Information states and dialogue move engines

Peter Bohlin, Robin Cooper, Elisabet Engdahl, Staffan Larsson
Department of linguistics
Göteborg University
Box 200, Humanisten, SE-405 30 Göteborg, SWEDEN
{peb,cooper,engdahl,sl}@ling.gu.se

Abstract

We explore the notion of *information state* in relation to dialogue systems, and in particular to the part of a dialogue system we call the *dialogue move engine*. We use a framework for experimenting with information states and dialogue move engines, and show how an experimental dialogue system currently being developed in Göteborg within the framework can be provided with rules to handle accommodation of questions and plans in dialogue.

1 Introduction

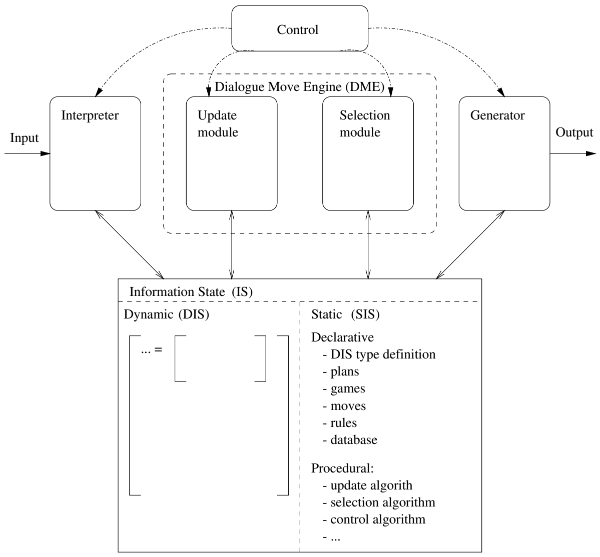
We use the term *information state* to mean, roughly, the information stored internally by an agent, in this case a dialogue system. A *dialogue move engine* updates the information state on the basis of observed dialogue moves and selects appropriate moves to be performed. In this paper we use a formal representation of dialogue information states that has been developed in the TRINDI[1], SDS[2] and INDI[3] projects[4].

The structure of this paper is as follows: First, we give a brief description of a general dialogue system architecture which can be used for experimenting with different kinds of information states and dialogue move engines. We explain the distinction between static and dynamic information state, and discuss how rules formulated in terms of conditions and operations on information states can be used to (1) update information states based on observed dialogue moves and (2) select dialogue moves based on the current information state. We present a particular notion of dynamic information state based on Ginzburg's theory of Questions Under Discussion (QUD) [Ginzburg, 1996a; 1996b; 1998]. An experimental dialogue system which uses this notion of information state is presented. We then look at the role of accommodation in information state transitions and point to examples of two kinds of accommodation: accommodation of questions under discussion and of dialogue plan. Finally, we argue that accommodation should be associated with update rules, or tacit moves (not associated with an utterance), and show how the implementation of these rules yields improved behaviour in the experimental dialogue system.

2 General architecture

The general architecture we are assuming is shown in (1).

(1)



[1] TRINDI (Task Oriented Instructional Dialogue), EC Project LE4-8314, www.ling.gu.se/research/projects/trindi/

[2] SDS (Swedish Dialogue Systems), NUTEK/HSFR Language Technology Project F1472/1997, http://www.ida.liu.se/ nlplab/sds/

[3] INDI (Information Exchange in Dialogue), Riksbankens Jubileumsfond 1997-0134.

[4] We will illustrate our discussion from a Swedish dialogue in the travel booking domain that 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.

The components in the architecture are the following:

- Information State (IS), divided into Dynamic IS (DIS) and static IS (SIS)
- Interpreter: Takes input utterances from the user and gives interpretations in terms of moves (including semantic content). The interpretation is stored in the DIS.
- Update module: Applies update rules (specified in SIS) to the DIS according to the update algorithm (also specified in SIS)
- Selection module: Selects move(s) using the selection rules and the move selection algorithm specified in SIS. The resulting moves are stored in the DIS. The update module and the selection module together make up the dialogue move engine.
- Generator: Generates ouput utterances based on the contents of the DIS.
- Control: wires together the other modules, either in sequence or through some asynchronous mechanism.

Apart from the general architecture show in (1), the framework also specifies formats for defining update rules, selection rules and dialogue moves (see section 2.2), and provides a set of tools for experimenting with different information states, rules, and algorithms. Simple intepreters and generators are also provided. To build a dialogue system, one needs to supplement the framework with definitions of rules, moves and algorithms, as well as the structure of the dynamic information state.

2.1 Static and dynamic information state

We distinguish between *static* (SIS) and *dynamic* (DIS) information states of a dialogue agent. The dynamic state is the part of the information state which can change during the course of the dialogue, while the static state stays the same. In the static state we can include rules for interpreting utterances, updating the dynamic information state, and selecting further moves. Also, we can include dialogue move definitions, plan libraries, dialogue game definitions (e.g. in the form of Finite State Transition Networks) and domain databases, insofar as these knowledge sources do not change during the dialogue. If e.g. the database can be updated during the dialogue by information from the user or in any other way, or if the system is capable of learning new rules, these resources should be included in the dynamic state.

2.2 Moves and rules

Traditionally, dialogue moves (or speech acts) are defined using preconditions, effects, and a decomposition [Allen, 1987]. From the perspective of implementing a dialogue move engine, we think it may be useful to think about what a dialogue system (or any dialogue participant) actually needs to do (not necessarily in a sequential order):

- interpret utterance from the user
- update the information state according to the move(s) (supposedly) performed by the user
- select a move/moves to be performed by the system
- generate appropriate utterance to perform move(s)
- update the information state according to the move(s) performed by the system

Instead of defining the dialogue moves themselves in terms of preconditions and effects, we define *update rules (u-rules)* and *selection rules (s-rules)* for updating the DIS based on the recognised move(s) and selecting the next move(s), respectively.

The update rules are rules that update the (dynamic) information state, e.g. when the user has input something to the system. The selection rules are rules that both update the (dynamic) information state and selects a dialogue move to be executed by the system. Both rule types have preconditions and effects. The preconditions are a list of conditions that must be true of the information state. The effects are a list of operations to be executed if the preconditions are true. The preconditions must guarantee that the effects can be executed.

Dialogue move definitions consist of a name, a type (optional) and a list of number and types of arguments (e.g., speaker, content, etc). Dialogue moves are the output of analysis and input to generation. Also, they are the objects selected by s-rules. U-rules may refer to them, and they may be part of the information state.

We also use the term *tacit move* to refer to the act of applying an update rule, i.e. the act of updating the DIS.

3 Question-based DIS

The question about what should be included in the dynamic 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.

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

$$\begin{bmatrix} \textsc{private} & : & \begin{bmatrix} \textsc{plan} & : & \textsc{List(Action)} \\ \textsc{agenda} & : & \textsc{Stack(Action)} \\ \textsc{tmp} & : & \begin{bmatrix} \textsc{bel} & : & \textsc{Set(Prop)} \\ \textsc{qud} & : & \textsc{Stack(Question)} \\ \textsc{lm} & : & \textsc{Move} \end{bmatrix} \end{bmatrix} \\ \textsc{shared} & : & \begin{bmatrix} \textsc{bel} & : & \textsc{Set(Prop)} \\ \textsc{qud} & : & \textsc{Stack(Question)} \\ \textsc{lm} & : & \textsc{Move} \end{bmatrix} \end{bmatrix}$$

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 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 PLAN field contains 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 field, 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.

We have included a field TMP that mirrors the shared fields. This field keeps track of shared information that has not yet been grounded, i.e. confirmed as having been understood by the other dialogue participant[5]. 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 on TMP until it has been acknowledged by the other dialogue participant whereupon it can be moved from TMP to the appropriate shared field.

The SHARED field is divided into three subfields. One subfield is a set of propositions which the agent assumes for the sake of the conversation. The second subfield is for a stack of questions under discussion (QUD). These are questions that have been raised and are currently under discussion in the dialogue. The third field contains information about the latest move (speaker, move type and content).

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

4 GoDiS

In Göteborg, an experimental dialogue system called GoDiS (Gothenburg Dialogue System) is being developed based on the framework described above and using the type of dynamic information state described in Section 3.

4.1 Rules, moves and algorithms

In this section we describe some of the rules and algorithm definitions we use. The current algorithms are very simple and the behaviour of the system is therefore mainly dependent on the definitions of the update and selection rules.

Update algorithm:

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

Selection algorithm:

1. Are there any selection rules whose preconditions are fulfilled in the current IS? If so, proceed to step 2. If not, stop.
2. Does the rule specify a dialogue move? If so, stop. If not, execute the updates specified in the effects of the rule.
3. Repeat

Control algorithm:

1. Call the interpreter
2. Call the update module
3. Call the selection module
4. Call the generator
5. Call the update module
6. Repeat

The update rules include rules for question and plan accommodation, as well as rules for handling grounding and rules for integrating the latest move with the DIS.

The latter rules look different depending on whether the user or the system itself was the agent of the move. As an illustration, in (2) we see the update rule for integrating an "answer" move when performed by the user, and in (3) the converse rule for the case when the latest move was performed by the system[6].

(2) U-RULE: **integrateLatestMove(answer(usr))**

$$\text{PRE: } \begin{cases} \text{valRec(SHARED.LM, answer}(usr, A)) \\ \text{fstRec(SHARED.QUD, } Q \text{),} \\ \text{relevant\_answer(} Q, A \text{)} \end{cases}$$

$$\text{EFF: } \begin{cases} \text{popRec(SHARED.QUD)} \\ \text{reduce}(Q, A, P) \\ \text{addRec(SHARED.BEL, } P) \end{cases}$$

(3) U-RULE: **integrateLatestMove(answer(sys))**

PRE: valRec(PRIVATE.TMP.LM, answer(sys, Q, A))

$$\text{EFF: } \begin{cases} \text{setRec(SHARED.LM, answer}(sys, Q, A)) \\ \text{popRec(SHARED.QUD)} \\ \text{reduce}(Q, A, P) \\ \text{addRec(SHARED.BEL, } P) \end{cases}$$

In the current implementation, interpretation and generation are canned, which means that the range of input and output strings is very restricted. However, it is also possible to communicate using moves directly, e.g. by typing `ask(P^(price=P))` instead of `'What is the price?'`.

The semantics (if it deserves the name) represents propositions as pairs of features and values, e.g. (month=april), and questions are λ-abstracts over propositions, e.g. $\lambda x(month = x)$. A set of propositions and a query together constitute a database query which is sent to the database once the system has received sufficient information to be able to answer the question. A question and an answer can be reduced to a proposition using β-reduction. For example, the question λx(month=x) and the answer *april* yield the proposition $[\lambda x(month = x)](april)$, i.e. $(month = april)$.

4.2 Dialogue plans

In our implementation, the static information states contains, among other things, a set of *dialogue plans* which contain information about what the system should do in order to achieve its goals. Traditionally [Allen and Perrault, 1980], it has been assumed that general planners and plan recognizers should be used to produce cooperative behaviour from dialogue systems. On this account, the system is assumed to have access to a library of domain plans, and by recognizing the domain plan of the user, the system can produce cooperative behaviour such as supplying information which the user might need to execute her plan. 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.

5 Accommodation

We define dialogue moves as updates to information states directly associated with utterances. If you take a dialogue or information update perspective on Lewis' notion of accommodation, it corresponds to moves that are tacit (i.e. not associated with an utterance). Tacit moves can be seen as applications of update rules, which specify how the information state should be updated given that certain preconditions hold. Tacit moves could also be called "internal" or "inference" moves. The motivation for thinking in terms of accommodation has to do with generality. We could associate expressions which introduce a presupposition as being ambiguous between a presuppositional reading and a similar reading where what is the presupposition is part of what is asserted. For example, an utterance of "The king of France is bald" can either be understood as an assertion of that sentence and a presupposition that there is a king of France or as an assertion of the sentence "There is a king of France and he is bald". However, if we assume an additional tacit accommodation move before the integration of the information expressed by the utterance then we can say that the utterance always has the same interpretation.

In a similar way we can simplify our dialogue move analysis by extending the use of tacit moves so that the updates to the information state normally associated with a dialogue move are actually carried out by tacit moves. One argument for doing this is that very

[6]Note that this definition embodies an optimistic approach to grounding by putting answer(sys, Q, A) in SHARED.LM, thereby assuming the systems utterance was understood by the user. Also, the system optimistically assumes that the user accepts the resulting proposition P by adding it to SHARED.BEL.

few (if any) effects of a move are guaranteed as a consequence of performing the move; rather, the actual resulting updates depend on reasoning by the addressed participant. Thus, we define an update rule **intergrateLatestMove** which, given that the latest move was accepted by the system, performs the appropriate update operations. The updates for a move are different depending on whether it was the system or the user who made the move, but the same module is used in both cases.

5.1 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 our dialogue:

(4) `$J: vicken månad ska du åka`
(what month do you want to go)
`$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 (5).

(5) 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).

5.2 Accommodating the dialogue plan

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

(6) `$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 in order to have content. 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. 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 his plan, thereby providing a plan to ask relevant question for flights, e.g. when to travel?, what date? etc. Once he has accommodated this 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 plan needs to be accommodated. The other interesting aspect of this is that accommodating the dialogue plan 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.

5.3 Associating accommodation with tacit moves

Update rules can be used for other purposes then intergrating the latest move. For example, one can provide update rules which accommodate questions and plans. One possible formalization of the **accommodate_question** move is given in (7). 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 reqires 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.

(7) U-RULE: **accommodateQuestion**(Q, A)

PRE: $\left\{ \begin{array}{l} \text{valRec(SHARED.LM, answer(usr,}A\text{)),} \\ \text{inRec(PRIVATE.PLAN, raise(}Q\text{))} \\ \text{answer-to(} A, Q \text{)} \end{array} \right.$

EFF: $\left\{ \begin{array}{l} \text{delRec(PRIVATE.PLAN, raise(}Q\text{))} \\ \text{pushRec(SHARED.QUD, } Q\text{)} \end{array} \right.$

6 Accommodation in a dialogue system

In this section we show an example of how the dialogue system described above can handle accommodation of questions and plans. The example is actual (typed) dialogues with the system, supplemented with information about dialogue moves, tacit moves, and (partial) illustrations of the systems dynamic information state at different stages of the dialogue.

```
$S: Welcome to the travel agency

$U: flights to paris.
```

```
[ private = [ plan   = ⟨⟩
              agenda = ⟨⟩
              tmp    = ... ]
  shared  = [ bel = {}
              qud = ⟨⟩
              lm  = answer(usr,[how=plane,to=paris]) ] ]
```

```
# accommodatePlan
     # setRec(private.plan,[raise(A^(how=A)),
                            raise(B^(to=B)),
                            raise(C^(return=C)),
                            raise(D^(month=D)),
                            raise(E^(priceclass=E)),
                            respond(F^(price=F))])

# accommodateQuestion
     # delRec(private.plan,raise(A^(how=A)))
     # pushRec(shared.qud,A^(how=A))

# integrateLatestMove(answer(usr))
     # popRec(shared.qud)
     # addRec(shared.bel,how=plane)

# accommodateQuestion
     # delRec(private.plan,raise(A^(to=A)))
     # pushRec(shared.qud,A^(to=A))

# integrateLatestMove(answer(usr))
     # popRec(shared.qud)
     # addRec(shared.bel,to=paris)

# refillAgenda
     # popRec(private.plan)
     # pushRec(private.agenda,raise(A^(return=A)))

private =
    agenda = [raise(A^(return=A))]
    plan = [raise(A^(month=A)),
            raise(B^(priceclass=B)),
            respond(C^(price=C))]
shared =
    bel= [(to = paris)
          (how = plane)]
    lm = answer(usr,[how=plane,to=paris])
```

```
$S: From where do you want to go?
```

```
[ private = [ plan   = ⟨ raise(R^(return=R)),
                         raise(M^(month=M)),
                         raise(C^(class=C)),
                         respond(P^(price=P)) ⟩
              agenda = ⟨⟩
              tmp    = ... ]
  shared  = [ bel = {(to=paris),(how=plane)}
              qud = ⟨X^(from=X)⟩
              lm  = ask(sys,Y^(from=Y)) ] ]
```

After interpreting the users utterance as an **answer** move with the content `[how=plane,to=paris]`, the system starts checking if there are any u-rules which apply. Following the ordering of the rules given in the list of rule definitions, it first checks if it can perform **integrateLatestMove(answer(usr))**. 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, **accommodatePlan** (8)[7] does apply, since there is (in the SIS) 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.

(8) U-RULE: **accommodatePlan**

PRE: $\left\{ \begin{array}{l} \text{emptyRec(PRIVATE.PLAN)} \\ \text{emptyRec(SHARED.QUD)} \\ \text{emptyRec(PRIVATE.AGENDA)} \\ \text{valRec(SHARED.LM, } LM \text{)} \\ \text{relevant\_to\_plan(} LM, Plan \text{)} \end{array} \right.$

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

Once this rule has been executed, the update algorithm starts from the beginning of the rule list. This time, it turns out the preconditions of **accommodateQuestion** hold, so the rule is applied. As a consequence of this, the preconditions of **integrateLatestMove(answer(usr))** now hold, so that rule is applied. Actually, it turns out that the latest move 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 u-rules apply, the system proceeds to perform the next action on the plan: asking where the user wants to travel from. At the end of the dialogue fragment, the dynamic information state after the system has uttered this question is shown.

[7] 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.

References

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

[Allen and Perrault, 1980] J. F. Allen and C. Perrault. Analyzing intention in utterances. 15(3):143–178, 1980.

[Allen, 1987] J. F. Allen. *Natural Language Understanding*. Benjamin Cummings, Menlo Park, CA, 1987.

[Carletta, 1996] J. et. al. Carletta. Hcrc dialogue structure coding manual. Technical Report HCRC/TR-82, HCRC, 1996.

[Cooper and Larsson, 1999] R. Cooper and S. Larsson. Dialogue moves and information states. In *Proc. of the Third IWCS*, Tilburg, 1999.

[Ginzburg, 1996a] J. Ginzburg. Dynamics and the semantics of dialogue. In Seligman and Westerståhl [Seligman and Westerståhl, 1996].

[Ginzburg, 1996b] J. Ginzburg. Interrogatives: Questions, facts and dialogue. In *The Handbook of Contemporary Semantic Theory* [Lappin, 1996].

[Ginzburg, 1998] J. Ginzburg. 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, 1998. Universiteit Twente, Faculteit Informatica.

[Lappin, 1996] ed. Lappin, Shalom. *The Handbook of Contemporary Semantic Theory*. Blackwell, Oxford, 1996.

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

[Seligman and Westerståhl, 1996] Jerry Seligman and Dag Westerståhl, editors. *Logic, Language and Computation*, volume 1. CSLI Publications, 1996.

TRINDI homepage: `www.ling.gu.se/ research/ projects/ trindi/`