\ ]\!]_1(\emptyset)))$$

Some examples from the autoroute demonstrator are:

Game Type	Operation	
qw	add	$\lambda(p,q).[p \cup q]$
pardon	copy	$\lambda(p,q).[p]$
inf	add	$\lambda(p,q).[p \cup q]$

A game sequence which constitues a dialogue determines by composition a function $\emptyset \rightarrow Cn$. $\emptyset$ indicates the absence of commitments before the first game. The function's range indicates the set of commitments entered into during the dialogue. Commitments of D = $[\![\ D\ ]\!]_2(\emptyset)$.

6.2.4 Update and Selection Rules

The two-level nature of CGT theory means there is a division of updates also into two parts. For the rational agency part of CGT, there are currently just three very simple rules: makePlan, dropAction and undertakeAction. `makePlan` checks whether there is no current plan and if so, generates one for the goal of the user knowing a route. The current implementation of the planner `(fplan/2)` is trivial. `dropAction` checks whether the goal of the top action in the current plan is already satisfied. If so, the top action is removed from the plan. `undertakeAction` also checks whether the goal of the top action is satisfied. If it is not, it generates an Agenda Item (described further below) for the conversational game player. Agenda Items are stored in the `agenda` in the CGT Information State. In this way, the rational agent updates itself in such a way as to cause it to play a conversational game. The three "rational agency" rules contain a precondition that the `agenda` be empty so that they cannot operate whilst a conversational game is playing. Games are played until `agenda` becomes empty again, at which the rational agent restarts its deliberations.

(15) U-RULE: **makePlan**

PRE: { emptyRec(PLAN)
emptyRec(AGENDA)
latest_moves:is_unset

EFF: { fplan(knows(user,(lb(x,route(x)))), P)
setRec(PLAN, P)

(16) U-RULE: **dropAction**

PRE:
- fstRec(PLAN,*Action*)
- valRec(SCOREBOARD,*Score*)
- action_goal(*Action*,*Goal*)
- goal_sats(*Score*,*Goal*
- emptyRec(AGENDA)
- latest_moves:is_unset

EFF:
- popRec(PLAN)

(17) U-RULE: **undertakeAction**

PRE:
- fstRec(PLAN,*Action*)
- valRec(SCOREBOARD,*Score*)
- action_goal(*Action*,*Goal*)
- action_prop(*Action*,*Prop*)
- action_game(*Action*,*Game*)
- goal_notsats(*Score*,*Goal*
- emptyRec(AGENDA)
- valRec(CURRTOKEN,*CPos*), latest_moves:is_unset

EFF:
- popRec(PLAN)
- generate_agendaitem(*Game*,s0,p1,*Prop*,*CPos*,*Agendaitem*)
- pushRec(AGENDA,*Agendaitem*)

There are 6 urules pertinent to game playing: `start_parse`, `finish_parse`, `end_game_no_task`, `end_game_task_success task_covers_input`, `extend_parse`. Each corresponds directly to natural operations in a general agenda-based search mechanism. Given a current agenda of possible parsing states and a new input, the objective is to generate all legally valid extensions of those states which cover the new input. Consequently, the output of the search is a further set of agenda items. `start_parse` is a urule which simply initializes that set (in `newtasks`)

(18) U-RULE: **start_parse**

PRE:
- valRec(SEARCHSTATUS,none)
- latest_moves:is_set

EFF:
- gen_new_agenda(*AgendaSofar*)
- setRec(NEWTASKS,*AgendaSofar*)
- setRec(SEARCHSTATUS,searching)

In a similar fashion, `finish_parse` takes the new agenda items generated by the search in `newtasks` and resets the `agenda` list of current agenda items accordingly.

(19) U-RULE: **finish_parse**

PRE:
- valRec(SEARCHSTATUS,finish)
- latest_moves:is_set
- valRec(NEWTASKS,*NewTasks*)

EFF:
- setRec(AGENDA,*NewTasks*)
- latest_speaker:unset
- latest_moves: unset
- setRec(SEARCHSTATUS,none)

The 4 remaining urules dictate whether a) to end the search for valid game extensions either because there are none left, or because we have found a game termination state b) whether

to add the current search state to `newtasks` because the parse it describes includes all inputs so far c) whether to extend the current state .

`end_game_no_task` simply ends the current search for parses covering all inputs so far because the current agenda of states to extend has become empty. In such a case, there is no scoreboard update to perform since the game itself has not finished, merely the search for parses including the latest input. A dummy update is provided.

(20) U-RULE: **end_game_no_task**

PRE: { latest_moves: is_set; valRec(SEARCHSTATUS,searching); emptyRec(AGENDA)

EFF: { setRec(SEARCHSTATUS,finish)

`end_game_task_success` tests whether the top agenda item represents a success in the current search space. In our case, this means that the current state is a final state in a game's transition network. If it is, then we generate the scoreboard update operation for that game. The game that is closing may be a nested game, consequently we also need to generate a new search state (returning to the game state which is top of `gamestack`) and add it to the current agenda.

(21) U-RULE: **end_task_success**

PRE: { latest_moves: is_set; valRec(SEARCHSTATUS,searching); not(emptyRec(AGENDA)); fstRec(AGENDA,$Task$); task_succeeded($Task$,$Update$); valRec(SCOREBOARD,$Score$)

EFF: { extract_new_task($Task$,$NewTasks$); update_scoreboard($Score$,$Update$,$NewScore$); setRec(SCOREBOARD,$NewScore$); setRec(AGENDA,$AllTasks$)

`task_covers_input` tests whether the top agenda item is a search state that includes a parse of all the input so far. If it does, then that agenda item can be moved onto `newtasks`.

(22) U-RULE: **task_covers_input**

PRE: { latest_moves: is_set; valRec(SEARCHSTATUS,searching); not(emptyRec(AGENDA)); fstRec(AGENDA,$Task$); valRec(CURRTOKEN,$Token$); task_covers_input($Token$,$Task$)

EFF: { pushRec(NEWTASKS,$Task$); popRec(AGENDA)

`extend_task` is the urule which updates the current agenda by adding in new extensions of the current top agenda item

(23) U-RULE: **extend_task**

PRE:
- latest_moves: is_set
- valRec(SEARCHSTATUS,searching)
- not(emptyRec(AGENDA))
- fstRec(AGENDA,$Task$)
- valRec(AGENDA,$Agenda$)

EFF:
- generate_newtasks($Task$,$NewTasks$)
- update_agenda($Agenda$,NewTasks,$NewAgenda$)
- setRec(AGENDA,$NewAgenda$)

There are also 4 selection rules pertinent to game playing: `undertake_selection`, `end_search_no_task`, `end_search_task_success` and `extend_search`. Again, these correspond to natural operations in a general agenda-based search mechanism. Given a current state of the game and the fact that it is the *system's* turn to say something, the objective is to generate all the possible legal next moves within the game. The output of the search is again a further set of agenda items. `undertake_selection` resembles `start_parse` in the update rules, by initializing the agenda of possible next states. `end_search_no_task` resembles `end_game_no_task` in the update rules by terminating the search when nothing is left on the agenda. `end_search_task_success` resembles `end_game_task_success` by testing whether the topmost item on the agenda represents a successful point in the current search space. In this case, a legal next move represents success. Finally, `extend_search` resembles `extend_parse` by adding in legal extensions of the current state. These include all possible next moves and games from the current state.

6.2.5 CGT Control Strategy

The general picture in the CGT version of the Trindi DME is one of a dialogue monitor in the sense of chapter 1 alternating with a dialogue contribution generator. The monitor simply updates its picture of the current state of the dialogue. The picture that the monitor generates then influences what the contribution generator does. The contribution generator then updates the list of tokens to be parsed. Subsequently, the dialogue monitor then further updates its picture to incorporate the new tokens. And so on.

The overall control algorithm for CGT in the Trindi DME is therefore as follows

Control algorithm:

1. Call the update module
2. Call the register-utterance module
3. Repeat

Following the TDME framework, information states are updated according to urules, which are executed according to the default TDME update algorithm (with one small addition described further below):

Update algorithm:

1. Are there any update rules whose preconditions are fulfilled in the current IS?
2. If so, take the first one and execute the updates specified in the effects of the rule and repeat.
3. If not, stop

This update algorithm is sufficient for the simple implementation of rational agency.

The register-utterance module decides whose turn it is to say something and either waits for appropriate user input or generates something itself.

Register-utterance algorithm:

1. If it is the system's turn to say something
   (a) Call the selection module
   (b) Call the generator
2. If it is the user's turn to say something
   (a) Call the input module

Control Preferences

So far, we have described the CGT game player as a parser and the CGT information state is designed to reflect this view. However, it is a substantive question what sort of a parser a conversational game player should be? CGT itself contains no answer to this question. In this respect, the question bears the same relation to CGT as the psycholinguistic question "What properties does the human (sentence) parser possess?" to formal sentence grammars. Consequently, one may attempt to provide a psychologically valid answer, a practical engineering answer or an educative answer. For TDME, we have chosen the latter approach. The TDME conversational game parser is a non-deterministic, parallel, no-look-ahead, incremental parser.

The parse is incremental in that, after each utterance is registered, the set of next possible states is calculated, each of which is at the end of a possible path through the network. The possible path traverses all the utterances to the current conversational game. Each possible parse is then evaluated according to a utility based preference measure and the highest scoring parse is selected. By setting a system flag, the user can also review all the options and their scores and override the system choice.

The parse is non-deterministic because the unselected possibilities remain available as options. Subsequent information can cause the system to change its mind about the best parse of earlier inputs. For example, when the system asks a question "When do you want to travel?", it analyses its own output as possibly being an `unrecognized` move (that is, a publically indecipherable output), as well as being a question move. Subsequently, should the system hear "pardon", it will determine that the best interpretation of its own original utterance is that it was not recognized. No question was in fact asked. No such move was made.

In order to take account of this utility based preference idea, the Update algorithm is modified in a small way as follows

Update algorithm:

1. Are there any update rules whose preconditions are fulfilled in the current IS?
2. If so,
   (a) take the first one and execute the updates specified in the effects of the rule.
   (b) repeat
3. If not,
   (a) sort the agenda according to the preference mechanism
   (b) if the user flag is set, ask user to pick the most preferred agenda member
   (c) otherwise the highest scoring member is the most preferred agenda member

In order to make the necessary choices, all the possible next moves (or, analyses of the last move) are evaluated using weighted preference functions and the move with the best score is chosen. That is, if $f(m)$ is the score assigned to move m by preference function f and $v(f)$ is the weight assigned by the agent to the preference function. Then, to calculate which move to choose, the agent calculates for each move its utility $U(m)$ defined by:

$$U(m) = \sum_f v(f) \times f(m)$$

The agent then chooses the move with the highest utility.

Each preference function is intended to represent some salient feature of a possible move and determines a numeric score according to how much the feature applies to that move. For example, in the current Autoroute system, there are three features applicable to choices of the next possible move to make. One feature concerns how much confidence one has in one's current belief of what the current state of the game is. Confirmation moves and games naturally have a higher value for this feature than other possible moves since they are likely to increase our confidence in that we have understood what has gone before. A second feature concerns how much additional information (in terms of slot fillers) one might expect to gain from a given move. An "unrestricted" question move such as "Where do you want to go?" scores more highly with regard to this feature than "Please state the name of your destination" since the former are more likely to be answered by more informative answers such as "I want to go from Malvern to Cambridge". The third feature measures the likely impact on game length of making a given move. This feature positively disfavours confirmation games and encourages simple acknowledgments.

The weights on the preference functions are also numeric scores and reflect the value of the preference function to the agent. For example, a "safety-first" agent will generally attach a high weight to the confidence preference function and a lesser weight to the expected game length.

The mechanism permits easy reconfiguration of different strategies in playing conversational games. One can easily experiment with making the system strongly disprefer lengthy games or not care overly about accuracy.

Selection Algorithm

The move selection algorithm is conceptually and implementationally very similar to the update algorithm. Given the current state of the conversational game parser, a selection agenda of possible legal next moves is generated. For example, if a reply has been received to a question in a query game, then there are a number of possible next moves. The questionner may embark on a confirmation game with respect to the reply, or he may simply acknowledge the reply or he may decide to terminate the current game there and then without acknowledgement. The selection agenda is then sorted, again according to a utility based preference mechanism, and the topmost item on the agenda is executed as the next move in the current game.

6.3 Illustrative Examples

We illustrate the CGT framework by working through two example dialogues.

6.3.1 Confirmation examples

w: Where do you want to go from?
u: from Duxford to Cambridge
w: You are starting from Duxford. Is that correct?
u: from Oxford
w: You are starting from Oxford. Is that correct?
u: pardon
w: You are starting from Oxford. Is that correct?
u: yes
w: You are going to Cambridge. Is that correct?
u: yes
w: okay

First, we recall that the general control strategy is to alternate between a module which updates the string of dialogue tokens (either by generating one itself or accepting one from the user) and a module which incrementally parses that string. The string is initially empty so update of the token string occurs first.

The rational agent plans the next conversational game to try and he chooses a QW game concerning the origin of the journey. Next a choice must be made within the QW game as to which move to make first. Owing to the possibility of nested games, there are in fact more possible first moves than just the qw move of the QW game itself. However, the preference mechanism prefers the QW move and the system realizes it as "Where do you want to go from?". The new utterance is added to the string of dialogue input tokens.

The game parser then constructs its agenda of possible parse states which cover this string. These states again include more than just the state in which the system has made a qw move. The dialogue that ensues might for example end up beginning with a PARDON game if the user cannot understand what the system utters. Similarly, the dialogue might begin with an INTERRUPTION if the user decides that the system's utterance is simply not important enough to merit responding to. In the parallel parsing model for the implemented TRINDI CGT system, all these possibilities are generated and then they are chosen between. Again, the preference initially is that the move was indeed a qw move which results in a set of propositions under discussion including `from(X)`. However, other possible states, including one where the initial utterance was not understood and in which no propositions are currently under discussion are maintained on the agenda. At this point, the "dialogue monitor" has completed its task and control reverts to the register-utterance module.

The register-utterance module determines from the state at the top of the agenda that it is now the user's turn to say something and the standard Trindikit input modules are invoked for obtaining an input of "from Duxford to Cambridge" (or rather, its semantics [`from(duxford),to(cambridge)`]) from the user.

The dialogue monitor is again called to update its parse of the input string which now includes one more token. The preferred analysis of the input is that it represents an rw (or reply) move. The resulting preferred dialogue state is one where the propositions under discussion have been refined from `from(X)` to [`from(duxford),to(cambridge)`]. That the utterance is indeed a possible *reply* is determined by reasoning about the relation between `from(X)` and the content of the utterance [`from(duxford),to(cambridge)`]. Not any utterance could be a reply.

The register-utterance module determines that it is the system's turn to say something and this time it chooses to make a confirmation move, concerning one of the propositions under discussion. The move is realized as "You are starting from Duxford. Is that correct?".

The game parser again updates its parse of the input and prefers the analysis where the system has indeed uttered a confirmation move. `from(duxford)` now becomes the specially marked member of the propositions under discussion.

The register-utterance module next obtains `from(oxford)` from the user and the preferred analysis is that this is an *rmod* (modifier reply) move. Again, the analysis of `from(oxford)` as being an *rmod* depends on its relation to the marked proposition, namely `from(duxford)`. The propositions under discussion are therefore updated with `from(oxford)` replacing `from(duxford)`.

Register-utterance again determines that it is the system's turn to say something and it again chooses to make a confirmation move, this time realized as "You are starting from Oxford. Is that correct?".

The game parser again prefers the analysis where a confirmation move was made. However, this time the user replies with "pardon". This is not a valid continuation of the currently preferred path through the QW game. Consequently, this path is discarded. There is another possible path on the agenda, however, which resembles the discarded path except that the

last system utterance is analysed as the first move in a nested pardon game and in which the user's "pardon" itself completes that game. Since a nested pardon game does not alter the propositions under discussion, they remain unaltered from the state in which the system decided to confirm `from(oxford)`. In this state, register-utterance again chooses to make the confirmation move "You are starting from Oxford. Is that correct?". The user reply of "yes" is analysed as an *ryes* move and the set of propositions under discussion is unchanged.

Register-utterance again determines that it is the system's turn to say something and it again chooses to make a confirmation move, this time realized as "You are going to Cambridge. Is that correct?". After receiving a reply of "yes", the propositions under discussion are again unchanged and the system now chooses to close the game with the acknowledgment "Okay". At this point, the scoreboard is updated with the propositions [`from(oxford),to(cambridge)`].

6.3.2 User Interruption Example

w: Where do you want to go from?
u: I want to go to Cambridge
w: Okay

In this example, the user responds to the system's query with "I want to go to Cambridge". This response is not a possible *reply* to the query because its content `to(cambridge)` does not bear the right relation to the propositions under discussion `from(X)`. However, there are alternative parses of the dialogue so far. First, it may be that the system has not recognized what the user said correctly. In this case, the dialogue begins with a PARDON game and the system's next utterance should be "pardon". Alternatively, it may be that the system's own utterance was deemed to be an "unimportant" move which is the first move in an interruption game. In this scenario, the user's utterance is then the first move in a nested information-giving game. The system's utterance results in no propositions being under discussion and the user's utterance results in `to(cambridge)` being under discussion. The preference module chooses this interpretation. The register-utterance module then determines that it is the user's turn to say something and it chooses to close the interruption game with an acknowledgment. The scoreboard is updated with `to(cambridge)`. Game-playing ceases and the rational agent must consider what operation it should undertake next.

6.4 Discussion

The main objective of this chapter has been to show how an extant theory of dialogue (Conversational Game Theory, at least as understood in the baseline SRI Autoroute dialogue demonstrator) can be successfully recast in the Trindi Framework and implemented within the TrindiKit. This aim has been demonstrably satisfied.

There are a number of other points about the resulting system worthy of remark and further

discussion. We shall discuss these under three broad headings of "monitoring", "re-analysis" and "re-configuration".

6.4.1 Dialogue Monitoring and Dialogue Participation

One particularly salient point of difference between the current system and the baseline version is the use of the idea of a "dialogue monitor" alternating with a dialogue token generator. The idea of a dialogue manager being capable both of monitoring a dialogue and taking part within it is one that is made more pertinent by the general nature of the Trindi Framework. The SRI autoroute demonstrator in its original version was conceived entirely as a dialogue participant and the implementation inevitably came to reflect that picture. The idea of an incrementally updating monitor associates itself quite naturally with that of a parallel parser which develops alternative lines of analysis and, at given points, can sort them according to a preference measure. The implementation of the original demonstrator was considerably distorted by an attempt to use backtracking in its dialogue parsing. (The complication being, essentially, that one only wants to backtrack over the analyses of dialogue inputs and not over the whole dialogue generation process). Although the current implementation does separate the "participant" and "dialogue parsing" roles, the model still incorporates explicit turn-taking. For instance, although the user can show some initiative and interrupt a game by choosing to ignore the system's last utterance and start a game of his own, he can still only do this when it is his turn. If the system believes it is its turn to make a move, then the user cannot interrupt it. Therefore, a further degree of separation between the modules permitting asynchronous interaction seems highly desirable.

6.4.2 Dialogue analysis and re-analysis

The issue of backtracking and parallelism is perhaps also an interesting point of difference with the other dialogue theories developed so far in the Trindi project and discussed elsewhere in this document. Those theories currently extend a model which, after each input, commits to an analysis of that input. For example, in the formalization based on QUDs (the "Cooper and Larsson" approach), when the system asks a question it is placed in *qud* and *tmp* (or perhaps only in *tmp* depending on one's optimistic or pessimistic strategy). The point of *tmp* is precisely that if one's analysis of one's own utterance *i.e. I asked a question* turns out to be mistaken, then one can revoke it. Technically, one will delete the contents of *tmp* from *qud*. Cooper and Larsson make the point that they need to keep around enough information from previous turns in order to reinstate previous information in case of a grounding problem. It is an interesting question how much of a more general point can be made. In the general case, mere reinstatement may not be enough. Re-analysis may be required and that may generate different information states with different consequences for future action. Consider the following dialogue.

officer	: You'll improve tomorrow!
new recruit	: I hope so too.
officer	: That's an order!

The officer's final remark makes clear that he understands precisely what mistake the new recruit has made. Reinstatement of the information state immediately preceding the officer's initial order would not be sufficient to determine his final utterance. Of course, such an example is well beyond what is required for undertaking simple autoroute dialogues but this example has been deliberately chosen as an example of mis-interpretation at the level of dialogue act (i.e. order versus assertion). Re-interpretation of earlier material at the sub-sentential level is rather more common.

enquirer	: Is the Belfast flight on time?
info-desk	: The 17:05 already departed
enquirer	: Sorry, I mean the flight *from* Belfast
info-desk	: Ok. The 16:20 from Belfast due 17:55 is on time.
enquirer	: Thanks

Again, in this example, the correction by the enquirer makes it clear he has understood the original misinterpretation.

However, it does seem very likely that extended re-analysis (over several dialogue turns for example), which is certainly possible in the CGT framework, is simply not at all psychologically plausible. In fact, dialogue updates tend to be highly local. It will generally not be worthwhile trying to repair analysis of what was said much earlier. Of course, precisely similar remarks can be made concerning the generation and analysis of very long sentences even though our notion of grammatical well-formedness is so much clearer than it is for dialogues.

6.4.3 Dialogue re-configurability

One of the objectives behind our development of Conversational Game Theory has been to gain a clearer idea of what the notions in the theory might be and how the theory might be used to help in reconfiguring dialogue systems to new domains.

It can of course be argued that a simple recasting of a theory into a new framework is unlikely to lead to new insights into the theory itself. This point is well-made if the theory itself is already well-specified. In this case, the idea of "re-casting" rather resembles the idea of re-implementing a very well understood algorithm in a new programming language. However, theory does not always precede practice and implementation. The central notions have not been altered during development within the Trindi environment but the dialogue system that has been described and implemented in this chapter is not precisely the same as the baseline version with which the Trindi project began. For example, in the baseline version, sequences of confirmation moves were treated as a nested game. A confirmation game would take a proposition under discussion in the QW game and place it under discussion in the CNF game. At the end of the CNF game, the proposition would be added to the scoreboard

of agreed propositions if the game finished suitably. The proposition would however also need to be deleted from those under discussion in the enclosing QW game and this was the function of the CNF game, in its role as a move within the QW game. In the current version, confirmation moves are just moves within a QW game and do not constitute a nested game. Is this a distinction without a difference? Certainly, the current arrangement is simpler. The function of the moves associated with confirmation is simply to move information around within the set of propositions under discussion. Currently, this is just to place a proposition in the specially marked position or not, although better, more complex schemes can easily be imagined. The important point is that there is no need to assign a *dual* role to confirmations - first, as scoreboard updaters and then as propositions-under-discussion updaters. Although this appears to be a rather technical and theory-internal matter, the point acquires some significance if one asks whether nested games ought, *in principle* to be allowed to affect the set of propositions under discussion. That is, when a nested game finishes, can the set of propositions previously under discussion be different from when it began and, if so, would that be so in virtue of that type of game (including its game-type and content) being played and not in virtue of some *side effect* (e.g. the deployment of general reasoning in support of belief revision). The question is difficult. The issue turns partly on the very important question of the meaning of "nesting" in dialogue.

Nesting structure is usually suggested for dialogue in response to examples such as

Where is George Street on this map please?
Do you know where you are now?
No
Well, you're here and George Street is there

It is very plausible to assert that interruptions cannot affect the propositions under discussion - if one did, its content would appear to be directly relevant to the enclosing discussion and it becomes difficult properly to deem it an interruption at all. Similarly, PARDON games seem unable to alter the set of propositions under discussion. However, there may be other functions (follow-up questions, clarifications, disambiguations) which really require the more sophisticated analysis.

The practical perspective is also important. If the game-and-move architecture proposed here is to be adopted for the purposes of dialogue specification and management, then the degree of complexity engendered by the more sophisticated analysis may simply not be worth it.

The current CGT architecture is in fact neutral on the point. The possibility of assigning an update function over propositions-under-discussion to nested games is permitted. In the particular game and move instantiations provided for the Autoroute scenario, the functions are always, in fact, the identity function.

The second major question on which CGT is intended to provide a grasp is that of reconfigurability of dialogue systems to new domains. Some rather general and a priori points can be considered here but the real test must lie in actual experience. CGT employs three main knowledge sources. First, there is the shared knowledge of game structure. In principle, this knowledge (e.g. how to make a query, how to confirm an answer, how to interrupt and

so forth) ought to be re-usable from domain to domain. The game structure is intended to abstract from specific propositional contents. Consequently one should not expect games such as "obtain a calendar date (in dd/mm/yy format)". It is a central question for CGT whether knowledge of such structures really is able to be abstracted and redeployed. The use of such knowledge by dialogue designers might encourage more usable interfaces. For example, designers would be encouraged always to consider what a confirmation move for any user-reply might look like. Similarly, shared game knowledge means that the invocation of general reasoning in order to decide what to do next can be avoided. The restriction has positive and negative aspects. It may very well be that certain sorts of phenomena cannot be captured without requiring a hook to general reasoning. On the other hand, many phenomena may not require it and systems which can be described without hooks to wise "oracles" may be easier to understand and re-wire.

The second main knowledge source used in CGT is that employed in the preference functions. In fact, this knowledge comprises three elements: the move-features on the basis of which one can choose between moves; the relative values of those features; the weighting of those features for the agent. It is again an interesting and ultimately empirical matter whether such a solution structure can be applied and re-applied successfully. The features, values and weights are all currently hand-coded and experience in other linguistic fields could easily be invoked in an argument that the picture will not scale up. Could the features, or values or weights be learned? It may be that automatic learning techniques would not be able to produce results compatible with the knowledge-rich structures invoked in CGT.

One advantage in the CGT formalization is that knowledge about what is possible is clearly demarcated from knowledge about how to choose from what is possible. Game knowledge defines the possibilities. The Preference mechanism defines how to choose amongst them. One potential problem with Trindi-style dialogue rules employing a rather flatter structure is remembering, documenting and understanding the roles of different parts of an information state in the dialogue update rules. As a simple example, if one considers the "pessimistic_grounding" rule in Midas cited earlier

```
urule(
      pessimistic_grounding,
      [getLastMoves([]),flag(strategy,pessimistic),
       getNextMoves(Moves),ungrounded,release(no)],
      [setNextMoves([check|Moves]),releaseTurn]
     ).
```

one might note that some parts of the rule condition govern what is legally possible next (e.g. `ungrounded`) whilst other parts govern what to choose next(e.g. `flag(strategy,pessimistic)`). Although the operation of this rule is fairly easy to decipher, one can easily imagine the difficulty in deciphering the functions of parts of more complex rules. In general, reconfiguration of a complicated system employing a very flat structure of this form may be expected to be a highly complex task. Of course, flatter structure also has a possible benefit – it may be easier

to encode interesting dependencies between different aspects of information structure. As is often the case, there is an important practical trade-off between the degree of complexity permissable and how easy it is to understand, maintain and reconfigure systems which use it.

In fact, the problem arises to a smaller degree even for CGT itself since the update rules are designed both to encode the strategy for the rational agent and the search path for the conversational game player. Knowledge of rational agency is the third main knowledge source for CGT. Consequently, if one examines the planning rule

(24) U-RULE: **makePlan**

PRE: { emptyRec(PLAN)
emptyRec(AGENDA)
latest_moves:is_unset

EFF: { fplan(knows(user,(lb(x,route(x)))), P)
setRec(PLAN, P)

one can note that one of the preconditions governs rational agency (make a plan if you haven't got one) but another (the agenda of conversational moves is empty) is really designed to inhibit the game parser. There are two quite different sorts of update rules although this fact is not immediately apparent merely from the rules themselves. Again, the flexibility of the resulting system may in fact prove highly interesting and useful.

Bibliography

Allen, J. and Core, M. (1997). DAMSL: Dialogue act markup in several layers. Draft contribution for the Discourse Resource Initiative.

Allen, J. F. (1979). A plan-based approach to speech act recognition. Technical Report 131, University of Toronto. PhD Dissertation.

Allen, J. F. (1983). Recognizing intentions from natural language utterances. In M. Brady and R. C. Berwick, editors, *Computational Models of Discourse*. MIT Press.

Allen, J. F. (1987). *Natural Language Understanding*. Benjamin Cummings, Menlo Park, CA.

Allen, J. F. and Perrault, C. (1980a). Analyzing intention in utterances. *Artificial Intelligence*, **15**(3), 143–178.

Allen, J. F. and Perrault, C. R. (1980b). Analyzing intention in utterances. *Artificial Intelligence*, **15**(3), 143–178.

Blackburn, P., Bos, J., Kohlhase, M., and de Nivelle, H. (1999). Inference and Computational Semantics. In H. Bunt and E. Thijsse, editors, *Third International Workshop on Computational Semantics (IWCS-3)*, pages 5–19. Computational Linguistics, Tilburg University.

Bohlin, P., Bos, J., Larsson, S., Lewin, I., Matheson, C., and Milward, D. (1999). Survey of existing interactive systems. Deliverable D1.3, Trindi Project.

Bos, J., Bohlin, P., Larsson, S., Lewin, I., Matheson, C., (1999). Evaluation of the model with respect to restricted dialogue systems. Deliverable D3.2, Trindi Project.

Carletta, J., Isard, A., Isard, S., Kowtko, J. C., Doherty-Sneddon, G., and Anderson, A. H. (1997). The reliability of a dialogue structure coding scheme. *Computational Linguistics*, **23**(1), 13–31.

Clark, H. H. and Schaefer, E. F. (1989). Contributing to discourse. *Cognitive Science*, **13**, 259 – 94.

Cohen, P. (1996). Dialogue modeling. In R. Cole, J. Mariani, H. Uszkoreit, A. Zaenen, and V. Zue, editors, *Survey of the State of the Art of Human Language Technology*, chapter 6.3. Cambridge University Press, Cambridge, MA.

Cohen, P. R. (1978). *On Knowing What to Say: Planning Speech Acts*. Ph.D. thesis, University of Toronto. Reproduced as TR 118 Department of Computer Science, University of Toronto.

Cohen, P. R. and Levesque, H. J. (1990a). Persistence, intention and commitment. In P. Cohen, J. Morgan, and M. Pollack, editors, *Intentions in Communication*, chapter 12. Morgan Kaufmann.

Cohen, P. R. and Levesque, H. J. (1990b). Rational interaction as the basis for communication. In P. Cohen, J. Morgan, and M. Pollack, editors, *Intentions in Communication*, chapter 12, pages 221–256. Morgan Kaufmann.

Cohen, P. R. and Perrault, C. R. (1979). Elements of a plan-based theory of speech acts. *Cognitive Science*, **3**(3), 177–212.

Cooper, R. and Larsson, S. (1999). Dialogue moves and information states. In H. Bunt and E. C. G. Thijsse, editors, *Proceedings of the Third International Workshop on Computational Semantics*.

Cooper, R., Larsson, S., Matheson, C., Poesio, M., and Traum, D. (1999). Coding instructional dialogue for information states. Deliverable D1.1, Trindi Project.

Gabsdil, M. and Striegnitz, K. (1999). Classifying Scope Ambiguities. In *First Workshop on Inference in Computational Semantics*, pages 125–131.

Ginzburg, J. (1996). Interrogatives: Questions, facts and dialogue. In *The Handbook of Contemporary Semantic Theory*. Blackwell, Oxford.

Ginzburg, J. (1998). Clarifying utterances. In J. Hulstijn and A. Niholt, editors, *Proc. of the Twente Workshop on the Formal Semantics and Pragmatics of Dialogues*, pages 11–30, Enschede. Universiteit Twente, Faculteit Informatica.

Grosz, B. J. and Sidner, C. L. (1986). Attention, intention, and the structure of discourse. *Computational Linguistics*, **12**(3), 175–204.

Houghton, G. (1986). *The Production of Language in Dialogue*. Ph.D. thesis, University of Sussex.

Larsson, S., Bohlin, P., Bos, J, and Traum, D. (1999). TRINDIKIT Manual Deliverable D2.2 - Manual, Trindi Project.

Levinson, S. C. (1983). *Pragmatics*. Cambridge University Press.

Lewis, D. K. (1979). Scorekeeping in a language game. *Journal of Philosophical Logic*, **8**, 339–359.

Moore, R. and Browning, S. (1992). Results of an exercise to collect 'genuine' spoken enquiries using woz techniques. In *Proceedings of the Institute of Acoustics 14 6*, pages 613–620.

Muskens, R. (1995). Tense and the logic of change. In U. Egli, P. Pause, C. Schwarze, A. von Stechow, and G. Wienold, editors, *Lexical Knowledge in the Organization of Language*, pages 147–183. John Benjamins, Amsterdam / Philadelphia.

Perrault, C. R. and Allen, J. F. (1980). A plan-based analysis of indirect speech acts. *American Journal of Computational Linguistics*, **6**(3-4), 167–82.

Poesio, M. and Muskens, R. (1997). The dynamics of discourse situations. In P. Dekker and M. Stokhof, editors, *Proceedings of the 11th Amsterdam Colloquium*, pages 247–252. University of Amsterdam, ILLC.

Poesio, M. and Traum, D. (1997). Conversational actions and discourse situations. *Computational Intelligence*, **13**(3), 309–347.

Poesio, M. and Traum, D. (1998). Towards an axiomatisation of dialogue acts. In J. Hulstijn and A. Nijholt, editors, *Proc. of the Twente Workshop on the Formal Semantics and Pragmatics of Dialogues*, pages 207–222, Enschede. Universiteit Twente, Faculteit Informatica.

Power, R. (1979). The organization of purposeful dialogues. *Linguistics*, **17**, 107–152.

Sadek, D. and De Mori, R. (1998). Dialogue systems. In R. D. Mori, editor, *Spoken Dialogues with Computers*. Academic Press.

Searle, J. R. (1969). *Speech Acts*. Cambridge University Press, New York.

Traum, D. R. (1994). *A Computational Theory of Grounding in Natural Language Conversation*. Ph.D. thesis, University of Rochester, Department of Computer Science, Rochester, NY.

Traum, D. R. (1999). 20 questions on dialogue act taxonomies. In *Proceedings of Amstelogue'99 workshop on the semantics and pragmatics of dialogue*.

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*, pages 1–8, New Mexico.

Traum, D. R. and Hinkelman, E. A. (1992). Conversation acts in task-oriented spoken dialogue. *Computational Intelligence*, **8**(3). Special Issue on Non-literal Language.