Developing Natural Language Interfaces: a Test Case

Gert Veldhuijzen van Zanten and Rieks op den Akker
Department of Computer Science, University of Twente,
P.O.Box 217,
7500 AE Enschede

Abstract

Dutch is the best language there is. For the Dutch that is. Therefore they prefer to think in Dutch, not in a foreign language like English, SQL or some other computer language. Instead of learning people to use a strange language for communicating with machines, it may be preferable to learn machines to communicate with Dutch people in Dutch. This has led to the idea of making something "in between man and machine": an interface to bridge the gap between natural language and machine language. How do you make such a bridge and what kind of things do you need for making it? If you can't make a Natural Language Interface for data base querying there are more things you can't do. Making an NLI: a test case for the ability of making more sophisticated natural language processing systems.

1 What is a Natural Language Interface?

A natural language interface for a particular data base allows people who want to extract information from the data base to use their own "natural language", the language they think in and in which they express their questions immediately, i.e. without an explicit process of formulating. Is it possible to make a natural language interface for data base querying by typed queries? That is: an NLI the intended users can really use[1]. Maybe it is true that the ideal NLI allows its users to query the data base by speaking to it. But since you can't make a speech understanding NLI if you can't make an NLI that handles typed queries – although in written text we lack information provided by prosodic variations in spoken utterances – we first leave speech recognition out of discussion. We also assume an NLI is made for users using one natural language, Dutch let's say.

Being toolmakers we are interested in making tools, things that can be worked with effectively and efficiently and that make the work easier to do than without them. Of course as mathematicians, linguists, and/or computer scientist we love formal languages, grammars, and we like to write nice, efficient algorithms even if we haven't the slightest idea for what practical purposes they could be used. These things have their own intrinsic and attractive values. Maybe this is the main reason that in an academic environment it never comes to the making of a true NLI. (An informatics department of a technical university is something different as a pure mathematics or linguistics department at a university. The former one should have an open eye for the practical applications of their products.) A welcomed aspect of NLI's is that it confronts us with many interesting problems that need scientific research of very different kinds in order to be solved. For instance we need parsing theory, results from linguistic research, formal logics and software

[1]We distinguish between a "prototype" of a system that translates natural language sentences into formulas of a data base query language and a useful natural language user interface. Most prototypes, how promising they may be (like PHLIQA1 for instance [Phl79]) never reached the stage of a real NLI.

engineering methods. There is clearly a research management problem here that need to be solved before you can make a technological tool like an NLI. And there is a problem of financing. We will, however, abstract from these kinds of problems here.

If it was clear to us that practical NLI's for particular data bases are impossible this paper wouldn't have been written. The theorem

> Natural language interfaces for data base querying are impossible

cannot be proved in the mathematical sense[2]. This doesn't mean that you can't be convinced of its *correctness*. Some people are maybe more convinced of the impossibility of Artificial Intelligence than about the fact that 1 and 1 makes 2. Maybe they have the opinion that there is no thing "in between natural language and formal language" because these languages are creatures from heterological universes. There is definitively a point in this. But, this is not the time and place to hear philosophers. We try to explore the borders of language technology by trying to construct the bridge. The methods to prove the *incorrectness* of the theorem are both constructive. Either show one or, if you can't find one, make one. Since we didn't find one, (to be honest we didn't spend much time in searching one), and since we are not only interested in the product itself but also, and even more, in the process of making one (since this is the only way to become known about all finesses of how it works as it works) we decided to make one. NLI is a first test for how far we are in making computers understand people using natural language. It is a first test. Our interest in making an NLI for a data base doesn't stand for itself. We will use knowledge, techniques, experiences, contacts, and methods gained and developed for succesfully passing this test in the Schisma project (see the contribution about Schisma in these proceedings). If we don't succeed in passing the test we can always say that it is because of bad management or lack of finances. Or maybe we will find ourselves convinced of the fact that there is something wrong in principal with that "in between natural and machine language" or with the idea that this "in between" can be implemented as a "technological interface" between man and machine. We don't ignore the possibility that there might be something principally wrong with this idea.

In pondering about NLI's we should answer the question whether a natural language, like Dutch, is also a "natural" language for man-machine communication. Of course the answer depends on what you still consider to be Dutch, or to be "a natural language" in general. The fact is that people adapt their language use to the ones they are communicating with. In communicating with machines people don't have to show how well-behaved or well-educated they are, although some users may act as if they are communicating *by means of* the machine with someone inside or behind it. If a user of an NLI happens to experience that typing or uttering the phrase "employees Philips Research Eindhoven" (not a sponsor) has the same effect as typing the full sentence "Please show me all names of employees that work at Philips Research Department in Eindhoven", he will definitely prefer the shorter one in communicating with a machine, but most likely not in communicating with someone at Philips Research. In man-machine communication it is the effect of the question that counts, and nothing else. And if you're sure that saying *yep* to a machine has the same effect as saying *Please start up the database* you will say *yep*. Hence, in men-machine communication, more than in men-to-men communication, efficiency is a dominant factor in evaluating the communication language that is (to be) used. But does this mean that we are arguing in favour of using a formal language

[2] A mathematician could come up with a functional specification of a natural language interface, calling this function *nli*. The function *nli* can be either possible (in the sense of computable) or not. But a function is not a practical tool.

instead of a natural language? No. Although every NLI is based on some (formal) *model* of a natural language, we still believe the user of a data base should not be bothered with learning a formal language before he or she can phrase his first question. That is why the model on which the NLI is based should be a good model of a *natural language*. It means that an NLI should adapt it's language in communicating with the user. So, starting with a clear fragment of proper Dutch, the NLI and the user should be allowed to propose short phrases for complete sentences. Once both parties agree upon the use of these short phrases, having a well defined semantics in terms of a complete sentence, they can from that time on be used for the same purpose as the complete sentence. A natural language interface should have an adaptive natural language in order to be practical. It is often remarked that an NLI shouldn't stand on its own. It should be a part (mode) of a *flexible* multi-media interface. ("Flexible" in the sense that it is up to the user to decide in what form he wants to provide his information: by speech, by pointing at objects or by typing.) We agree with this but in this paper we will consider single-mode NLI's for data base querying. Moreover, we assume no references are made in questions to former queries or answers to former questions. Until now we assumed that the NLI leaves the user free in the way he formulates his question. A lot of problems introduced by this freedom can be "solved" by a syntax directed NLI. Such an NLI will show the user, each time he has typed a word, the possible continuations of the typed prefix. The possible continuations can be offered as a list of possible categories of words or phrases or in terms of concrete words (belonging to closed word groups, like *what, where, how, how many*) and the user can select the category or word he wants to use. The user should of course not be bothered with grammatical categories; not with NP's. Some typical examples of categories allowed and covered by the interface could be offered. The ordering of possible continuations offered can be dynamically updated depending on the history of use; either the last used continuation, or the most frequent used one is listed first. For uninitiated users and not dayly users an NLI that has such a syntax directed mode may be quite handy. Experiences from real users will decide whether this is really true. Existing tools for the generation of syntax-directed editors may be used for this mode of an NLI.

In this paper we present our experiences, interests, problems, and our plans for passing the test. We discuss several aspects of NLIs for data bases (like robustness and quality) and of the process of making them. Especially a parser for unification grammars [Shi86] and a tool for developing the front-end part of NLI's (i.e. the natural language parser) will be discussed in some more detail. The paper is organised as follows.

2 Stages of Handling Users Input: a categorization of users input according to the various reactions of the NLI. How do we handle typing errors?

3 User Requirements: what does the user expect from a practical NLI? (not completely hypothetical)

4 The Kernel of an NLI: in which the grammar and the intermediate languages between the natural query language and the data base query language are discussed.

5 Developing a Grammar for an NLI: how do we make a grammar, what is a good tool for developing one, and how do we specify the meaning of a natural language query? (presenting some ideas to be worked out in the future)

6 Left-Corner and Head-Corner Parsing: why? and how does it work? (not formal either but there are references given to more detailed specifications of this parser.)

7 Efficient Unification: how do we unify feature structures? In which a table is presented with experimental results from a

comparison between our unification algorithm and the best one we found in the literature.

8 Robustness of NLI's: what could we mean by this buzz word? Are stochastic methods useful for obtaining more robust natural language processing systems and why not?

9 Plans for the Future: what should we do without them?

2 Stages of Handling Users Input

Before we come to formulate user requirements for NLI's in terms of systems performance, we must categorize typed user input with respect to the possible reactions of the system.[3].

A user types a sequence of tokens. Sometimes this is a perfect sentence, but sometimes it contains typing errors. Typing errors can be destructive for people who read the "sentence", they can also be nondestructive to people, in the sense that human readers have no problem in recognizing a sequence of words and separators and punctuation marks. Typing errors sometimes are information destructive. They lead to a diversion in meaning among human readers about what the user intended to write. A typing error may result in an other word than the word that was meant. Such an error is not destructive in the sense defined above, but it is information destructive if it is not clear using only grammatical knowledge what the intended word was. Consider a user who types "female" instead of "male". This can be, of course, because of misthinking. Such an error does by itself not lead to an incorrect sentence and we cannot expect from a system that it recognizes such kind of errors, which are not typing errors. We cannot even expect that from human readers. But if a user types "fmale" we have to do with a destructive error because by itself we cannot know whether "male" or "female" was meant. However, in the sequence "male or fmale" the human reader will recognize the intended phrase "male or female". Even in the sequence "male or male" the human reader may recognize the intended phrase "male or female" or "female or male". There is semantic knowledge involved in recognizing this.

If a user types a sequence of tokens the system should correct nondestructive typing errors and show the user its result asking whether this is what he/she intended to write. The user can either confirm or correct the query. At this *first* stage only lexicographical word knowledge is used, no grammatical or semantical knowledge. The user seeing what he has typed has the possibility to correct his sentence. The result of this first stage is a sequence of tokens, t, not containing words that do not belong to a well defined set of words D. This set D is a superset of the words that the system can handle lexicographically, grammatically or semantically, and it may be a proper subset of the set of all English words. The token sequence t may contain spelling errors or ambiguities the user isn't aware of. This sequence is fed into the *second* part of the system, which is the morphological analyzer. No grammatical or semantical knowledge is involved in this process either. The result of this stage is a sequence of words, with their morphological analyses. There is no feedback to the user in this stage. (For the realization of this stage we can use results obtained by Vosse in making a Dutch spelling corrector, see [Vos94]). In the *third* phase words are looked up in the lexicon, a set of words with syntactical and semantical information. We will return to syntactical and semantical information later on. If a sentence contains a word w that cannot be built from a word in the lexicon at this stage the system responds to the user that it doesn't know the word. Let S denote

[3]Although Dutch is the best language for us, and although we intend to make a Dutch NLI, for the sake of communication we will adopt a kind of English in this paper; in the language used as well as in the examples.

the set of all sentences accepted by this stage for further processing. Sentences in S are fed into the following, the *fourth*, stage in which the sentences are analysed grammatically. A sentence will be called *correct* if it can be analyzed with respect to the formal grammar on which this parsing phase is based. [4] Correct sentences can be semantically ambiguous or unambiguous. If a sentence is not correct the system will respond to the user that it cannot handle the query: "I don't understand what you mean by this query". Ambiguities (i.e. semantic ambiguities– syntactic ambiguities that don't lead to semantic ambiguities fall outside this class) are reported to the user. The user can either be asked to choose from a small set of possible readings or to rephrase his question. Sentences that reach this stage of the process are correct and semantically unambiguous. They form the set U of correct unambiguous sentences. Notice that "correct" doesn't mean correct English. The "natural language" the NLI can handle may, and preferable will, contain sentences that are not correct English. (As we mentioned in our introduction, this language will cover some short sentences that don't have to be grammatically correct English, they are "private" formulas. And the system will also not stumble over spelling errors if they don't introduce ambiguities.) Not all sentences in U can be answered by the system. It may contain phrases like "the price of the employees" the system does not know how to handle since although it knows of prices and of employees, the type of the semantic entity the word "price" denotes is not consistent with the type of the entity the word "employee" denotes. These type restrictions not only depend on the linguistic category of the words (employees may have prices) but also on the semantic domain of the data base: in some domain the phrase may have a well-defined meaning. Type information may resolve syntactic ambiguities: "the names of companies that have more than 100 employees". The set US contains all sentences that are in S and also well-typed with respect to the semantic domain. Sentences that are in S but not in US will be answered by: "I can't answer this question".

The performance of an NLI can partly be given in terms of the relation between users inputs that result in a sentence belonging to the set S after the stages mentioned above and the systems response. A system *responds correctly* to a typed query in natural language if the response of the data base is the same as that of the system that is fed by the formal query found by experts who translated the natural language query into a formal query without using additional information to the information expressed in the users input. This definition (similar to the one found in [Bov93], discussing an NLI for the ATIS domain) can be used for obtaining an objective measure for measuring this aspect of the performance of the NLI. (In order to cope with the problems introduced by the informal notion of "information expressed in the users input", we should ask a number of experts to translate the natural language query into a formal query.) This makes it possible to compare different NLI's for a fixed data base. Comparing NLI's for data bases covering different domains is far more difficult, however, since the complexity of the formal model of the natural language part (defining syntaxis as well as semantics) depends on the complexity of the domain.

3 User Requirements

An NLI for information querying from a particular data base should meet the following user requirements.

1. The user doesn't have to know about the organisation of the data base, only about the kind of information it can provide in order to query it by using a natural lan-

[4] If one conclusion from experiments with commercial NLI's for data bases can be made it is that partial analyses (keyword recognition) of user input is simply out of the question if you want a robust system (see [Sij93] for a review of "commercial" NLI's for data bases).

guage.

2. The natural language provided by the NLI is
   (a) reliable: the answer to a question resulting in a sentence that belongs to the set US is the same as to the question phrased by experts in a formal query language (SQL lets say).
   (b) information-complete: all information that can be obtained by a formal query can be obtained by some natural language query.
   (c) broad covering: most natural language utterances used by an uninitiated user, who knows only about the kind of information stored in the data base, are covered by the natural query language.
   (d) adaptive: the NLI offers the possibility to communicate about the language that may be used for phrasing his questions.
3. The NLI doesn't accept semantically ambiguous sentences or phrases, but reports them to the user, offering alternatives or the possibility to rephrase his query.
4. An NLI should be robust. That is:
   (a) the NLI doesn't stumble over non-information-destructive typing, spelling, or grammatical errors.
   (b) The NLI accepts syntactically ambiguous sentences if they are not semantically ambiguous.
   (c) If a user request contains words (like *flower* or phrases like *beating about the bush* the NLI doesn't cover, because they haven't anything to do with the conceptual domain of the contents of the data base, it gives a message like "I don't know about *flowers, bushes*".
5. The time spend by the NL front-end on processing a well-phrased question is less than a small constant times the time spend by the actual querying of the data from the data base.

Requirement 1 will be clear. Some prototypes of NLI's expect from the user that he knows names of the tables of a relational data base. In our view such systems ask too much from the user. The user should of course know something about the data base: its contents, the kind of information it contains, but not about things like tables, keys, which have to do with the organisation of this contents. Notice the difference between requirements 2a) and 2c): a useful NLI should be 100% reliable (is 99% also reliable?), 2c) coverage has to do with how many percent of the sentences typed in by the users reach the stage of being answered properly, at one of the stages or by a reliable answer. We have already given arguments for requirement 2d) in the Introduction. If a user uses similar sentences quite often, the NLI should offer him the possibility to shorten the sentence. This can be implemented by storing questions, or schemes of questions. So the user may write "names of kids of Johns son" instead of "What are the names of the kids of Johns son?" In a further section we will come back to the issue of robustness. Sometimes requirement 5) is sorted under robustness-requirements. We don't because it depends too much on other things, like efficiency of algorithms, clever choice of data structures, and properties of the software and hardware environment in which the NLI operates, than on the method implemented for handling the users language proper. Efficiency is however an important aspect of performance: a system may use, for instance, some sophisticated method for semantical error correction but if the user has to pay an extra few minutes waiting before he gets his answer the tool will probably be put on the shelf. We will return to efficient parsing and translation later on.

4 The Kernel of an NLI

It will be clear by now that we have a linear architecture for the NLI in mind. The processes discussed before filter the input and sentences that pass these filters are fed

to the kernel of the NLI. These first four stages allow us to choose for online word by word processing, although some typing errors may destroy the intended word boundaries. The grammatical and semantical kernel of the natural language interface will be split up in two translation phases at least. The front-end translates NL into some Intermediate Language (IL) and the back-end translates IL into SQL, or some high level logical/knowledge representation language for data base querying. This makes it possible that the grammar, the lexicon and the IL become to a large extent independent of the contents and the organisation of the particular data base. The front-end becomes reusable for other data bases. This hasn't so much to do with the performance of a particular NLI for a particular data base, but more with the management of the process of developing an NLI and– not unimportant– the reusability and extendibility of parts of it. Moreover, further research and development may be directed towards the generation of NLI's for particular data bases using fixed parts of the grammar, the lexicon and the intermediate language(s).

Several candidates could be nominated for the price of the best intermediate language: QLF (the Core Language Engine), WML (from PHLIQA1), PTQ (Montague Grammars), to name a few. The Quasi Logical Formalisms used in the CLE project are not specifically for the semantics of natural query languages for data bases, making them quite, maybe too heavy for this purpose. The same holds for all other "general purpose" quasi logical languages. One characteristic of QLF [Als92] is the notion of event and state, event-variables and state-variables that allows to abstract over actions or events, and formulate higher order predicates for the representation of the semantics of phrases like "living in Paris is nice" in a systematic way. The World Model Language used in PHLIQA1 is an intermediate language between the English-oriented Formal Language (EFL) and a data base query language. EFL is completely independent of the subject domain (not considering the limitations of the lexicon; they are of course domain dependent). Hence references of words or phrases to objects in the semantic domain are not made in translating the natural language sentences into formulas in EFL. This is done in the second phase: the translation from EFL into WML. This "multi-level" approach clearly distinguishes the several steps in finding the proper semantics of a natural query. Adopting this distinction not necessarily implies that each step is implemented in separate modules that proces in line as it is done in PHILQA1. Unification grammars (we return to unification grammars in a later section) allow to specify the several aspect of semantics (rules or constraints that are "linguistic" and rules that are dependent on the domain) in one and the same formalism. The price to be paid is that words may occur more than once in the lexicon; one occurrence for each semantics. EFL assigns only one constant to each word. Disambiguation is done in a later phase, consisting of the translation into WML and the interpretation of WML-expressions. Most IL's proposed contain lambda-constructs for higher order predicates and set-denotation, and generalized quantifiers for denotation of phrases like "most employees" and "some department". For a motivation of using generalized quantifiers in the IL for a Dutch NLI for data base querying see [Spe93]. Experiments with IL's will have to decide what is an appropriate IL for data base query languages. We have chosen to use context-free grammar rules as the basis for specification of the semantics, and since the semantics of a sentence has to be defined compositionally, this choice may restrict us in the choices we have for an IL.

5 Developing a Grammar

The kernel of a Dutch NLI is based on a grammatical model of the Dutch language. You may think that there is someone somewhere who has such a thing. And indeed

there are but they can't offer you what you want. Either grammars are specified in a dreadful formalism it takes a year to comprehend, or the parser that goes with it, (sometimes the parser is the only "grammar") is written in Prolog or Lisp and after running a complete weekend on "list all the names of employees, that work in London" it outputs 135 parses. (There are people who have similar experiences, see [Ter93] and [Vet94].)

It should be remarked that we are not primarily interested in a "principle-based" grammar, a formalization or implementation of some linguistic theory, like GB. Since we are not primarily interested in describing a natural language, or the linguistic competence of an ideal language user, who knows the grammatical rules of Dutch. Nor are we interested in a "mentalistically motivated" theory of parsing, what ever that may be. We are only interested in methods, techniques and formalisms for the generation of robust NLI's. We decided to write a grammar. Since this is quite a job a tool for developing grammars is needed.

We decided to use our Head-Corner Chart Parser for unification grammars, as the basis for developing the tool for making the front-end. (We call this tool HCP, as long as we don't have a better name.) There is no better reason for choosing this parser than that it is a product of "our own", and that it is the best parser for context-free unification grammars available to us. What do we expect from a tool for developing a specification of a translation from a natural language into some intermediate language, when this translation is to be used for a natural query language? Here is our list of demands.

- The specification language for the grammar and the lexicon should be easy, high-level and declarative.
- The tool offers good diagnostics of syntactic errors in the specification.
- The parser should be fast
- The tool offers facilities to trace the parser for debugging the grammar.
- The tool offers the possibility to test modules of the grammar, like the part of the grammar for NP's.
- The tool offers a standard back-end language for semantics.

In [Ned92] the authors describe GWB, a grammar work bench developed at the Univerity of Nijmegen for affix grammars. HCP is not a grammar work bench in the sense GWB is. HCP doesn't offer the grammar writer technical information about the grammar he writes (like look-ahead sets of non-terminals). Maybe the tool should also be able to generate sentences, a functionality offered by GWB. But, it should be remarked that grammars for NLI may perfectly well overgenerate, or better there is no proper notion of "overgeneration" here (things change of course if you make a syntax-directed NLI based on the grammar).

HCP uses a unification grammar as a specification language for both the syntaxis and the semantics. This language is similar to PATR, except that it has a true context-free grammar backbone. The rules of which have left- and right-hand side symbols with associated feature structures. These structures are defined by a set of feature equations. Feature equations serve two purposes: they constrain the use of a rule in a particular node of a parse tree (static semantics, type-checking, agreement checking) and they are used for building the "semantic feature values" of the nodes of the parse tree from other feature values. These two purposes correspond to the two possible results of unifying the left-hand side and the right-hand side of a feature equation. Either unification is not possible, or unification results in a more informative feature structure. The lexicon is a set of words with associated feature structures.

A tool for developing a front end for natural language interfaces for data-base querying, should not only offer its users a specifi-

cation language for the syntaxis of the natural language, but also a target language, that can be used as an intermediate language for expressing the semantics of the natural language sentences. Notice that the formalism of unification grammars leaves the user free how to specify the meaning of its syntactic constructs. The target language should be general enough to serve as an IL for all kinds of data base contents. On the other hand it isn't necessary to translate all possible meanings of sentences of a natural language into it. We are trying to describe the contents of data bases, not the real world. This means that we "only" have to formalize the syntaxis and meanings of that "part" of our natural language that is used for data-base querying. So, we don't need a complete grammar for Dutch. And we don't want it either because a too large grammar would lay a great burden on the parser.

The functional meaning of a question (i.e. its result) is its answer. We distinguish the *functional* meaning from the *core* meaning of a sentence. The core meaning of the sentence "Which employees are working in Amsterdam?" is the set of all employees that form the semantics of the noun phrase "employees, that work in Amsterdam". The sentence "List all people, that work in Amsterdam." has the same core meaning, although for us it may have a different "meaning". Yes/no-questions like "Does John work in Amsterdam?" will have the same core meaning as its imperative form "John is working in Amsterdam". It is the *mood* of the sentences that makes the difference in their functional meanings, the answer to the question. Sentences like "Are there any men, that work in Amsterdam" will have as core meaning the meaning of the noun phrase "men, that work in Amsterdam". The conclusion is that, apart from the mood of sentences (Yes/No questions, WH-questions, declarative and imperative sentences) we have only two different kinds of sentences. Sentences that have as their core meaning the denotation of a noun phrase (Object Valued Sentences, OVS), and sentences that have a statement as their core meaning (Truth Valued Sentences, TVS). Similar distinctions can be found in the PHLIQA1-report [Phl79]. A sentence will have three semantic features.

- mood
- core, and
- assumptions

The feature assumptions will have as value the implicit assumptions concerning the existence of objects in the world of the data base. If we ask for "the brothers of John" we implicitly assume that there is a person in the data-base world whose name is "John". Using the noun phrase "the father of John" we also assume that there is someone who is the father of "John". Remark that this assumption is not made if we say: "Is there anybody named John?". The latter question should in fact be properly phrased as: "Is there anybody named 'John'?", and the kernel of a robust NLI should be able to see that this is meant. By distinguishing the assumptions from the core meaning we are able to generate better answers to questions: first the assumptions are validated and then, if all assumptions are true, the answer is produced.

Our IL thus has terms for denoting objects as meanings of noun phrases and OVS's, and assertions denoting truth-values for the meanings of TVS's. A grammar writer uses categories like NP, VP, PP and so on. The tool should provide the writer with standard feature structures for these categories. The same holds for the lexical entries: the category of an entry fixes its feature structure.

6 Left- and Head-Corner Parsing

In this section we explain the parser which is used in HCP and which will be used in the NLI as well. A head grammar is a context-free grammar in which each rule is assigned

a head. If a rule is S→NP $\widehat{\text{VP}}$ then VP is the head. S and VP are in the head-corner relation. The head-relation is the transitive closure of this head-corner relation. It is up to the grammar writer to assign heads to the rules of his grammar.

Suppose that S is the start symbol of the grammar, following a head-corner strategy, a parser starts with an assumption—the words on the input form a sentence. This is notified by recording on the chart that S is a goal symbol of the grammar for the whole sentence. Given this goal symbol, the parser determines the possible heads of the sentence[5]. Such a head is a word in the sentence that has a category C that is in the head relation with S. Then all rules are searched for that satisfy the following two conditions:

a) the head of the rule has the category C of the word selected, and

b) the left-hand side of the rule is in the head-relation with the start symbol.

Suppose that A→B $\widehat{\text{C}}$ D is one of the selected rules. A double dotted item of the form [A→B • C • D; i, j] is added to the chart, recording the fact that C has been recognized, i.e. C derives the part $a_i ... a_j$ of the sentence[6].

From this item, we derive two more goal symbols—B and D. These will play the same role as the start symbol in the beginning. The same proces starts but now possible heads are searched for in those parts of the sentence that are to the left (for B) and to the right (for D) of the recognized head with category C.

If B or D has been recognized, then an item is generated in which one of the dots is moved over the recognized symbol. When both dots are at the ends of the right-hand side, then the whole right-hand side has been recognized and thus, the left-hand side symbol A will now play the role of a recognized head.

When the item [S→ •α•, 0, n] is on the chart, then the sentence $a_0 ... a_n$ has completely been recognized and a complete parse has been found.

For a more formal specification of the head-corner chart parser we refer to the papers published elsewhere, like [Sik93, SiA93].

The symbols in the items have associated feature structures and during parsing these structures are built by unification according to the rules in the grammar and the lexicon. Since the number of items can be quite large, the space and time efficiency of the parsing is strongly influenced by the unification algorithm, and by the choice of the data-structures for the chart. In the next section the unification method is considered in some more detail.

Head-corner parsing can be seen as generalisation of left-corner parsing. In the sense that in left-corner parsing, parsing always starts with the left-corner that has been recognized. The advantage of left-corner parsing comes from the fact that it parsers strictly from left to right. Therefore only one index is needed whereas we need two in the head-corner parser. So why should we follow a head-corner strategy? A motive for head-corner parsing is that heads of rules and heads of sentences are those parts of the sentence or the tree that offer information on the basis of which possible rules can be ruled out if you look at the feature-constraints. This advantage, however, only pays off if heads are selected in a clever way. In the current implementation of the head-corner parser there is no top-down filtering by feature-unification.

[5]For each word w in the sentence, and possibly for some combinations of subsequent words in the sentence also (idiom), the lexical analyzer creates one or more (in case of multiple occurrence of a word in the lexicon) start items [i, F, j], where F is a feature structure for the word w if w occurs between positions i and j in the sentence. These start items are in fact initially put on the chart.

[6]If a lexicalized unification grammar is used, with more general trees of depth one in the lexicon, the lexical analyser will put for each word in the sentence this kind of double dotted items on the chart where C is a possible category of the word. Hence, the parser can not only be used for traditional context-free grammars but also for grammars that are completely specified by the lexicon.

This could be possible if symbols that are in the head-relation share their feature structures. Top-down filtering is only done by looking at the head relation. Therefore for most grammars we have seen left-corner parsing is more efficient than head-corner parsing.

The parser doesn't output parse trees but feature-values of the sentence (of course if the grammar writer wants he can define the parse trees as feature values). This implies that sentences that have more than one parse-tree but only one meaning in terms of features values, have only one output structure. Hence, also in case of cyclic grammars we may have only one output structure for an infinitely ambiguous sentence.

7 Efficient Unification

Feature structure unification is generally considered to be a very expensive operation, and, in many implementations of parsing systems it takes more than 80% of the total parse time. Therefore, we have taken special care to search for efficient implementation of this part of the parser.

In several papers, typed feature structures have been proposed to improve efficiency. The idea is that type checking can be done relatively fast and preempts many unifications that would have failed anyway.

In our current implementation, we haven't implemented
typed feature structures, mainly because we haven't had the time. Furthermore, we are using a context-free parsing strategy on top of the unification grammar. Therefore, the top-level feature structures do have a type in the guise of the grammar symbol attached. So, we do have some of the advantages of typed feature structures in our current system. In the future, we probably will implement typed feature structures, for there are many advantages other than efficiency.

The bulk of effort in unification is to do with copying of feature nodes. In a chart parser, we cannot use destructive unification, and therefore a copying scheme must be deviced. Tomabechi [Tom91] states that copying should be prevented for unifications that fail and describes how this can be achieved, without much additional overhead. His strategy is called quasi-destructive unification, and that is exactly what it is. When unifying two feature graphs, the result is built on top of one of the arguments. This is done in such a way that features from the second argument that are not in the first, are added to the first argument, in such a way that they can still be distinguished as being added in the current unification. If unification fails at some point, then the process is simply aborted and the additional features in the first argument are thrown away by a clever increment of a global generation counter. If unification succeeds, however, a copy is made of the first argument before the generation counter is incremented. This copy serves as the result.

Tomabechi's algorithm is considered to be quite efficient, but its virtues have been underestimated, as it allows for a slight modification that results in a major improvement in efficiency of our parser—when copying the first argument, we can share most of the substructures of the arguments. And by doing so, we have actually succeeded in decreasing the number of feature nodes used in the parser by a factor 4 to 10, in some example grammars. Also, because of the reduced copying, the cost of unification is significantly less than in other known implementations. An article in which this unification algorithm is described is forthcoming.

The tables show results from two small experiments for comparing our unification algorithm and the one from [Tom91]. The columns of the first table show: the number of the sentence (see below), the number of nodes created using our unification algorithm, and the number of nodes created using Tomabechi's algorithm, respectively.

nr	HCP	TOM
1	43	526
2	60	894
3	43	520
4	104	1279

In this experiment the following 4 sentences were used.

1. *Waar wonen de broers van Jan?*
2. *Wie zijn de zussen van de vader van Marie?*
3. *Hoe heet de vader van Marie?*
4. *Is de zus van Jan de tante van Marie?*

These 4 sentences were parsed with respect to a grammar for a small fragment of Dutch, having 30 context-free rules, with 5 features rules per production in the mean, and 3 right hand side symbols in the mean. The lexicon contains about 100 words. All these sentences are processed by HCP in less than a second.

n	parses	HCP	TOM
4	5	100	542
5	14	249	1.827
6	42	662	6.268
7	132	1897	22.187
8	429	5799	80.685

The second table shows results from parsing the "sentences" Jan^n, for $n = 4, 5, 6, 7$ and 8, respectively. These sentences were parsed using a grammar with the two rules $S \rightarrow S\ S \mid Jan$, so these sentences are rather ambiguous. The semantics defined in this grammar associates the parse trees to the sentences. The second column of this table shows the number of parses. For the last sentence, i.e. Jan^8, HCP outputs the 429 parse trees within less than 30 seconds.

8 Robustness of NLI's

More and more people seem to get convinced of the fact that even the most sophisticated model of a natural language can only partly account for the rich nature of natural language. Sooner or later a user types in a sentence the system can't handle in a correct, meaningful sense. This has led to the interest in what is now called "robust parsing". There is a lot of knowledge involved in the process of grasping the meaning of a question or an utterance as it was intended by the user. We use knowledge of the context, the dialogue for instance, we use also our knowledge about the speaker, in order to filter the relevant information from the utterances. Also filtering of noise in spoken language is an aspect of robust processing. Robust parsing aims at incorporating these different "knowledge recources" in finding the appropriate (intended) meaning of the utterance. The interest in (and the urge for) robust parsing comes from the principal shortcomings involved in any process of formalisation of natural languages and their use. In section 3 we have formulated some robustness requirements for NLI's. How can these requirements be met? What kind of solutions have been proposed for dealing with the problem of lack of robustness in existing NLI's?

Some people have advocated the use of statistical data in order to get more robust analyses of utterences. Hence, the interest and motivation for studying stochastic grammar models, Hidden Markov Models, and statistics on the occurrences of trigrams of syntactical categories. It is not clear whether these stochastic models could offer a solution for the robustness problem.

> "... on their own PCFGs [i.e. probabilistic context-free grammars] probably are not very useful in syntactic disambiguation.",

Charniak says in his book on statistical language models, [Cha93]. This also holds for more sophisticated probabilistic grammar

models of natural languages than the classical models studied in Charniak's book, as for instance DOP [Bod93] or weakly restricted CFGs [Doe94]. You need semantic information in addition. And very large corpora!

Sometimes the problem of robustness is almost identified with the problem of "overgeneration", the phenomenon of a parser that presents 135 parses for a simple sentence that has only one meaning (or maybe two). Stochastic grammars can be used to deliver the most likely parse of a sentence. But sometimes you don't want the likely one, and if you always want this one why not remake the grammar, so it only outputs this one? Overgeneration is not the most important problem and it isn't even a robustness problem in itself. We reserve the term robustness to describe a quality or property of a system (like a natural language interface)—how well does it behave in case of unpredicted input. In that sense, you cannot say that a grammar that produces 135 analyses for a simple sentence is not robust. It depends on what the grammars is used for. A user of an NLI will say that it is not robust, if he types in a good sentence but the parser cannot recognize it or, he makes a typing error or a grammatical error and the parser cannot recognize this error and restore it, while human beings have no problem in recognizing the intended meaning. One can, of course, extend the formal grammar so the parser can handle more correct sentences and also most frequent typing errors and grammatical errors, but this is unfeasable for larger domains.

A solution to some problems of robustness consist of skipping unparsable parts of a sentence. Then the problem is to find a most complete parse of the sentence, i.e. a partial parse that covers most of the sentence. The idea is that this partial parse may offer enough information about the semantics of the sentence. To this end existing parsers are "made more robust": they skip unparsable parts of the input and resume parsing as soon as possible. Pure bottom-up parsing, like GLR-parsing (Tomita) or a generalisation of CYK-chart parsing for general context-free grammars, is the best strategy for this kind of robust parsing, although the results are not very promising for reliable NLIs. Head-corner parsing is bad for obtaining the best partial parse because only heads that are predicted top-down are recognized. The same holds for left-corner parsing. They don't build as much incomplete parse trees as a pure bottom-up parser. We think that for particular domains we can use head-corner parsing and extend the grammar for dealing with common "grammatical" errors that do not introduce semantical ambiguities.

9 Plans for the Future

We will continue to work on developing an NLI and on HCP, the tool, simultaneously. HCP is being used for developing natural language front-ends for database querying for educational purposes. Problems discussed are: how do you specify a small fragment of a natural language by means of a context-free unification grammar and how can you specify the semantics of the sentences of this fragment in a systematic way? Special attention is paid to reusability and extendibility of the grammar. The system is also used for the Plinius project at our department. This project aims to develop a system able to acquire knowledge semi automatically from natural language abstracts about the mechanical properties of ceramics. For the reasons that have led to choose for using HCP and for more information about the Plinius grammar and this project in general we refer to [Vet94].

HCP is not in its final form, although rather efficient it hasn't a user-friendly interface yet. The grammar writer is not supported in writing well-structured orthogonal grammar specifications. Using a unification system with typed feature structures could be a good step in this direction. Standard types for agreement features and even some standard typed semantic features (related to a standard intermediate formalism)

can help the grammar writer in writing well-typed and well-structured grammar specifications. As a side-effect the introduction of standard typed feature structures may have a positive effect on the efficiency of unification ([Car91, Car92]). As far as the specification language is concerned, the tool doesn't support disjunct feature structures. The question is whether the disadvantages of disjunct feature unification (complexity, [Kas87], [EiD88]) outweights the advantages for the grammar writer. HCP not only allows us to develop grammars but also to study several candidates for the intermediate language to be used in an NLI.

There is no morphological analyser, no error-correcting phase implemented and so in using small prototypes currently all word forms have to occur in the lexicon and have to be typed in without errors.

Although we haven't said anything about answer generation it is an important part of a practical NLI. Also this part will be the subject of further investigation. A next step towards a natural language dialogue system then consist in looking at problems introduced by offering the user the possibility to refer to phrases mentioned in a previous query: "can you give me more information about these things?"

References

[Als92] Alshawi, H. (ed.) (1992), *The Core Language Engine*, The MIT Press, Cambridge, Mass., 1992.

[Bod93] Bod, R. (1993). Data Oriented Parsing as a General Framework for Stochastic Language Processing. *Twente Workshop on Lang. Techn. 6* (*TWLT6*), 37–46.

[Bov93] Boves, L. (1993). Spoken Language Systems: An Overview. *Twente Workshop on Lang. Techn. 5* (*TWLT5*), 9–13.

[Car91] Carpenter, B., C. Pollard and A. Franz
(1991). The Specification and Implementation of Constraint-based Unification Grammars. *Proc. Second Internat. Workshop on Parsing Technol.*, 143–153.

[Car92] Carpenter, B. (1992). *The Logic of Typed Feature Structures.* Cambridge University Press.

[Cha93] Charniak, E. (1993). *Statistical Language Learning.* A Bradford Book, The MIT Press, Cambridge Mass., London, England.

[Doe94] ter Doest, H. and R. op den Akker (1994). Weakly Restricted Stochastic Grammars. *15th Internat. Conf. Comput. Ling., COLING'94* , 929-934. (see also: Ta! nr. 4 jaargang 2, 84-98)

[EiD88] Eisele, A. and J. Dörre (1988). Unification of Disjunctive Feature Descriptions. *Proc. 26th Annual Meeting of the Association of Computational Linguistics*, 286–294.

[Kas87] Kasper, R.T. (1987). A Unification Method for Disjunctive Feature Descriptions. *Proc. 25th Annual Meeting of the Association of Computational Linguistics.*

[Ned92] Nederhof, M.-J., C.H.A. Koster, C. Dekker, A. van Zwol (1992). The Grammar Workbench: a first step towards lingware engineering. *Twente Workshop on Lang. Techn. 2* (*TWLT2*), 103–115.

[Phl79] Bronnenberg, W.J.H.J. et al. (1979). The Question Answering System PHLIQA1. In: L.Bolc (ed.), Natural Language Question Answering Systems. (Natural Communication with Computers, Vol. II) Carl Hanser Ver-

lag, Muenchen, Wien; Macmillan, London, 1979.

[Shi86] Shieber, S.M. (1986). *An Introduction to Unification-Based Approaches to Grammar. CSLI Lecture Notes* 4, Center for the Study of Language and Information, Stanford University, Stanford, CA.

[Sij93] Sijtsma, W. and O. Zweekhorst (1993). Comparison and Review of Commercial Natural Language Interfaces. *Twente Workshop on Lang. Techn. 5 (TWLT5)*, 43–57.

[Sik93] Sikkel, K. (1993). *Parsing Schemata*. Ph.D. Thesis, Dept. of Computer Science, University of Twente, Enschede, the Netherlands.

[SiA93] Sikkel, K. and R. op den Akker (1993). Predictive Head-Corner Chart Parsing. *Int. Workshop on Parsing Technologies (IWPT'93)*, 267–276.

[Spe93] Speelman, D. (1993). A Natural Language Interface that uses Generalized Quantifiers, *Twente Workshop on Lang. Techn. 5 (TWLT5)*, 69–74.

[Ter93] Stefanova, M. and W. ter Stal (1993). A Comparison of ALE and PATR: Practical Experiences. *Twente Workshop on Lang. Techn. 6 (TWLT6)*, 47–62.

[Tom91] Tomabechi, H. (1991). Quasi-Destructive Graph Unification. *Proc. 29th Annual Meeting of the Association of Computational Linguistics*, Berkeley, 315–322.

[Vet94] Van der Vet, P.E. et al. (1994). Plinius intermediate report. Memorandum Informatica 94-35, University of Twente, Enschede.

[Vos91] Vosse, T.G. (1991). Detection and Correction of Morho-Syntactic Errors in Shift-Reduce Parsing. *Twente Workshop on Lang. Techn. 1 (TWLT1)*, 69–77.

[Vos94] Vosse, T.G. (1994). *The Word Connection; Grammar based error correction in Dutch* . Uitgeverij Neslia Paniculata, Enschede.