What exactly is wrong and why?
Tutorial Dialogue for Intelligent CALL Systems

Manfred Klenner (Zurich) / Henriëtte Visser (Heidelberg)

Abstract

We introduce a dialogue-based explanation facility for Intelligent CALL (ICALL) Systems. Our prototype system, DiBEx, uses meta reasoning to build up an explanation (error) tree, given a faulty user input. It relies on correct grammatical subtheories, instead of explicit error taxonomies. DiBEx, thus, realizes anticipation free error diagnosis. The system enters in a tutorial dialogue with the student, where each explanation (dialogue) step is based on the principles of a single tutorial strategy and a dynamic user model.

1 Introduction

Computer Assisted Language Learning (CALL) has been one of the first incarnations of E-Learning, which has not only become a trend nowadays but also a commercial factor. CALL has its own international conferences and a wide range of academic prototypes and commercial products are available. As with E-Learning software in general, advanced CALL systems are based on the principles of multi media design, facilitating modalities like audio and video, e.g. speech input and output, animated graphics and avatars for guiding the user. Although these design standards do improve the quality of E-Learning systems, it is widely accepted that an optimal learning environment is one where the learner is guided by an intelligent personalized tutor, that adapts to the level of expertise of the learner, for example in tailoring explanations to the user's domain knowledge and selecting appropriate exercises. This is the area of Intelligent Tutorial Systems (ITS) and User Modelling (UM), so far a mainly academic discipline. If at all, only few commercial (CALL) products incorporate such capabilities. The problem is that domain knowledge, reasoning and learning facilities and - in the case of verbal explanations - natural language understanding capabilities are to be integrated; altogether a non-trivial task for many application areas. As a consequence, intelligent tutorial systems seem to be suited well only for restricted domains, e.g. circuit design (Davis 1984).

We believe that in the area of CALL systems we can in fact find such restricted domains (portioned by the thematic closure of the exercises of a lesson). The domain knowledge here is knowledge about grammar (word order, agreement, subcategorisation etc.) and the domain modelling can be held local to each lecture. The challenge then is the reasoning and natural language part of such an intelligent CALL (ICALL) system. We will elaborate on this below. Let us first discuss the kind of learning tasks we are focussing on.

ISSN 1615-3014

2 Examples of Explanatory Dialogues

We are focussing on language generation tasks involving grammatical knowledge, such as forming a sentence given a couple of words in base form or translating a sentence. Such an exercise could be: Given the words 'Mann sehen grün Auto (man see green car)', form a full inflected sentence (in the word order as given).

The correct answer is: 'Der Mann sieht das grüne Auto' (the man sees the green car). Here, diverse grammatical knowledge (of German grammar) is inolved: the subject must be realized in nominative case, an article must be selected that agrees with the noun in case, gender and number. The object should be realised as accusative, adjective inflection, the article and the noun must correspond to this.

Or consider the translation of the sentence "La lune s'est levée" into German language. The result should be: 'Der Mond ist aufgegangen (the moon has risen)'. Here, among others, the learner must know that 'lune' ('Mond') is masculine in German (in contrast, it is feminine in French). If the learner responds with 'Die Mond ist aufgegangen', she probably is not aware of that fact.[1] A system that simply responds with 'no' is of little use. In Fig. 1, a wide range of possible feedback statements to a faulty input 'die Mond' are enumerated.

```
1.  'die' is wrong (is feminine)
2.  the wrong article was selected
3.  'Mond' is masculine
4.  the noun is masculine
5.  the noun is masculine, but the article is not
6.  there is an agreement error
7.  there is an agreement error within the subject
8.  'die' and 'Mond' do not agree
9.  the article and noun do not agree
10. 'die' and 'Mond' do not agree in a grammatical category
11. 'die' and 'Mond' do not have the same gender
12. the (grammatical category) case of the subject is wrong
13. a grammatical category of the subject is wrong
14. the subject is wrong
```

Figure 1: Error Feedback

But what is the right level of explanation? For a learner who is familiar with grammatical concepts like subject, agreement etc. the explanations given in e.g. 7 or 13 might suffice, i.e. help her to find the correct solution. In contrast, beginners might prefer the explanations in 1 or 5. Explanation 3 ('Mond' is masculine') is a very sophisticated one, since it requires some inferences to be made by the student (e.g. 'Mond' is masculine' means 'something that also should be masculine actually is not'). But if a single explanation does not provide sufficient information to allow the learner to correct her input, a tutorial explanatory dialogue is needed. In Fig. 2 an example of such a top down dialogue is given.

[1] Note however that often more than one explanation of a learner's mistake is possible. Here the learner might well know that 'Mond' is masculine but erroneously believe that 'die' also is masculine

```
system: there is an agreement error
user:  where?
system: at the subject
user:  why?
system: the article and the noun do not agree in gender
user:  i can't fix the problem
system: German 'Mond' is masculine
user:  i see, but what am i supposed to do?
system: choose the corresponding masculine article
user:  what is it?
system: 'der'
```

Figure 2: A Top Down Explanatory Dialogue

This is an example of an exhaustive dialogue, where the learner is not able to utilize the explanations of the tutor at all. Thus, the dialogue finishes with a correction statement. We expect such a situation to be the exception, which nevertheless should be manageable by the system. Normally, after one or two dialogue steps, the learner should come up with the right solution. Otherwise, the user model must be adapted to a lower level of expertise. Another tutorial strategy is based on a bottom up explanation as given in Fig. 3.

```
system: the article is wrong
user: why?
system: 'die' is feminine
user: so what?
system: the article must agree with the noun
user: what's the problem?
system: German 'Mond' is masculine
user: i see
```

Figure 3: A Bottom Up Explanatory Dialogue

Why not simply provide the student with the correct solution? Language learning is a kind of problem solving. Psychological studies have indicated that self explanation can improve problem solving skills (Conati et al. 2001). We adopt that view and opt for a tutorial strategy that helps the student to explain her mistakes to herself.

Another question that remains open is whether students pay attention to the metalinguistic feedback at all. However, studies seem to suggest that students appreciate meaningful learner-computer interaction and are willing to get involved in a longer dialogue as long as the option of receiving the correct answer immediately remains open (Heift 2001).

In the next section we present an overview of the historical development of CALL and the current challenges in system design. Our prototype system DiBEx (**Di**alogue-**B**ased **Ex**planation) is described in section 4.

3 Previous Work

3.1 CALL Systems: Introduction

Computer Assisted Language Learning (CALL)[2] systems were developed as early as the 60s, when mainframe computers were installed at universities and large firms and experimentation with more personalized, non-mathematical and industrial uses of the computer began. The pedagogy at that time was largely convinced of the value of repetitive and memorization tasks for the learning of language and the use of computerized exercises meant alleviation for the teacher, but the use of such programs remained limited. In the 70s the role of the teacher began to change toward the role of a tutor, not only a corrector, who communicates more individually with the students and sets tasks. During this time (70s and 80s) the computer world was revolutionized by the introduction of Personal Computers for individual use. The flexibility of PCs in schools and in the workplace lead to initial euphoria about the role of the computer in learning. Initially the attraction of the new medium seemed so all pervasive, that it was often thought that the mere opportunity to use the computer meant that the students would concentrate on the learning tasks. Pedagogy in the CALL software was often ignored or restricted and lagged behind the changes taking place in the classrooms. The introduction of multimedia (audio, video, advanced graphics) in the 90s greatly improved the user attractiveness of the medium. Also the new communication technologies such as the Web and email meant direct access to original language environments. This convinced many users to invest in a PC for the home, also because of the growing pressure toward life-long learning to secure the workplace and participate in social life. Even though these developments increased the market for CALL products, it also increased the drive to rapid commercialization and reduction of funds for research and prototyping.

Although many commercial products are now available for a variety of language learning tasks, the pedagogical demands have also increased. The computer no longer has the role of being a mere tool for 'drill and kill' exercises, nor even that of a tutor imparting 'neutral' information or knowledge, it is expected to be a teacher or facilitator. Real life teachers of course have a variety of choices at their disposal to respond to a student. They can respond both in writing and orally, make judgements about the students' expertise and learning progress and adapt accordingly, in certain teaching situations they can overlook language errors to advance the communicative tasks and build confidence. This flexibility of the teacher, both consciously driven by the curriculum and the underlying pedagogy and partly driven by intuition, is very difficult to simulate.

Research in this direction is sometimes referred to as Intelligent Tutoring Systems (ITS), or more specifically for the language learning field, Intelligent CALL (ICALL). These systems are conceived to include a user model, a tutorial model and underlying NLP tools. The user model registers the learner history (exercises chosen, errors made), the tutorial model includes the general curriculum and the individual exercises, as well as the response routines. Ideally both the diagnosis and response routines are supported by NLP tools, such as a parser. All

[2] For an interesting overview of the history of CALL related to technological developments: Warschauer 1996.

modules should be dynamic, i.e. output should change according to the information supplied by the other modules. These tasks are certainly not trivial, but would improve the quality of CALL systems. One of the criticisms of the current systems is the rigidity in handling the input and in responding to the student. Most exercises allow at the most the input of short phrases, sometimes only clicking and moving graphic objects. Behind such systems are fixed sets of correct input to the exercises. However, learners respond with irritation when an alternative but correct answer is not accepted. Another reason for irritation is the fact that systems often cannot analyze faulty input and do not give error explanations. If error explanations are provided, these are usually fixed sets of responses to anticipated errors. Even though experienced teachers can often anticipate which errors students will make, the limitations of such an approach for a computer system are clear: if the error was not anticipated, there will be no reaction at all and if the explanation given is not understood there is no possibility of asking the teacher.

These limitations can be overcome to a certain extent by introducing NLP techniques, such as parsing, to analyze the input as well as to provide an indepth error diagnosis. Building on such a diagnosis, a response can be generated, which then can also be varied according to input of the user model and the tutorial model.

3.2 The Problem of Error Diagnosis

In this section we discuss the problem of error diagnosis which is a prerequisite for any error explanation. In general, parsing ill-formed sentences for error diagnosis is not trivial, especially if the correct solution is unknown. In DiBEx, the problem is somewhat relaxed, since the correct solution is known, i.e. is specified in advance by a human exercise creator. This simplifies error diagnosis significantly, as we will discuss in section 4.2. We nevertheless give here a brief overview of the general problems involved in error diagnosis.

Normally, parsing is the application of a correct grammar to an input sentence. If the input sentence is incorrect, parsing simply fails. We illustrate this problem with a simple example (taken from Yazdani 1988). Consider the phrase structure grammar (PSG) given in Fig. 4.

1 *S* → *NP VP*
2 *VP* → *Verb*
3 *NP* → *Det Noun*
4 *NP* → *Name*
5 *Det* → *the*
6 *Noun* → *student*
7 *Name* → *joe*
8 *Verb* → *laughs*

Figure 4: A Simple PSG

Given the student input: 'The joe laughs', a reasonable (high level) error explanation would be: 'The noun group is wrong'. However, to fix the problem that the noun group (!) is wrong, the system must infer that 'the joe' in fact is meant to be a noun group. Since parsing the noun group fails, parsing of the whole sentence fails, including the verbal phrase. But 'the verbal

phrase is wrong' is not true. Note that not necessarily the phrase of the first proof failure is to blame. Assume an additional S rule $S \rightarrow PPVP$ as the first S rule. Clearly, PP then is the first phrase structure rule that fails to be proved. But the system should not come up with the error explanation: 'the prepositional phrase is wrong'. The only trivial solution to that problem is to augment the phrase structure grammar with error rules. For example:

NP ⟶ *Det Name (Error: names are used without determiners)*

But this widely used strategy leads to a proliferation of error rules (e.g. *PP* → *Prep Det Name* and so on). Moreover, more complex phenomena like agreement hardly can be modelled by such an anticipation based error diagnosis (but see Schneider/McCoy 1998, see also Menzel 1988 for a discussion of these problems).

The alternative to the anticipation of errors is an approach called *model based diagnosis*. We discuss this kind of error diagnosis in the next section.

3.3 Menzel's Approach

In model based, anticipation free error diagnosis, no knowledge about possible errors is represented. There are no error rules (mal-rules) or precompiled error taxonomies (for an example of such a system, see McCoy et al. 1996). Instead, error diagnosis is based on a universal procedure that infers errors made from a pool of valid domain knowledge. This kind of error detection has first been applied to technical systems (Davis 1984, De Kleer/Williams 1987). Menzel (1992) adapted it to the field of computer assisted language learning. We give a brief introduction to Menzel's approach, since we use his representations. Note however that we do not use Menzel's reasoning scheme, instead a meta interpreter is used.

First of all, to enable model based error diagnosis grammatical knowledge is represented more explicitly than in standard NLP systems. For example, the principle of agreement in unification based grammar is captured by the unification of feature structures. So the two feature structures [gender=mas] ('der') and [gender=mas] ('Mond') do unify. However, the fact that they *should* agree in gender is not explicitly represented. It is - so to say - built in. Such meta knowledge about grammar is necessary, if those parts of the domain knowledge that are violated by the user input should be identifiable. A possible, more explicit representation for that kind of knowledge is: *agree*(*noun,article,case*). This simply means: The noun and the article must agree in case.

An anticipation free diagnosis model is made out of a set of such (named) model components and the relation between them. To model, for example, case agreement within a prepositional phrase, the following components can be used:

```
agr1: agree(noun,article,case)
agr2: agree(preposition,noun,case)
```

where *agr*1 and *agr*2 are the names of the model components. Each model component has an *input* and an *output value*. The function *in*(*x*) (input value) returns the value of the first argument of a model component (which is always a triple) wrt. the third argument. For example: *in*(*agr*1)= case of noun. Similarly, *out*(*x*) is the value for the second argument with respect to

the third, i.e. *out*(*agr*1)=case of article. Such information is part of the lexicon which is represented by facts of the general form *val*(*word*,*feature*,*value*).

Given such a definition of *in* and out, the (conditional) semantics of a model component can be stated as:

$$type\_agree(x) \wedge \neg ignored(x) \rightarrow in(x) = out(x).$$

To paraphrase this: given *x* is of type agree and (the principle) *x* is not ignored, the values of a feature of the first and the second argument must be the same. We come back to that below. Relations between (!) model components - similarily - relate input and output values. For example: *out*(*agr*2)=*in*(*agr*1), which guarantees a consistent value assignment for the case feature within prepositional phrases.

How can this be used for error diagnosis? Let us look at an example: 'mit des Hauses' ('with the house'). Its representation is:

```
agr1: agree(Hauses,des,case) % both genitive
agr2: agree(mit,Hauses,case) % mit/dative, Hauses/genitive
```

The relation between *agr*1 and *agr*2 is not violated, since it holds that *out*(*agr*2)=*in*(*agr*1) (*in*(*agr*1) = genitive and *out*(*agr*2) = genitive). But according to the conditional semantics of a model component of type agree, *in*(*x*)=*out*(*x*) also must hold, i.e. *in*(*agr*1)=*out*(*agr*1) and *in*(*agr*2)=*out*(*agr*2). The later restriction is violated, since *in*(*agr*2)=*dative* and *out*(*agr*2)=*genitive*. So, ¬(*in*(*x*)=*out*(*x*)) is true for *x*=*agr*2. Applying this (via modus tollens) to the conditional semantics of model components of type agree, we get *ignored*(*x*) (additionally, ¬(¬(*type-agree*(*x*))) holds, thus *ignored*(*x*) is true via disjunctive syllogism). *ignored*(*agr*2) represents the fact, the student has ignored the agreement between the noun and the preposition.

Note that this exposition is nothing but a rough description of the kind of reasoning involved in anticipation free error diagnosis.

We saw an example of a *rule violation.* In a similar manner, the error can be traced back to a (lexical) fact, namely: *val*(*Hauses*,*case*,*gen*).

Menzel's system, although designed to generate explanations at different levels of generality, is not meant to participate in an explanatory tutorial dialogue (which is the focus of DiBEx).

3.4 NLP-based Intelligent Tutoring Systems

Work in the area of ITS often does not incorporate any natural language processing (NLP) facilities at all. Graphical devices and precompiled (canned) text are used instead. This way, the 'oddities' of dialogue modelling (i.e. parsing, semantic interpretation, response planning and surface generation) can be circumvented and, consequently, these systems focus on other essentials of the tutoring problem e.g. domain and learner modelling. These approaches are mainly concerned with cognitive modelling.

There are a few exceptions, e.g. Aleven et al. (2001). Here, a dialogue based geometrical tutor is introduced that helps students to explain the reasons behind their problem-solving actions.

In this system, the student is requested to state (in natural language) an explanation of a (previously learned) geometrical theorem. The task of the tutor is to evaluate and criticize the student explanation. For example, the student's statement 'angles are equal' as a trial to explain the isosceles triangle theorem is criticized with a question: 'are any two angles congruent? '. The ultimate goal is to help the student to form a complete and correct statement about a geometrical rule. This is not to far from DiBEx' objective: to help the student find out why her input is incomplete. In both cases, more precise knowledge needs to be activated by the student. However, in Aleven's system the parsing results are mapped to a taxonomy of possible student answers, where canned feedback messages are stored under each answer category.

The system that comes most close to DiBEx is Johanna Moore's PEA (Program Enhanced Advisor) framework (Moore 1995). PEA is a tutor that helps students to improve their programming skills in Lisp (the readability and maintainability of Lisp programmes). For example, the students input for the task of accessing the first element of a list x might be: (car x) (where *car* is the command to access the first element of a list). A correction advise is, for example: (first x). An expert systems called EES is used to generate a so-called development history that fixes the reasons for such an advice (a kind of explanation structure). Follow-up why questions use the development history to provide more precise reasons (explanations) for the advice.

The quality of the generated explanations is debatable, but the discourse planning component is compelling. PEA uses Rhetorical Structure Theory (RST, see Mann et al. 1992) to integrate various knowledge resources (domain knowledge, discourse history, a speech act and user model) into a single model component.

One main difference to DiBEx is that the development history is explicitly tailored to the explanation task. That is, PEA uses a domain theory about how to enhance a program. It is, so to say, an anticipation based tutor (in contrast to DiBEx). As a consequence, the DiBEx model is much more concerned with getting concise explanations out of knowledge structures that are only indirectly related to the explanation task at hand.

EES uses a user model based on the so-called overlay technique, that is, the users' knowledge and goals are assumed to be a subset of the system's knowledge and goals. EES records four types of knowledge about the user: the user's goal, her knowledge about methods for achieving goals and performing acts, the concepts the user is familiar with and the facts the user believes. Any individual user model is based on a stereotype representing knowledge common to all users (e.g. (know user (concept program))) and further facts derivable from the user interaction with the system (e.g. the lisp functions the user supplies in her program are assumed to be known to the user). The user model of DiBEx is very similar to that of EES.

ISSN 1615-3014

4 The DiBEx System

DiBEx (Klenner/Visser 1999) is a prototype ICALL system that is able to participate in an explanatory tutorial dialogue. It is implemented in the programming language Prolog and has been tested on three different kinds of grammatical phenomena (agreement, word order, dominance). The main components of DiBEx are:

- background knowledge about grammar theory
- an error diagnosis component
- a reasoning component that forms an explanation structure from the faulty learner input and the grammatical (sub-) theory under consideration
- tutorial strategies (currently a single one) how to guide the learner to identify his/her mistake and how to correct it
- a dynamic user model
- a dialogue module including a natural language parser and a natural language generator

Additionally, there is an interface that facilitates the specification of new exercises. In a specification file, a complete and correct solution of the problem must be provided by the CALL lecturer. This is a prerequisite for DiBEx' error diagnosis component. The system expands these specifications to its internal format (see section 4.1). A subsequent user input is compared to that representation as part of the error diagnosis. In this module, also an error explanation tree is built, which is the starting point for the tutorial dialogues.

We describe the representation of grammatical knowledge (section 4.1), and the principles of error diagnosis (section 4.2) done by DiBEx. We then discuss in detail the tutorial component (section 4.3) including the dialogue module.

4.1 The Representation of Grammatical Knowledge

4.1.1 Grammatical Facts

We build on the representation format described in Menzel (1992). For example, the representation for the subject of 'der Mond ist aufgegangen' is given in Fig. 5.

```
val(mond,case,nom).      val(der,case,nom).
val(mond,number,sg).     val(der,number,sg).
val(mond,gender,mas).    val(der,gender,mas).

val(np1,isa,noun_group).
val(np1,isa,subject).
val(np1,has_part,mond).
val(np1,has_part,der).
val(np1,noun,mond).    % the head of np1 is the noun mond
```

Figure 5: Representing Knowledge about 'der Mond'

Here, for example, the first fact states that the case of 'Mond' is nominative.[3] Additionally, general grammatical restrictions are represented that way. For example that the noun restricts the grammatical features case, number and gender:

```
val(noun, restricts,[case,number,gender]).
```

```
val(mond,isa,noun).
val(der,isa,artice).
val(article,isa,word).
val(noun,isa,word).
```

Let us elaborate on the translation task given above, i.e. 'La lune s'est levée'. We assume as input the incorrect subject surface form: 'die Mond'. This corresponds to the facts in Fig. 6 (suppressing some possibilities for the ease of exposition):

```
val(mond,case,nom).
val(mond,number,sg).
val(mond,gender,mas).

val(die,case,nom).    % also: accusative, feminine, singular
val(die,number,sg).   % also: plural
val(die,gender,fem).  % unrestricted, if plural nominative
```

Figure 6: Representing Knowledge about 'die Mond'

Given the representation in Fig. 5 and 6, we can easily fix the problem, viz. that 'die' is wrong because the 'gender' of 'die' is 'fem' instead of the required 'mas'. One just has to find the two most distinct corresponding facts from the correct and the incorrect representations, i.e.,

```
val(der,gender,mas) and val(die,gender,fem).
```

Given such an analysis, some error messages (e.g. 'die' is wrong) or correction advice ('replace 'die' by 'der') could already be generated. However, to initiate an elaborated tutorial dialogue, additional knowledge is necessary. For dialogue steps like 'there is an agreement error' a theory of agreement is needed as well as a reasoning scheme that finds all theory parts that are explicitly or implicitly (defined below) violated by the user input.

Note that the editing distance of 'die' being nominative, masculine, plural compared to the correct values (nominative, masculine, singular) is the same as the one given in the example (Fig. 5) ('die' being nominative, feminine, singular). One has to alter one value (plural or feminine). However, since 'lune' in French is singular, feminine, an error stemming from a 'gender transfer' is more plausible than an error steming from a 'number transfer'. We have just started to investigate such a model of heuristic 'error selection'.

4.1.2 Grammatical Rules

We represent grammatical rule knowledge as a logical theory in the programming language Prolog. Such a theory must serve two purposes: under a declarative point of view, it must be a valid description of the grammatical knowledge about a grammatical subtheory (e.g. agree-

[3] *np*1 represents the *internal* identifier of the subject noun group.

ment), procedurally, it must be applicable to the user input in order to qualify the input as correct ('YES') or incorrect ('NO').

Fig. 7 represents the theory for agreement within noun groups.

```
1 proper_agreement(Noun_Group) :-
2  val(Noun_Group, isa, noun_group),
3  val(Noun_Group, noun, Noun),                    % the head
4  val(noun, restricts, Restriction_Set),          % case, ..
5  forall(val(Noun_Group, has_part, Word),
6     feature_agreement(Noun, Word, Restriction_Set)).

7 feature_agreement(Word, Modifier, Restriction_Set) :-
8  forall( member(Feature, Restriction_Set),
9     agree(Word, Modifier, Feature)).

10 agree(Head, Modifier, Feature) :-
11  val(Head, Feature, Val),
12  val(Modifier, Feature, Val).
```

Figure 7: Agreement within Noun Groups

Let us begin with the end, the lines 10 to 12. Here, the agreement of a head (the Prolog variable Head) and a modifier wrt. a grammatical feature is expressed. This rule states that the head (e.g. 'Mond') and its modifier (e.g. 'die') agree in a feature (e.g. 'case'), if the head has a value Val for that feature and the modifier has the same value Val for that feature. The grammatical fact that all (!) grammatical features of the head and the modifier must agree is defined in the predicate *feature_agreement* (lines 7 to 9). Here Restriction_Set is the list [case,number,gender], the forall predicate forces that each member of that list is bound to the variable Feature and successively passed to the predicate *agree* (together with Word and Modifier). The predicate *proper_agreement* (lines 1 to 6) is invoked with the name of the noun group, e.g. *np*1 in our example. Line 2 checks whether the predicate is applicable at all, i.e. whether Noun_Group actually is a noun group (there are other predicates for agreement). Line 3 fetches the head of the noun group ('Mond'), line 4 selects the restriction set ([case,number,gender]) and the lines 5 and 6 express that each word must agree with the noun wrt. to restriction set.[4]

The background theory in Fig. 7 can be used as the basis for a verbal explanation of the theory of agreement. The (rough) translation of it given in Fig. 8 might exemplify this.

[4] The logicians among the readers might have noticed that something like not(Word=Noun) is missing. This is omitted here for readability.

```
Something has a 'proper_agreement'
    IF it is a noun group with noun Noun as (head) noun
    AND all words of the noun group agree with Noun in
      all features from 'Restriction_Set'

Two words agree wrt. to 'Restriction_Set'
    IF they agree for all features from 'Restriction_Set'

Two words agree wrt. to a grammatical feature
    IF they share the same value for that feature
```

Figure 8: A Rough Translation into Rules

The background theory in Fig. 7 also is operational. So, for example, a Prolog call 'proper_agreement(np1)' terminates with 'NO', since 'np1' ('die Mond') does not agree in gender. On the other hand, a Prolog call 'proper_agreement(np2)', with 'np2' representing the correct phrase 'der Mond', would terminate with 'YES'.

Agreement, word order, dominance etc. are modelled in such a fashion in Prolog. The input of the learner must pass all those grammatical subtheories. If the application of one subtheory fails, a meta interpreter is invoked, which is able to derive an explanation (trace) of the faulty input.

Because it is possible to directly access Prolog program code within Prolog, it is easy to write an interpreter for Prolog code in Prolog. Such an interpreter is called a meta-interpreter (cf. Sterling/Shapiro 1986). Meta-interpreters are usually used to add some additional control mechanisms to Prolog e.g., altering the built-in proof strategy (e.g. forward instead of backward chaining). They are also used to generate proof trees as part of knowledge-based (expert) systems or as debugging facilities. Actually, the DiBEx tutor is a kind of debugger that produces proof trees. It traces the user input and produces an explanation tree that marks violated parts of the grammatical (sub-)theory under consideration. We will discuss this in more detail in the next section.

4.2 Error diagnosis

The input of the student is automatically translated into grammatical facts (cf. Fig. 6 for the representation of 'die Mond'). This is nothing but a lexicon lookup and not a full parse. For example, 'die' is represented in the lexicon as, among others, an article with feminine gender (nominative, singular or plural). No attempt is made to recognise that (the incorrect) 'die Mond' is meant to be the *subject noun group* of the sentence 'die Mond ist aufgegangen'. DiBEx has no procedures to parse ill-formed sentences (see the discussion in section 3.2). But what then makes an analysis of the student input in terms of the grammatical rule knowledge (e.g. the rules for agreement from Fig. 7) feasible? To prove *proper_agreement* from Fig. 7 'die Mond' must be classified as a noun phrase, otherwise the predicate (trivially) fails already at a very early stage of processing. We have defined a mapper that compares (a copy of) the correct input to the incorrect input. According to its minimal editing distance to the incorrect input, the copy is adapted: correct facts are replaced by incorrect facts, the rest is left as it was.

This produces, for example, the representation in Fig. 9.

```
% from the correct solution
val(np1,isa,noun_group).
val(np1,isa,subject).
val(np1,has_part,mond).
val(np1,noun,mond).

val(np1,has_part,die). % replacement of 'der' by 'die'

% from the faulty user input
val(mond,case,nom).     val(die,case,nom).
val(mond,number,sg).    val(die,number,sg).
val(mond,gender,mas).   val(die,gender,fem).
```

Figure 9: Guessing the Intended Input Structures

Crucial here is the replacement of 'der' by 'die'. Now, 'die Mond' is accessible with *np*1 as a noun group. We believe that this approach is valid for a wide range of faulty input structures (although very complicated and abstruse errors might lead to a confusing representation). At this stage, all grammatical subtheories (agreement, word order etc.) can be invoked with these 'intended user input structures'. If one subtheory does fail (a Prolog fail), error diagnosis is triggered.

The error diagnosis component of DiBEx is realized as a meta interpreter. It receives as input the subtheory that is violated by the (augmented) user input, and it outputs an explanation of the error. This explanation is represented by a tree structure of the subtheory, where all variables are instantiated with the values from the user input and where, beginning from the root of the tree, a path down to a tree leaf indicates everything that is explicitly (a leaf node) or implicitly (intermediate nodes) violated by the user input. Let's have a look at the (slightly simplified) error tree given in Fig. 10.

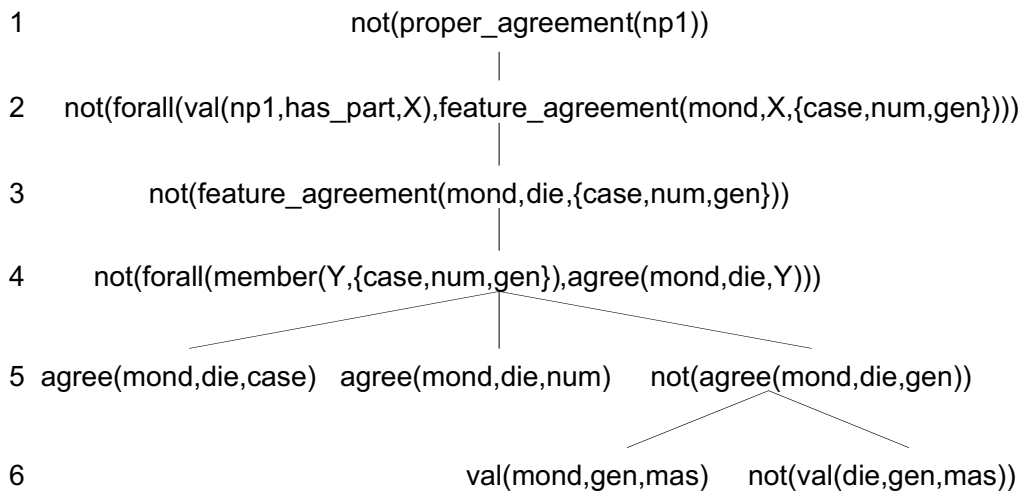

Figure 10: Explanation (Proof) Tree

This tree structure represents an error proof tree underlying the application of the subtheory from Fig. 7 to the user input *np*1 ('Die Mond', cf. Fig. 9) as represented by the call *proper_agreement*(*np*1). The whole tree can be read as an explanation sequence of the (erroneous) input:

The noun group ('np1') does not have a proper agreement (top node, depth 1 in Fig. 10), since: not for all of its words, the features of the word and the features of 'mond' do agree (subordinate of top node, depth 2). Not for all of the words of the noun group, the features of the word and the features of 'mond' do agree (subordinate of top node), since: it is not true that 'mond' does agree with 'die' in case, number and gender (next subordinate, depth 3), and so on.

The basic fact that is violated by the user input is given at the bottom of the tree in Fig. 10, namely 'not(val(die,gen,mas))'. This is derived from the failure to prove line 12 in Fig. 7 (the variable Modifier holds 'die', Feature holds 'gen' (gender), Val is preoccupied with 'mas'). Starting from that prove failure, the error diagnosis component marks (recursively) all intermediate nodes of the tree that dominate this leaf as violated (the 'not' in front of each node description). Each intermediate node forms a (more or less abstract) error description for the user input. 'not(val(die,gen,mas)' could be verbalized as: 'die' is not masculine, or 'not(agree(mond,die,gen)))' as: 'mond' and 'die' do not agree in gender.

It is the task of the tutor to select the right level of abstraction. This includes the choice of an appropriate tree node to be verbalized with respect to the ongoing dialogue and the user model, but also the choice of how the arguments of the selected node should be referred to: 'die' could be referred to by 'die' or by 'the article' or even by 'a word' (although this might be too general in most circumstances). The tutor is introduced in the next section.

4.3 The Tutor

Depending on the dialogue history, the user model and a tutorial strategy, the tutor selects a node from the error tree that is assumed to be the best response (reaction) to a user question (or, initially, the faulty user input).

DiBEx currently uses the following tutorial strategy:

1. guide the user to identify the faulty fact in a top down manner, i.e. identify the grammatical principle violated by the input
2. provide sparse information, i.e. explain everything on an abstract level if possible (e.g. refer to a word by its word class).
3. introduce at each explanation step at least one piece of information that is new to the user
4. avoid becoming too vague (i.e. don't use more than one indefinite description)

Often such a top down strategy is called a Socratic (tutorial) Strategy, attributing this kind of dialogue to the Greek philosopher Socrates. The assumption here is that people already know the correct answer to (philosophical) questions and the task of the philosopher (tutor) is to elicit that knowledge by posing the right questions.

Let's continue our example. At the beginning of the dialogue, the root node of the error tree (Fig. 10) is selected, i.e. *not*(*proper_agreement*(*np*1)). According to tutorial principle 2, the tutor decides to suppress the referent *np*1, thus, 'There is an agreement error' is generated. The dialogue history is updated, i.e., it records that *proper_agreement* is verbalized without reference to *np*1. Assume 'where? ' as follow-up question. This would lead to an update of the user model. If the user does not question a grammatical term, the user is assumed to know the underlying concept. So, for example, *know*(*user,definition*(*agreement*)) is added to the user model. *np*1 is the only referent recorded in the dialogue history that can provide localization information, thus it is selected and: 'at the subject' (*np*1 is the internal identifier of the subject noun group) is generated. The dialogue history (*referred_to_by*(*np*1,*subject*)) and the user model (*know*(*user,is*(*np*1,*location_of_error*))) are updated.

Let's assume the next user question to be 'why?'.[5] A why question[6] causes the tutor to search for a more precise explanation down the error tree. The next node (depth 2, Fig. 10) is a complex formula (not(forall ...)) that includes the quantifier *forall*. There are several reasons not to select that node. Since the user is assumed to know the concept 'agreement', she knows that there must be two words that do not agree in a grammatical category. The only new information here could be that one of these words is 'Mond' (but there are only two words). Another observation leads us to a general rejection of such formulas as an explanation to a why query. They (procedurally) define the grammatical principle represented by their immediate mother node. So agreement is defined by: all words of a noun group must agree with the noun in case, number and gender. Such formulas are thus reserved to questions that ask for the definition of a grammatical concept. They are appropriate, for example, if DiBEx generates: 'there is an agreement error' and a user follow-up question is: 'what does that mean?'. Formulas including forall are thus not accessible to the tutor given a why question. But what about the node at depth 3 (not(feature_agreement(..)))? The user knows from the previous dialogue steps already that 'there is an agreement error at the subject' and she knows the definition of agreement. This implies the knowledge expressed by node at depth 3, viz. 'the article and the noun do not agree in a grammatical category'.

The node at depth 4 (again) is a formula containing forall. But at depth 5, the tutor succeeds: not(agree(mond,die,gen)): the information that 'Mond' and 'die' do not agree *in gender* is new. According to tutorial principle 2, 'Mond' and 'die' are referred to by their word class.

[5] In general, why questions can have several interpretations. Here: 'Why at that place' (and not e.g. at the verb phrase) and 'why is there agreement error at the subject?'.

[6] Other types of user questions are clarification questions (i don't understand), questions for more detailed information (where, in what way) and requests for instruction (what am I to do).

There are rules that restrict the kind of reference to predicate arguments:

- given two arguments, they must be referred to at the same level:
  *"'die' and the noun do not agree" is odd.
- reference to arguments must not be too general:
  *'a function word is wrong' (given e.g. several function words)
- the reference to arguments may switch from abstract (word class) to specific (word-form), but not the other way round. So a sequence:

  system: 'die is wrong'
  user: 'why'
  system: 'because the article does not agree with the noun'

  is pragmatically ill-formed.

The system verbalises the selected node with: 'The article and the noun do not agree in gender'. A follow-up why question can be answered in various ways. Fetching the next negated node (not(val(die,gen,mas)) at depth 6) would lead to: 'die is not masculine'. An alternative response is: 'die' is feminine. This fact is not explicitly represented in the error tree, but can be derived by domain specific rules like:

```
if not(val(X,Y,U)) is an explanation to a question of type 'why'
then val(X,Y,V) (part of the background knowledge) also is
```

Here, the if-part is *not*(*val*(*die*,*gen*,*mas*)) and the then-part *val*(*die*,*gen*,*fem*). This gives: (the gender of) 'die' is feminine.

Finally, *val*(*mond*,*gen*,*mas*) ('German Mond is masculine') is also a possible explanation. Again, a domain specific rule can be formulated:

```
if
not(val(Word1,Feature,Val)) is an explanation to a why question
then
val(Word2,Feature,Val) (under the same negated node) also is
```

The instantiations are: Word1/die, Word2/der, Feature/gen, Val/mas.

Two final dialogue steps from Fig. 2 are left to be explained. DiBEx generates 'choose the corresponding masculine article' as a response to the user question 'what am i supposed to do'. The following rule captures this:

```
if not(val(Word,Feature,Val)) is a leaf in the error tree
  and WC is the word class of Word
then inform(system,user,choose(user,WC,attribute(WC,Val)))
   (i.e. 'choose the corresponding Val WC') is an answer
   to a request for instruction
```

The instantiations are: Word/die, Feature/gen, Val/mas, WC/article. If the user still is not able to correct its input, but asks: 'what is it', then this question can be answered by looking up the correct solution in the knowledge base.

5 Summary and Outlook

We argued in this paper that an explicit representation of grammatical knowledge can be used for the verification of user input as well as for error diagnosis and error explanation (given a faulty input). An error tree generated by a meta interpreter serves as the starting point for a tutorial dialogue that (currently) is based on a single Socratic tutorial strategy: provide sparse error descriptions, give hints of why something is wrong. Each node in the error tree can be used as a more or less abstract error explanation. A why question posed by the student in response to such an explanation can be resolved by ascending the tree to a subordinated (negated) node, leading to a more precise error description.

Future work will be concerned with extending and broadening the grammatical subtheories that can be handled (integrating e.g. subcategorization and selectional restrictions). Also, the DiBex model needs to be refined in order to be able to recognize and cope with multiple errors. We also plan to port our mainly rule-based dialogue model to a plan-based model in the style of Moore (1995). Finally, we will integrate at least a second tutorial strategy and a preference strategy to cope with concurrent explanations given of a single error. The empirical evaluation of our tutor must await these modifications to the DiBEx model.

We would like to thank the two anonymous reviewers for their suggestions.

References

Aleven, Vincent/Popescu, Octav/Koedinger, Kenneth (2001): "Towards tutorial dialog to support self-explanation: Adding natural language understanding to a cognitive tutor". In: Moore, J. D./Redfield, C. L./Johnson W. L. (eds.): *Artificial Intelligence in Education: AI-ED in the Wired and Wireless Future. Proceedings of AI-ED 2001*: Amsterdam 246-255.

Conati, Cristina/Larkin, James/VanLehn, Kurt (1997): "A computer framework to support self-explanation". In: du Bolay, B./Mizoguchi, R. (eds.): *Artificial intelligence in education: knowledge and media in learning systems. Proceedings of AI-ED 97 World Conference.* Amsterdam: 279-286.

Davis, Randall (1984): "Diagnostic reasoning based on structure and behaviour". *Artificial Intelligence*, 24 (1-3): 347-410.

De Kleer, Johan/Williams, Brian C. (1987): "Diagnosing multiple faults". *Artificial Intelligence*, 32 (1): 97-130.

Heift, Trude (2001): "Error-specific and individualised feedback in a Web-based language tutoring system: Do they read it?". *ReCALL* 13 (1): 99-109.

Klenner, Manfred/Visser, Henriëtte (1999): "A dialogue-based explanation module for error correction for intelligent CALL systems". *EuroCall'99*: 158-160.

McCoy, Kathleen. F./Pennington, Christopher A./Suri, L. Z. (1996): "English error correction: A syntactic user model based on principled 'mal-rule' scoring". In: *(UM-96) Proceedings of the Fifth International Conference on User Modeling. Proceedings of the conference, 2-5 January 1996, Kailua-Kona, Hawaii.* Newton, MA: 59-66.
http://www.cis.udel.edu/~mccoy/publications/1996/McCoPenn96a.ps

Mann, William C./Matthiessen, Christian M./Thompson, Sandra A. (1992): "Rhetorical structure theory and text analysis". In: Mann, William C./Thompson, Sandra A. (eds.): *Diverse Analyses of a Fund Raising Text.* Amsterdam: 39-78.

Menzel, Wolfgang (1988): "Diagnosing grammatical faults - a deep modelled approach". In: O'Shea, Tim/Sgurev, Vasil (eds.): *Artificial Intelligence III - Methodology, Systems, Applications.* Amsterdam: 319-326.

Menzel, Wolfgang (1992): *Modellbasierte Fehlerdiagnose in Sprachlehrsystemen*. Tübingen.

Moore, Johanna D. (1995): *Participating in explanatory dialogues.* Cambridge, MA.

Schneider, David A./McCoy, Kathleen F. (1998): "Recognizing syntactic errors in the writing of second language learners". In: *Proceedings of International Conference on Computational Linguistics (Coling-ACL 1998)*: 1198-1204.
http://acl.eldoc.ub.rug.nl/mirror/P/P98/P98-2196.pdf

Sterling, Leon/Shapiro, Ehud (1986): *The Art of PROLOG: Advanced Programming Techniques.* Cambridge, MA.

Warschauer, Mark (1996): "Computer-assisted language learning: An introduction". In: Fotos, Sandra (ed.): *Multimedia language teaching.* Tokyo: 3-20.

Yazdani, Masoud (1988): "Language tutoring with Prolog". In: *Papers of the Int. Workshop Intelligent Tutoring Systems for Second Language Learning*. Trieste: 150-155. Reprinted in: Cameron, Keith (ed.): *Computer assisted language learning*. Oxford: 101-111.