Robust Grammatical Analysis for Spoken Dialogue Systems

Gertjan van Noord, Gosse Bouma, Rob Koeling, Mark-Jan Nederhof
Alfa-Informatica & BCN
University of Groningen

(*Received 25 December 1997*)

Abstract

We argue that grammatical analysis is a viable alternative to concept spotting for processing spoken input in a practical spoken dialogue system. We discuss the structure of the grammar, and a model for robust parsing which combines linguistic sources of information and statistical sources of information. We discuss test results suggesting that grammatical processing allows fast and accurate processing of spoken input.

1 Introduction

The NWO Priority Programme *Language and Speech Technology* is a research programme aiming at the development of spoken language information systems. Its immediate goal is to develop a demonstrator of a public transport information system, which operates over ordinary telephone lines. This demonstrator is called OVIS, Openbaar Vervoer Informatie Systeem (*Public Transport Information System*). The language of the system is Dutch.

At present, a prototype is in operation, which is a version of a German system developed by Philips Dialogue Systems in Aachen (Aust et al.1995), adapted to Dutch. This German system processes spoken input using "*concept spotting*", which means that the smallest information-carrying units in the input are extracted, such as locative phrases (mostly names of train stations) and temporal expressions, and these are translated more or less individually into updates of the internal database representing the dialogue state. The words between the concepts thus perceived are ignored.

The use of concept spotting is common in spoken-language information systems (Ward1989; Jackson et al.1991; Aust et al.1995; Allen et al.1996). Arguments in favour of this kind of shallow parsing are that it is relatively easy to develop the NLP component, since larger sentence constructs do not have to be taken into account, and that the robustness of the parser is enhanced, since sources of ungrammaticality occurring between concepts are skipped and therefore do not hinder the translation of the utterance to updates.

The prototype presently under construction (OVIS2) is based on a grammar which describes grammatical sentences, i.e. complete and well-formed user utterances, and thus differs radically from a concept spotting approach. This article presents a detailed account of a computational grammar for Dutch, and a robust parsing algorithm which incorporates this grammatical knowledge as well as other knowledge sources, such as acoustic evidence and Ngram statistics. We argue that robust parsing can be based on sophisticated grammatical analysis. In particular, the grammar describes full sentences, but in doing so, also describes the grammar of temporal expressions and locative phrases which are crucial for concept spotting. Robustness is achieved by taking these phrases into consideration, if a full parse of an utterance is not available. We show that our approach is feasible in terms of both accuracy and computational resources, and thus is a viable alternative to pure concept spotting.

Whereas some (e.g. Moore, Pereira, and Murveit (1989)) argue that grammatical analysis may improve recognition accuracy, our current experiments have as yet not been able to reveal a substantial advantage in this respect. However, the grammatical approach may become essential as soon as the application is extended in such a way that more complicated grammatical constructions need to be recognized. In that case, simple concept spotting may not be able to correctly process all constructions, whereas the capabilities of the grammatical approach extend much further.

The structure of this paper is as follows. In section 2 we describe the grammar for OVIS2. We present the grammar in some detail, since we believe it constitutes an interesting compromise between linguistic and computational considerations. Readers interested in processing issues rather than the details of linguistic analysis might prefer to skip section 2 (possibly except the first paragraph) and jump to section 3 immediately. That section describes the robust parsing algorithm. Section 4 reports test results, showing that grammatical analysis allows fast and accurate processing of spoken input.

2 A computational grammar for Dutch

In developing the OVIS2 grammar we have tried to combine the short-term goal of developing a grammar which meets the requirements imposed by the application (i.e. robust processing of the output of the speech recogniser, extensive coverage of locative phrases and temporal expressions, and the construction of fine-grained semantic representations) with the long-term goal of developing a general, computational, grammar which covers all the major constructions of Dutch.

The design and organisation of the grammar, as well as many aspects of the particular grammatical analyses we propose, are based on Head-driven Phrase Structure Grammar (Pollard and Sag1994). We depart from this formalism mostly for computational reasons. As is explained below, the grammar is compiled into a restricted kind of definite clause grammar for which efficient processing is feasible. The semantic component follows the approach to monotonic semantic interpretation using *quasi-logical forms* presented originally in Alshawi (1992).

The grammar currently covers the majority of verbal subcategorisation types (intransitives, transitives, verbs selecting a PP, and modal and auxiliary verbs), NP-syntax (including pre- and post-nominal modification, with the exception of relative clauses), PP-syntax, the distribution of VP-modifiers, various clausal types (declaratives, yes/no and WH-questions, and subordinate clauses), all temporal expressions and locative phrases relevant to the domain, and various typical spoken-language constructs. Due to restrictions imposed by the speech recogniser, the lexicon is relatively small (3200 word forms, many of which are names of stations and cities).

In sections 2.1- 2.4 we introduce the grammar formalism from both a computational and linguistic perspective. Section 2.5 describes the grammar of noun, prepositional, and verb phrases, subordinate and main clauses, WH-questions and topicalisation, and a number of domain specific constructions. Sections 2.6 and 2.7, finally, are concerned with semantics and the translation of quasi-logical forms into (application-specific) update-expressions.

2.1 Formalism

The formalism that we use for the OVIS2 Grammar is a variant of Definite Clause Grammar (DCG) (Pereira and Warren1980). We have chosen for DCG because:

- DCG provides for a balance between *computational efficiency* on the one hand and *linguistic expressiveness* on the other.
- DCG is a (simple) member of the class of declarative and constraint-based grammar formalisms. Such formalisms are widely used in linguistic descriptions for NLP.
- DCG is straightforwardly related to context-free grammar. Almost all parsing technology is developed for CFG; extending this technology to DCG is usually possible (although there are many non-trivial problems as well).
- The compilation of richer constraint-based grammar formalisms into DCG is well investigated and forms the basis of several wide-coverage and robust grammar systems (i.e. the Alvey-grammar (Briscoe et al.1987; Carroll1993; Briscoe and Carroll1993) and the Core Language Engine (Alshawi1992)).

The formalism for the grammar of OVIS2 imposes the following additional requirements:

- External Prolog calls (in ordinary DCG these are introduced in right-hand sides using curly brackets) are allowed, but must be resolved during grammar compilation time.
- Rules can be mapped to their 'context-free skeleton' (by taking the functor symbol of the terms appearing in the right-hand and left-hand sides of the rule). This implies that we do not allow the under-specification of categories in rules. This is motivated by our desire to experiment with parsing strategies in which part of the work is achieved on the basis of the context-free skeleton of the grammar. It also facilitates indexing techniques.
- An identifier is assigned to each rule. Such rule identifiers have a number of possible uses (debugging, grammar filtering, grammar documentation).

- The grammar specifies for each rule which daughter is the head. This allows head-driven parsing strategies.

An efficient head-corner parsing strategy for this formalism is discussed in van Noord (1997a). The restriction that external Prolog calls must be resolved at compilation time implies that we do not use delayed evaluation. More in particular, lexical rules (deriving a lexical entry from a given 'basic' lexical entry) must be applied at compile time and are not interpreted as (relational) constraints on under-specified lexical entries, as in van Noord and Bouma (1994). Although we have experimented with combinations of delayed evaluation and memoisation, as described in Johnson and Dörre (1995), the resulting systems were not efficient enough to be applied in the kind of practical system considered here.

Grammar rules. A grammar rule is defined by a ternary predicate, `rule/3`. The first argument of this predicate is a ground Prolog term indicating the rule identifier. The second argument of the rule is the mother category. Categories are non-variable Prolog terms. The third argument of the rule is a list of categories. Note that we require that the length of the list is given, and that none of the categories appearing in the list is a variable. An example of a grammar rule is provided:

(1)
```
rule(vp_vpnp, vp(Subj,Agr,Sem),
         [v(Subj,Agr,trans,l(Arg,Sem)),np(_,Arg)]).
```

Terminal symbols cannot be introduced in rules directly, but are introduced by means of lexical entries.

Lexical entries. The lexicon is defined by the predicate `lex/2`. As an example, the lexical entry 'sleeps' could be encoded as:

(2)
```
lex(sleeps,v(np,agr(3,sg),intrans,l(X,sleep(X)))).
```

The first argument is the terminal symbol introduced by this lexical category. The second argument is the category (a non-variable term). In cases where a lexical entry introduces a sequence of terminal symbols the first argument is also allowed to be a (non-empty) list of atoms.

Top category. The top category for the grammar (or *start symbol*) is defined by the unary predicate `top_category`. Its argument is an arbitrary non-variable Prolog term.

Feature constraints. Almost all work in computational grammar writing uses 'feature-structures' of some sort. It is fairly standard to compile (descriptions of) such features-structures into first-order terms (see Pulman (1996) for a recent overview). We use the HDRUG development platform (van Noord and Bouma1997b), which contains a library for compiling feature constraints into Prolog terms, and various predicates to visualise such Prolog terms as feature structures in matrix notation.

The most important operators provided by the HDRUG library are the type assignment operator ('`=>`'), the path equality operator ('`<=>`'), and the path operator ('`:`'). A typical grammar fragment employing those operators is:

(3)
```
rule(1,S,[Np,Vp]) :-
    S => s, np(Np), vp(Vp),
    Vp:vform => finite,
    subj_agreement(Vp,Np).

np(Np) :- Np => np, Np:lex => -.
vp(Vp) :- Vp => v,  Vp:lex => -.
subj_agreement(Vp,Np) :- Vp:agr <=> Np:agr.
```

In this rule, the constraint `Np:lex => -` indicates that the value of the `lex` attribute of `Np` is of type `-`. The constraint `Vp:agr <=> Np:agr` indicates that the value of the `agr` attribute of `Vp` is identical to the value of the `agr` attribute of `Np`. Internally, such a rule could be represented as follows (the actual result of the compilation depends on what attributes are allowed for what types; declarations of this sort are part of the grammar):

(4)
```
rule(1,s,[np(Agr,-),v(Agr,-,finite,_,_)]).
```

We often will write such rules in matrix notation, as follows:

(5) $$\texttt{rule(}\ 1,\ s,\ \langle \begin{bmatrix} \mathit{np} & \\ \mathsf{agr} & \boxed{1} \\ \mathsf{lex} & \text{-} \end{bmatrix}, \begin{bmatrix} \mathit{v} & \\ \mathsf{agr} & \boxed{1} \\ \mathsf{lex} & \text{-} \\ \mathsf{vform} & \mathsf{finite} \end{bmatrix} \rangle\ ).$$

The feature library also supports boolean combinations of atomic values; these are compiled into Prolog terms using a technique described in Mellish (1988) (who attributes it to Colmerauer) and Pulman (1996). Thus, we may specify `agr` values such as `sg` $\wedge$ `(sec` $\vee$ `thi)`, denoting an agreement value which is singular and either second or third person.

We have also found it useful to provide the predicates `unify_ifdef/3`, `ifdef/4`, and `unify_except/3`. The predicate `unify_ifdef(C1,C2,Att)` can be used to require that if both `C1` and `C2` can have the attribute `Att` (i.e. `C1, C2` are of a type for which `Att` is a possible feature), then the values `C1:Att` and `C2:Att` must be identical. The predicate `ifdef(Att,Cat,Val,Otherwise)` is used to require that `Cat:Att` is identical to `Val` if `Att` is an appropriate feature for `Cat`. Otherwise `Val` is identical to `Otherwise`. The predicate `unify_except(C1,C2,Path)` unifies `C1` and `C2`, with the exception of the value of `Path`, which must be defined for both `C1` and `C2`, but which may have incompatible values. These predicates simplify the definition of the grammar code below.

2.2 Signs

In unification-based grammar formalisms, linguistic information is represented by means of typed feature-structures. Each word or phrase in the grammar is associated with such a feature-structure, in which syntactic and semantic information is bundled. Within Head-driven Phrase Structure Grammar (HPSG), such feature-structures are called *signs*, a terminology which we will follow here.

At present, the grammar makes use of 15 different types of sign, where each type roughly corresponds to a different category in traditional linguistic terminology. For each type of sign, a number of features are defined. For example, for the type NP, the features AGR, NFORM, CASE, and SEM are defined. These features are used to encode the agreement properties of an NP, (morphological) form, case and semantics, respectively. A more detailed presentation of these features follows below.

There are a number of features which occur in most types of sign, and which play a special role in the grammar. The feature SC (SUBCATEGORISATION) (present on signs of type *v, sbar, det, a, n* and *p*), for instance, is a feature whose value is a list of signs. It represents the subcategorisation properties of a given sign. As will be explained below, it is used to implement rules which perform functor-argument application (as in Categorial Grammar).

The feature SLASH is present on *v, ques* and *sbar*. Its value is a list of signs. It is used to implement a (restricted) version of the account of nonlocal dependencies proposed in Pollard and Sag (1994) and Sag (1997). The value of SLASH is the list of signs which are 'missing' from a given constituent. Such a 'missing' element is typically connected to a preposed element in a topicalisation sentence or WH-question. The same mechanism can also be used for relative clauses.

The feature VSLASH is similar to SLASH in that it records the presence of a missing element, a verb in this case. It is used to implement an account of Dutch main clauses, based on the idea that main clauses are structurally similar to subordinate clauses, except for the fact that the finite verb occurs as first or second constituent within the clause and the clause final position where finite verbs occur in subordinate clauses is occupied by an *empty* verbal sign (i.e. an element which is not visible in the phonological or orthographic representation of the sentence).

The feature SEM is present on all signs. It is used to encode the semantics of a word or phrase, encoded as a *quasi logical form* (Alshawi1992). The feature MOD is present on the types *a, pp, p, adv, sbar* and *modifier.* It is used to account for the semantics of modifiers. Its value is a list of *quasi-logical forms.* In the sections below on syntax, we only give an informal impression of the semantics. The details of the semantic construction rules and principles are dealt with in section 2.6.

2.3 Syntax: principles, structures, and rules

An important restriction imposed by the grammar-parser interface is that each rule must specify the category of its mother and daughters. A consequence of this requirement is that general rule-schemata, as used in Categorial Grammar and HPSG cannot be used in the OVIS2 grammar. A rule which specifies that a head daughter may combine with a complement daughter, if this complement unifies with the first element on SC of the head (i.e. a version of the categorial rule for functor-argument application) cannot be implemented directly, as it leaves the categories of the daughters and mother unspecified. Nevertheless, generalisations of this type do play a role in the grammar. We adopt an architecture for grammar rules similar

```
hd_comp_struct(Head,Complements,Mother) :-
    hd_struct(Head,Complements,Head,Mother).

hd_mod_struct(Head,Modifier,Mother) :-
    hd_struct(Head,[],Modifier,Mother),
    Head:sem <=> HeadSem,
    Modifier:mod <=> [HeadSem].

hd_struct(Head,Complements,SemanticHead,Mother) :-
    head_feature_principle(Head,Mother),
    valence_principle(Head,Complements,Mother),
    filler_principle(Head,[],Mother),
    SemanticHead:sem <=> Mother:sem.

head_feature_principle(Head,Mother) :-
    unify_ifdef(Head,Mother,vform),
    unify_ifdef(Head,Mother,agr),
    unify_ifdef(Head,Mother,case),
    unify_ifdef(Head,Mother,mod),
    unify_ifdef(Head,Mother,pform),
    unify_ifdef(Head,Mother,aform),
    unify_ifdef(Head,Mother,vslash),
    unify_ifdef(Head,Mother,subj).

valence_principle(Head,Complements,Mother) :-
    ifdef(sc,Head,HeadSc,[]),
    ifdef(sc,Mother,MotherSc,[]),
    append(Complements,MotherSc,HeadSc)
```

Fig. 1. Structures and Principles

to that of HPSG, in which individual rules are classified in various *structures*, which are in turn defined in terms of general *principles*.

Rules normally introduce a structure in which one of the daughters can be identified as the *head*. The head daughter either subcategorises for the other (complement) daughters or else is modified by the other (modifier) daughters.

The two most common structures are the *head-complement* and *head-modifier* structure.[1] In figure 1 we list the definitions for these structures and the principles they refer to, except for the *filler* principle, which is presented in the section on topicalisation.

Head-complement and *head-modifier* structures are instances of *headed* structures. The definition of *headed* structure refers to the HEAD-FEATURE, VALENCE, and FILLER principles, and furthermore fixes the semantic head of a phrase. Note that the definition of `hd-struct` has a number of parameters. The idea is that a headed structure will generally consist of a head daughter, and furthermore of zero or more complement daughters and possibly a modifier. *Head-complement* and *head-modifier*

[1] Other structures are the *main-clause* and *head-filler* structure. These are discussed in the sections on main-clause syntax and topicalisation.

structures differ from each other only in that the first introduces complements, but no modifiers, whereas the second introduces no complements, but a modifier. Moreover, the syntactic head is also the semantic head in *head-complement* structures, but not in a *head-modifier* structure. In *head-modifier* structures, the semantic contribution of the head to the meaning of the phrase as a whole is handled by unifying the head semantics with the value of (the first element of) MOD on the modifier.

The HEAD FEATURE PRINCIPLE states for a number of features (the *head-features*) that their value on the head daughter and mother must be unified. As this principle generalises over various types of sign, its definition requires the predicate `unify_ifdef`.

The VALENCE PRINCIPLE determines the value of the *valence feature* SC. The value of SC on the head daughter of a rule is the concatenation (`append`) of the list of complement daughters and the value of SC on the mother. Another way to put this is that the value of SC on the mother is the value of SC on the head daughter minus the elements on SC that correspond to the complement daughters. Note that the formulation of the VALENCE PRINCIPLE is complicated by the fact that SC (or SUBJ) may sometimes not be defined on the mother. In that case, it is assumed that the value of SC on the head daughter must correspond exactly to the list of complement daughters. The constraint `ifdef(sc,Mother,MotherSc,[])` states that the value of SC on `Mother` unifies with `MotherSc`, if SC is defined for the type of `Mother`. Otherwise, `MotherSc` is assigned the value `[]` (i.e. the empty list).

The structures defined in figure 1 are used in the definition of grammar rules. The `np-det-n` rule introduces a *head-complement* structure in which (following the traditional semantic analysis) the determiner is the head, and the noun the complement:

(6)
```
rule(np_det_n, NP, [Det, N]) :-
    NP => np, Det => det, N => n,
    NP:nform => norm, hd_comp_struct(Det,[N],NP).
```

The `n-adj-n` rules introduces a *head-modifier* structure where the adjective is the modifier:

(7)
```
rule(n_adj_n, N1, [AdjP, N0]) :-
    N1 => n, AdjP => a, N0 => n,
    AdjP:agr <=> N0:agr, hd_mod_struct(N0,AdjP,N1).
```

Note that for a given rule, the types of the mother and daughters must be specified, and furthermore, the number of complements is always specified. This implies that the constraints in the principles in figure 1 can be reduced to a number of basic constraints on the values of particular features defined for the signs in the rule. The previous two rules can be depicted in matrix notation as (where ⟨⟩ denotes the empty list):

(8) $$\texttt{np\_det\_n:}\ \begin{bmatrix} \textit{np} & \\ \mathsf{agr} & \boxed{1} \\ \mathsf{nform} & \mathsf{norm} \\ \mathsf{sem} & \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{det} & \\ \mathsf{sc} & \langle \boxed{3} \rangle \\ \mathsf{agr} & \boxed{1} \\ \mathsf{sem} & \boxed{2} \end{bmatrix} \quad \boxed{3}\ \mathsf{n}$$

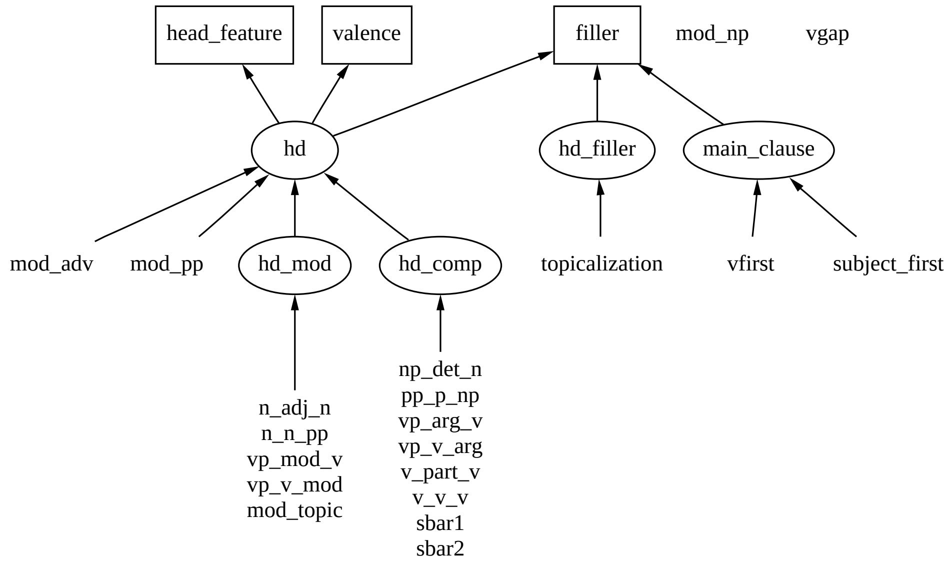

Fig. 2. The Rule Hierarchy (with PRINCIPLES shown in boxes, *structures* in ovals, and `rules` without frame). Note that the `mod_np` rule (a unary rule which transforms temporal NPs into verbal modifiers) and the `vgap` rule (a rule which introduces verbal gaps) are exceptional in that they do not inherit from general principles.

$$\text{(9)}\quad \texttt{n\_adj\_n:} \begin{bmatrix} n & \\ \mathsf{sc} & \boxed{1} \\ \mathsf{agr} & \boxed{2} \\ \mathsf{sem} & \boxed{3} \end{bmatrix} \rightarrow \begin{bmatrix} a & \\ \mathsf{sc} & \langle\rangle \\ \mathsf{agr} & \boxed{2} \\ \mathsf{sem} & \boxed{3} \\ \mathsf{mod} & \langle\boxed{4}\rangle \end{bmatrix} \begin{bmatrix} n & \\ \mathsf{sc} & \boxed{1} \\ \mathsf{agr} & \boxed{2} \\ \mathsf{sem} & \boxed{4} \end{bmatrix}$$

An overview of all grammar rules defined in the fragment at the moment, together with the structures and principles from which they inherit, is given in figure 2.

The classification of rules into structures, which are in turn defined in terms of principles, allows us to state complicated rules succinctly and to express a number of generalizations. Nevertheless, it is also clear that the rules could have been more general, if rule *schemata* (in which the type of the daughters, or even the number of daughters is not necessarily specified) had been allowed. Given this restriction, one may even wonder whether the VALENCE PRINCIPLE (and the feature SC that comes with it) cannot be eliminated in favour of more specific rules. Valence features are particularly important for grammars employing rule schemata, but they are much less crucial for more traditional types of grammar. Although eliminating valence features is not impossible in principle, we believe that the present set-up still has advantages, although these are less apparent than in grammars which make use of rule schemata. Expressing valence information lexically, instead of using more detailed syntactic rules, has the advantage that idiosyncratic subcategorization requirements (such as the restriction that *denken* (*to think*) requires a PP-complement headed by *aan* (*about*), or the fact that *komen* (*to come*) may combine with the particle *aan* (the combination of which means *to arrive*)) need not be stated in the rules. Similarly, all constraints having to do with case marking and agreement

```
intransitive(Pred,Sign) :- iv(Sign), iv_sem(Sign,Pred).

transitive(Pred,Sign) :-   tv(Sign), tv_sem(Sign,Pred).

v(V) :- V => v, V:lex => basic,
    V:vslash => [], V:subj <=> [Subj],
    Subj => np, Subj:nform => norm.

iv(V) :- v(V), V:sc <=> [].

tv(V) :- v(V), V:sc <=> [Obj],
    Obj => np, Obj:nform => norm, Obj:case => acc.

weather_v(V) :- iv(IV), unify_except(IV,V,subj:h:nform),
    V:subj:h:nform => it.
```

Fig. 3. Fragment of the lexical hierarchy

can be expressed lexically, as well as the semantic relation between a head and its dependents.

2.4 The lexicon

The lexicon is a list of clauses `lex(Word,Sign)`, associating a word (or sequence of words) with a specific sign.

Constraint-based grammars in general, and lexicalist constraint-based grammars in particular, tend to store lots of grammatical information in the lexicon. This is also true for the OVIS2 grammar. A lexical entry for a transitive verb, for instance, not only contains information about the morphological form of this verb, but also contains the features SC and SUBJ for which quite detailed constraints may be defined. Furthermore, for all lexical signs it is the case that their semantics is represented by means of a feature-structure. This structure can also be quite complex. To avoid massive reduplication of identical information in the lexicon, the use of inheritance is therefore essential.

In figure 3, we illustrate the use of inheritance in the lexicon. All lexical entries for verbs have a number of properties in common, such as the fact that they are of type *v*, and take a normal (non-locative and non-temporal) NP as subject. This is expressed by the template `v(V)`. Intransitive verbs (`iv(V)`) can now be characterised syntactically as verbs which do not subcategorise for any (non-subject) complements. Transitive verbs (`tv(V)`) subcategorise for an NP with accusative case. The templates `intransitive(Pred,Sign)` and `transitive(Pred,Sign)`, finally, combine the syntactic and semantic properties of intransitive and transitive verbs. The variable `Pred` is used in the semantics to fix the value of the predicate defined by a particular verb. A limited form of non-monotonic inheritance is supported (see Carpenter (1992) and Bouma (1992) for more general approaches). For instance, 'weather' verbs require the dummy pronoun *het* (*it*) as subject, but behave otherwise as intransitive verbs. This can be expressed by letting `weather_v` inherit from `iv`, with the exception of the value of the NFORM attribute of (head of the list containing) the subject, which is assigned an exceptional value. The attribute-value matrices for the templates `iv(V)` and `tv(V)` are:

(10) $\texttt{iv(}\begin{bmatrix} v & \\ \textsf{lex} & \textsf{basic} \\ \textsf{sc} & \langle\rangle \\ \textsf{subj} & \left\langle\begin{bmatrix} np & \\ \textsf{nform} & \textsf{norm}\end{bmatrix}\right\rangle \\ \textsf{vslash} & \langle\rangle \end{bmatrix}\texttt{).}\quad \texttt{tv(}\begin{bmatrix} v & \\ \textsf{lex} & \textsf{basic} \\ \textsf{sc} & \left\langle\begin{bmatrix} np & \\ \textsf{nform} & \textsf{norm} \\ \textsf{case} & \textsf{acc}\end{bmatrix}\right\rangle \\ \textsf{subj} & \left\langle\begin{bmatrix} np & \\ \textsf{nform} & \textsf{norm}\end{bmatrix}\right\rangle \\ \textsf{vslash} & \langle\rangle \end{bmatrix}\texttt{).}$

The lexicon itself (i.e. the predicate `lex/2`) is defined in terms of the predicates `entry`, `inflection` and `lexical_rules`:

(11)
```
lex(Word,Sign) :-
      entry(Root,Sign0),
      inflection(Root,Word,Sign0,Sign1),
      lexical_rules(Sign1,Sign).
```

The definition of `entry(Root, Sign)` defines for each root form what its associated sign is. For instance, for verbs we must typically distinguish a first person singular form, a second and third person singular form, and a plural form (which is also the form of the infinitive). The predicate `inflection` defines how inflected forms are derived. For example, there is an inflection rule which adds a `t` to the base form of a verb, and specifies that its agreement features are third person singular, and its VFORM value is *fin*. Lexical rules can be used to transform the sign associated with a lexical entry. For instance, the account of nonlocal dependencies sketched below makes use of a lexical rule which removes a sign from SC and places it on SLASH. A more detailed account of this lexical rule is given in the section on nonlocal dependencies. As an example, assume the stem `arriveer` (to arrive) is defined as an intransitive:

(12)
```
entry(arriveer,Sign):-
      intransitive(arriveren,Sign).
```

Such a definition will give rise to a number of lexical entries. One of these will be the third person singular finite form:

(13) $\textsf{lex(arriveert,}\begin{bmatrix} v & \\ \textsf{lex} & \textsf{basic} \\ \textsf{vform} & \textsf{fin} \\ \textsf{sc} & \langle\rangle \\ \textsf{subj} & \left\langle\begin{bmatrix} np & \\ \textsf{nform} & \textsf{norm} \\ \textsf{agr} & \textsf{sg} \wedge \textsf{thi}\end{bmatrix}\right\rangle \\ \textsf{vslash} & \langle\rangle \end{bmatrix}\textsf{).}$

2.5 Syntactic Coverage

Below, we describe the syntactic coverage of the grammar. The grammar is not intended as a general, wide-coverage, grammar for Dutch. This implies not only that coverage in the lexical domain is limited, but also that several grammatical constructions are not taken into consideration (e.g. passives) or accounted for only to a certain extent (e.g. the grammar of Dutch verb clusters). The coverage of the grammar is quite satisfactory for the OVIS application, however. For instance, when evaluating the grammar on a corpus of 1000 transcribed test-sentences, we obtained a semantic concept accuracy of 95% (see section 4.2 for discussion).

2.5.1 Noun phrases

The four types which are relevant in the syntax of noun phrases are *np* (noun phrase), *det* (determiner), *a* (adjective) and *n* (noun). Each type has the attributes AGR and SEM. Furthermore, *det* and *n* have an attribute SC. The *np* type has three further attributes: CASE, NFORM and PFORM. Finally, *a* is also specified for MOD.

The features AGR (*agreement*), CASE, and NFORM (*noun form*) are used to encode agreement properties (encoded as a boolean combination of *person*, *number*, *determiner* and *definiteness*), the case value and the form of an item. Their possible values are listed in (14). Note that AGR contains the information needed for subject-verb agreement, as well as for NP-internal agreement (between determiner, adjective, and noun). The agreement types *de* and *het* (the two forms of the definite article) distinguish between neuter and nonneuter nouns. CASE and NFORM are relevant for full NPs only.

(14)	*Agr*	(fir ∨ sec ∨ thi) ∧ (sg ∨ plu) ∧ (de ∨ het) ∧ (def ∨ indef)
	Case	nom ∨ acc
	Nform	norm ∨ loc ∨ temp ∨ num

Full NPs never take complements, so they do not have a feature SC. Adjectives may modify a noun, therefore the feature MOD is defined for type *a*.

The two rules we presented in (6) and (7) (section 2.3) are used to form NPs consisting of a determiner and a (possibly complex) noun, and Ns consisting of an adjective followed by a (possibly complex) noun. The derivation of the NP *de volgende intercity* (the next intercity) is shown in figure 4.

2.5.2 Prepositional phrases

Prepositional phrases are of type *pp* and are headed by prepositions, i.e. elements of type *p*. Prepositions subcategorize (usually) for an NP, so the value of SC on P will be a list of length one, containing the NP-complement. The feature PFORM takes as value the specific form of the preposition heading the PP (i.e. *van, op, naar,* ...). This information can be used to let a verb select a PP headed by a specific preposition.

Full PPs can modify nouns or verb phrases. Therefore, PP has a feature MOD. MOD has to be present on P as well, as the relation between the semantics of the preposition and the element it modifies is encoded as part of the lexical entry of a

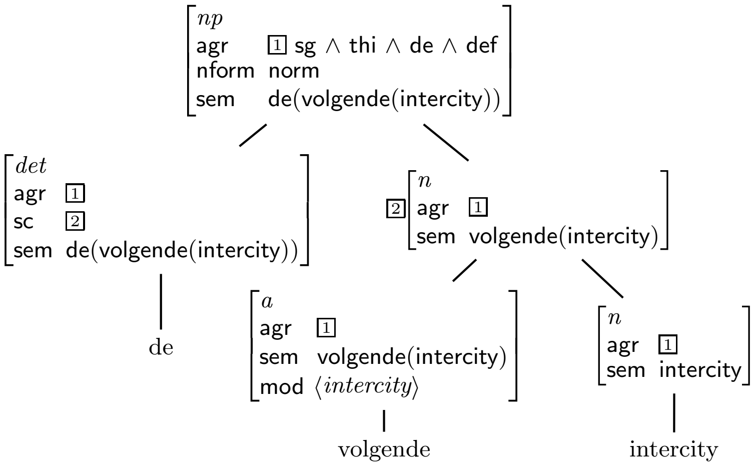

Fig. 4. de volgende intercity (the next intercity)

preposition. Here, we give the rule which forms PPs and the rule which lets a PP combine as a modifier with a noun.

(15) $\mathtt{pp\_p\_np}: \begin{bmatrix} \mathit{pp} & \\ \mathsf{pform} & \boxed{1} \\ \mathsf{sem} & \boxed{2} \\ \mathsf{mod} & \boxed{3} \end{bmatrix} \rightarrow \begin{bmatrix} \mathit{p} & \\ \mathsf{pform} & \boxed{1} \\ \mathsf{sc} & \langle\boxed{4}\rangle \\ \mathsf{sem} & \boxed{2} \\ \mathsf{mod} & \boxed{3} \end{bmatrix} \quad \boxed{4}\ \mathsf{np}$

(16) $\mathtt{n\_n\_pp}: \begin{bmatrix} \mathit{n} & \\ \mathsf{sc} & \boxed{1} \\ \mathsf{agr} & \boxed{2} \\ \mathsf{sem} & \boxed{3} \end{bmatrix} \rightarrow \begin{bmatrix} \mathit{n} & \\ \mathsf{sc} & \boxed{1} \\ \mathsf{agr} & \boxed{2} \\ \mathsf{sem} & \boxed{4} \end{bmatrix} \quad \begin{bmatrix} \mathit{pp} & \\ \mathsf{sem} & \boxed{3} \\ \mathsf{mod} & \langle\boxed{4}\rangle \end{bmatrix}$

Using these rules, we can derive the phrase *intercity uit Goes* (intercity from Goes) as illustrated in figure 5.

It should be noted that since adjectives precede the nouns they modify and PPs follow them, an expression such as *volgende intercity uit Groningen* (next intercity from Groningen) will receive two parses. This appears to be a case of spurious ambiguity. There are intensional adjectives, such as *zogenaamde* (*alleged*), which need to be able to take scope over a complex noun, but it seems that modifying PPs never need to take scope over a adjective + noun combination. It is not easy to rule out the latter type of derivation, however, without introducing additional features.

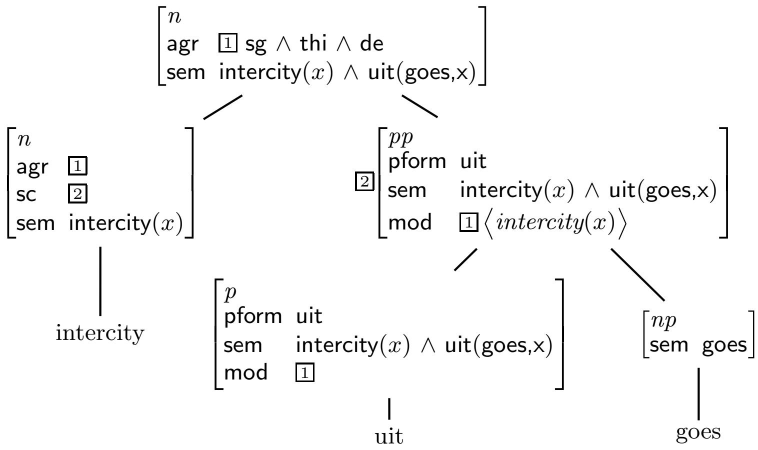

Fig. 5. intercity uit Goes (intercity from Goes)

2.5.3 Verb phrases

Both verbs and verb phrases are of type *v*:

$$(17)\quad \begin{bmatrix} \mathit{v} & \\ \textsf{lex} & \textsf{ylex} \lor \textsf{nlex} \\ \textsf{null} & \textsf{null} \lor \textsf{nonnull} \\ \textsf{vform} & \textsf{fin} \lor \textsf{inf} \lor \textsf{te} \lor \textsf{psp} \\ \textsf{sc} & \textsf{listof(Sign)} \\ \textsf{subj} & \textsf{listof(Sign)} \\ \textsf{sem} & \textsf{Qlf} \\ \textsf{slash} & \textsf{listof(Sign)} \\ \textsf{vslash} & \textsf{Vslash} \end{bmatrix}$$

The features LEX, NULL, and VFORM are specific for *v*. The feature VFORM is used to distinguish finite, infinitive, *te*-infinitive and past participle verbs (and verb phrases headed by such verbs). The feature LEX is used to distinguish lexical verbs (*ylex*) from verbal phrases that are not lexical (*nlex*). The feature *ylex* subsumes two further subtypes basic ∨ complex, to distinguish basic and complex lexical verbs. The latter are combinations of a verb and a separable prefix (*aan+komen*, arrive) or combinations of a modal verb and a main verb (*wil vertrekken*, want to leave). The feature NULL is used to distinguish verbal *traces* (i.e. verbal signs without phonological content) from other verbal signs. The features SUBJ, SLASH, and VSLASH and NULL are discussed in the section below on sentential syntax.

There are a number of similar rules for combining a verb or a verbal projection with one of its complements. One rule combines a noun phrase complement with a verbal head (*een kaartje kopen*, buy a ticket):

$$(18)\quad \texttt{vp\_np\_v}\text{:} \begin{bmatrix} \mathit{v} & \\ \textsf{lex} & \textsf{nlex} \\ \textsf{sc} & \boxed{1} \\ \textsf{vform} & \boxed{2} \\ \textsf{slash} & \boxed{3} \\ \textsf{vslash} & \boxed{4} \end{bmatrix} \rightarrow \boxed{6}\ \textsf{np} \quad \begin{bmatrix} \mathit{v} & \\ \textsf{sc} & \left\langle \boxed{6} | \boxed{1} \right\rangle \\ \textsf{vform} & \boxed{2} \\ \textsf{slash} & \boxed{3} \\ \textsf{vslash} & \boxed{4} \end{bmatrix}$$

Since PPs may either precede or follow the head (*vanuit Leiden vertrekken, vertrekken vanuit Leiden*, depart from Leiden), there are two rules to combine such a PP and a verbal head. Finally, there is a rule which combines a verbal head with a *te*-infinitive (*weigeren naar Groningen te komen*, refuse to come to Groningen). The result of combining a verb (or verbal projection) with its complement is a phrase (i.e. the value of LEX on the mother is *nlex*).

A verbal modifier can be either an adverb, a PP, or a temporal NP. There are unary rules rewriting signs of type modifier into each of these categories. One such rule is the following:

(19) `mod_adv:` $\begin{bmatrix} \textit{modifier} \\ \textsf{mod}\ \boxed{1}\langle\_\rangle \end{bmatrix} \rightarrow \begin{bmatrix} \textit{adv} \\ \textsf{mod}\ \boxed{1} \end{bmatrix}$

At the moment, we allow all modifiers to precede or follow the verb (*ik moet morgen in Assen zijn/ in Assen zijn morgen/ morgen zijn in Assen*, I must be in Assen tomorrow, *ik moet tien uur in Assen zijn/?in Assen zijn tien uur*, I must be in Assen at ten o'clock). Therefore, there are two similar rules, `vp_v_mod` and `vp_mod_v`, in which a verb combines with a modifier. The first is illustrated here:

(20) `vp_mod_v:` $\begin{bmatrix} \textit{v} & \\ \textsf{lex} & \textsf{nlex} \\ \textsf{sc} & \boxed{1} \\ \textsf{vform} & \boxed{2} \\ \textsf{slash} & \boxed{3} \\ \textsf{vslash} & \boxed{4} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{modifier} \\ \textsf{mod}\ \langle\boxed{5}\rangle \end{bmatrix} \begin{bmatrix} \textit{v} & \\ \textsf{sc} & \boxed{1} \\ \textsf{vform} & \boxed{2} \\ \textsf{slash} & \boxed{3} \\ \textsf{vslash} & \boxed{4} \\ \textsf{sem} & \boxed{5} \end{bmatrix}$

A special type *modifier* (with SEM and MOD as only attributes) in combination with three unary rules is used to introduce the various types of verbal modifier. A sample derivation is given in figure 6 (the value of the features SLASH and VSLASH is not shown, but is $\langle\ \rangle$ on all verbal signs in this derivation).

Finally, there are two VP-rules that give rise to 'complex' lexical expressions, instead of phrases. Firstly, consider the `v_v_v` rule:

(21) `v_v_v:` $\begin{bmatrix} \textit{v} & \\ \textsf{lex} & \textsf{complex} \\ \textsf{sc} & \boxed{1} \\ \textsf{vform} & \boxed{2} \\ \textsf{slash} & \boxed{3} \\ \textsf{vslash} & \boxed{4} \\ \textsf{sem} & \boxed{5} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{v} & \\ \textsf{lex} & \textsf{basic} \\ \textsf{sc} & \langle\boxed{6} \mid \boxed{1}\rangle \\ \textsf{vform} & \boxed{2} \\ \textsf{slash} & \boxed{3} \\ \textsf{vslash} & \boxed{4} \\ \textsf{sem} & \boxed{5} \end{bmatrix} \quad \boxed{6}\begin{bmatrix} \textit{v} & \\ \textsf{lex} & \textsf{ylex} \\ \textsf{vslash} & \langle\rangle \end{bmatrix}$

The `v_v_v` rule is used to derive phrases in which a modal verb precedes its infinitival complement (*(dat ik om tien uur) wil vertrekken*, that I want to leave at ten o'clock). We adopt an analysis of such constructions in which modals *inherit* the arguments on SC of the infinitival verb with which they combine. This is illustrated for the root *wil* (want) in (1).

(1) `wil` $\mapsto \begin{bmatrix} \textit{v} & \\ \textsf{lex} & \textsf{basic} \\ \textsf{sc} & \left\langle \begin{bmatrix} \textit{v} & \\ \textsf{lex} & \textsf{ylex} \\ \textsf{sc} & \boxed{1} \\ \textsf{vform} & \textsf{inf} \end{bmatrix} \mid \boxed{1} \right\rangle \end{bmatrix}$

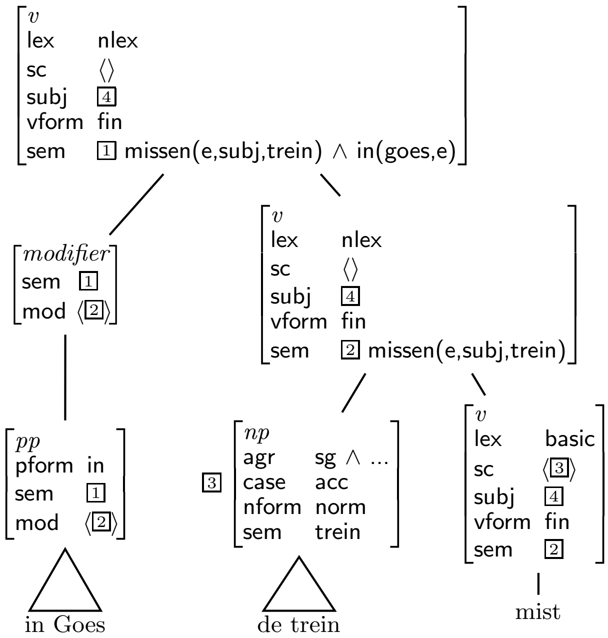

Fig. 6. (dat Rob) in Goes de trein mist (that Rob misses the train in Goes)

This allows us to derive phrases such as *(dat ik) een kaartje wil kopen* (that I want to buy a ticket) where the finite modal verb combines with the infinitival verb before combining with the object of *kopen* (figure 7). Note that it is essential that the modal verb selects a [LEX *ylex*] argument in this case, as this excludes the derivation of ungrammatical expressions such as *(dat ik) wil een kaartje kopen*. The result of combining a modal with an infinitival verb is [LEX *complex*] (i.e. subsumed by [LEX *ylex*]). This implies that such combinations can be selected by another modal verb (i.e. *(dat ik) een kaartje zou willen kopen*, that I would like to buy a ticket).

Next, consider the `v_part_v` rule:

(22) `v_part_v`: $\begin{bmatrix} \mathit{v} & \\ \mathsf{lex} & \mathsf{complex} \\ \mathsf{sc} & \boxed{1} \\ \mathsf{vform} & \boxed{2} \\ \mathsf{slash} & \boxed{3} \\ \mathsf{vslash} & \boxed{4} \\ \mathsf{sem} & \boxed{5} \end{bmatrix} \rightarrow \boxed{6}\ \mathsf{part} \quad \begin{bmatrix} \mathit{v} & \\ \mathsf{lex} & \mathsf{ylex} \\ \mathsf{sc} & \left\langle \boxed{6} \mid \boxed{1} \right\rangle \\ \mathsf{vform} & \boxed{2} \\ \mathsf{slash} & \boxed{3} \\ \mathsf{vslash} & \boxed{4} \\ \mathsf{sem} & \boxed{5} \end{bmatrix}$

The rule `v_part_v` is used to account for constructions such as *(dat ik voor tien uur) aan wil komen* (that I want to arrive before ten o'clock). The prefix (or particle) *aan* of the verb *aankomen* (arrive) is separated from the root *komen* in this case. As the root *komen* specifies that it selects such a particle on its SC-list, the modal verb inherits this specification. The rule `v_part_v` allows us to combine a verb or verbal complex with a particle. There are two reasons for not using an analogue of the `vp_np_v`-rule in this case. First, modifiers may not appear in between a particle and

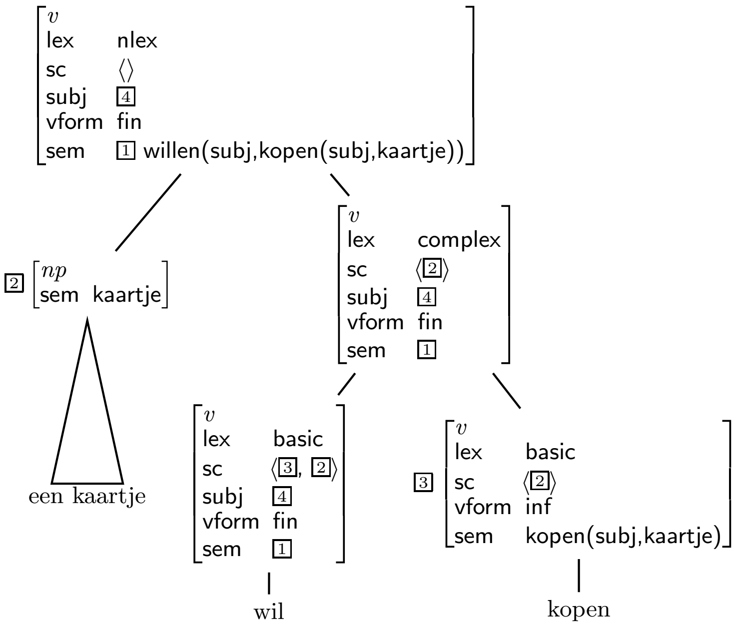

Fig. 7. (dat ik) een kaartje wil kopen (that I want to buy a ticket)

the verbal complex selecting this particle (* *(dat ik) aan om tien uur wil komen*). This is accounted for by requiring that the head in the rule for particles must be [LEX *ylex*] (and combinations of a modifier and a verbal head are always [LEX *nlex*]). Second, particles may appear 'inside' a verb cluster (*(dat ik voor tien uur) zou aan willen komen*, that I would like to arrive before ten o'clock). This implies that the result of combining a particle with a verb cluster must be [LEX *ylex*], instead of [LEX *nlex*] as specified on the `vp_np_v`-rule.

It should be obvious that these two rules, and the limited form of argument inheritance we allow (i.e. structure sharing of SC-lists only, and no concatenation of SC-lists), is not sufficient to account for the full range of verb clustering data in Dutch. For one thing, the grammar as it stands cannot handle 'inverted' word orders (*(dat ik de trein) halen moet*, that I must catch the train), where the infinitive precedes the modal verb. It is rather straightforward to include rules for inverted word orders. A potentially more problematic omission is the fact that perception verbs (*horen, zien*) and causative *laten*, which also introduce verb clusters (*(dat ik) Rob een kaartje laat kopen*, that I let Rob buy a ticket), cannot be accounted for. The analysis of this construction in van Noord and Bouma (1997a) is based on the notion 'argument inheritance'. This presupposes the possibility of recursive constraints in syntax (to concatenate SC-lists) as well as rules with an indefinite number of daughters. Both are excluded within the present formalism.

2.5.4 Subordinate clauses

Subordinate clauses containing a VP headed by a finite verb are of type *sbar* (the name *sbar* stems from X-bar grammar, where clauses introduced by a complementizer are (barred) projections of S). As finite subordinate clauses are always introduced by a complementizer, we assume that this complementizer is the head of the clause and that it subcategorises for a subject NP and a (finite) VP. The lexical entry for the complementizer *dat* (that), for instance, is:

$$(2)\quad \texttt{dat} \quad \mapsto \quad \begin{bmatrix} \textit{comp} & \\ \mathsf{sc} & \left\langle \boxed{1} \begin{bmatrix} \textit{np} & \\ \mathsf{case} & \mathsf{nom} \end{bmatrix}, \begin{bmatrix} \textit{v} & \\ \mathsf{vform} & \mathsf{fin} \\ \mathsf{sc} & \langle\rangle \\ \mathsf{subj} & \langle \boxed{1} \rangle \\ \mathsf{sem} & \boxed{2} \\ \mathsf{slash} & \boxed{3} \end{bmatrix} \right\rangle \\ \mathsf{sem} & \boxed{2} \\ \mathsf{mod} & \langle\rangle \\ \mathsf{slash} & \boxed{3} \end{bmatrix}$$

The complementizer unifies the NP on its SC with the subject of the VP. This implies that the NP is interpreted as subject of the VP. Furthermore, the complementizer has no independent semantics, but simply passes on the semantics of the VP. Since *dat* clauses cannot be modifiers, its MOD feature is empty. Other complementizers such as *omdat* (because) will have a non-empty value for this attribute to indicate that subordinate sentences headed by such complementizers can occur as modifier.

The rule constructing subordinate clauses is defined as follows:[2]

$$(23)\quad \begin{bmatrix} \textit{sbar} & \\ \mathsf{slash} & \boxed{1} \\ \mathsf{sem} & \boxed{2} \\ \mathsf{mod} & \boxed{3} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{comp} & \\ \mathsf{sc} & \langle \boxed{4}, \boxed{5} \rangle \\ \mathsf{slash} & \boxed{1} \\ \mathsf{sem} & \boxed{2} \\ \mathsf{mod} & \boxed{3} \end{bmatrix} \quad \boxed{4} \begin{bmatrix} \textit{np} & \\ \mathsf{case} & \mathsf{nom} \end{bmatrix} \quad \boxed{5} \begin{bmatrix} \textit{v} & \\ \mathsf{sc} & \langle\rangle \\ \mathsf{subj} & \langle \boxed{4} \rangle \\ \mathsf{vform} & \mathsf{fin} \\ \mathsf{slash} & \boxed{1} \\ \mathsf{vslash} & \langle\rangle \end{bmatrix}$$

A sample derivation is given in figure 8.

2.5.5 Main clauses

Main clauses with a finite verb in initial position (as in yes/no-questions) are of type *ques*. Main clauses in which the finite verb appears in second position (as in declarative sentences or WH-questions) are of type *root*. The attributes associated with these types are:

$$(24)\quad \begin{bmatrix} \textit{ques} & \\ \mathsf{subj} & \mathsf{listof(Sign)} \\ \mathsf{sem} & \mathsf{Qlf} \\ \mathsf{slash} & \mathsf{listof(Sign)} \end{bmatrix} \qquad \begin{bmatrix} \textit{root} & \\ \mathsf{sem} & \mathsf{Qlf} \end{bmatrix}$$

[2] There is an additional rule, for constructing subordinate clauses with a missing ('extracted') subject. This rule (`sbar2`) could be used in an account of nonlocal dependencies which allows for extraction out of subordinate clauses as well.

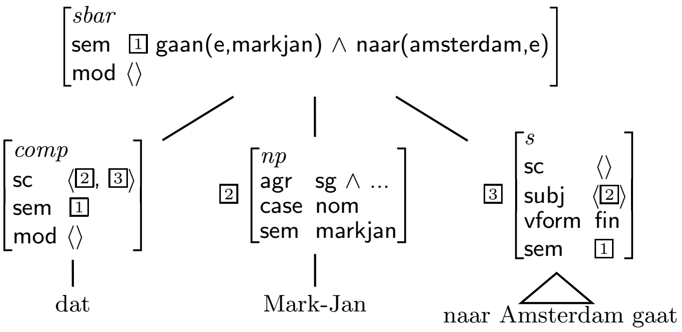

Fig. 8. dat Mark-Jan naar Amsterdam gaat (that Mark-Jan is going to Amsterdam)

Dutch main clauses differ from subordinate clauses in that the finite verb in main clauses appears in first or second position. There is a tradition, both in transformational and non-transformational grammar, to account for this fact by postulating a dependency between the finite verb and the position where finite verbs occur in subordinate clauses. The advantage of postulating such a dependency is that the grammar rules used for subordinate clauses are also applicable in main clauses. In transformational grammar, a dependency of this type can be established by means of a head-movement operation which moves the verb from its final position to a position at the beginning of the sentence.

Within the framework of HPSG (Netter1992; Frank1994) we can obtain a similar dependency by postulating a *verbal trace*, i.e. a verbal sign without phonological content, at the end of the clause. Using this verbal trace as the head, we can use the VP rules discussed above to build up a VP as usual.

The rule for introducing such a verbal trace is given in figure 9. Note that the sign for verbal traces differs from that of an ordinary verb in that its subcategorisation list in not instantiated, but made reentrant with VSLASH:VSC. Similarly, the semantics of the verbal gap is reentrant with VSLASH:VSEM. Furthermore, a verbal gap is a basic (i.e. non-complex) lexical verb, with no phonological content (i.e. [NULL *null*]). We can also safely assume that verbal traces are finite, as main clauses are always headed by a finite verb. The value of SUBJ is the empty list, as VPs headed by a verbal trace never combine with a subject directly (as will be shown below). Finally, SLASH also can be assumed to be empty.[3]

There are two rules which combine a finite verb with a VP containing a verbal trace, and which also introduce a subject (figure 10). Both rules are highly similar (they are therefore both instances of a MAIN-CLAUSE-STRUCT). The only difference is the category of the mother, and the order of the daughters. The *vfirst*-rule introduces phrases of the type *ques*, i.e. instances of verb-first clauses, in which

[3] The lexical rule which moves complements from SC to SLASH does not apply to verbal traces. Instead, it can be applied to the finite verb which 'binds' the trace. Also, if a verbal gap combines with a complement having a non-empty SLASH, the relevant passing on of the SLASH value is handled by the finite verb which binds the trace. This is possible because the SC-list of the verbal trace and the binder will be shared.

$$\texttt{vgap:}\ \begin{bmatrix} \multicolumn{2}{l}{v} \\ \text{lex} & \text{basic} \\ \text{null} & \text{null} \\ \text{vform} & \text{fin} \\ \text{sc} & \boxed{1} \\ \text{subj} & \langle\rangle \\ \text{slash} & \langle\rangle \\ \text{vslash} & \begin{bmatrix} \multicolumn{2}{l}{vslash} \\ \text{vsc} & \boxed{1} \\ \text{vsem} & \boxed{2} \end{bmatrix} \\ \text{sem} & \boxed{2} \end{bmatrix} \rightarrow \epsilon$$

Fig. 9. Verbal Gap

$$\begin{bmatrix} \multicolumn{2}{l}{ques} \\ \text{slash} & \boxed{1} \\ \text{sem} & \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} \multicolumn{2}{l}{v} \\ \text{sc} & \boxed{3} \\ \text{subj} & \langle\boxed{4}\rangle \\ \text{vform} & \text{fin} \\ \text{lex} & \text{basic} \\ \text{slash} & \boxed{1} \\ \text{vslash} & \langle\rangle \\ \text{sem} & \boxed{5} \end{bmatrix} \ \boxed{4}\begin{bmatrix} \multicolumn{2}{l}{np} \\ \text{case} & \text{nom} \end{bmatrix} \begin{bmatrix} \multicolumn{2}{l}{v} \\ \text{sc} & \langle\rangle \\ \text{subj} & \langle\rangle \\ \text{vform} & \text{fin} \\ \text{slash} & \langle\rangle \\ \text{vslash} & \begin{bmatrix} \multicolumn{2}{l}{vslash} \\ \text{vsc} & \boxed{3} \\ \text{vsem} & \boxed{5} \end{bmatrix} \\ \text{sem} & \boxed{2} \end{bmatrix}$$

$$\begin{bmatrix} \multicolumn{2}{l}{root} \\ \text{sem} & \boxed{2} \end{bmatrix} \rightarrow \ \boxed{4}\begin{bmatrix} \multicolumn{2}{l}{np} \\ \text{case} & \text{nom} \end{bmatrix} \begin{bmatrix} \multicolumn{2}{l}{v} \\ \text{sc} & \boxed{3} \\ \text{subj} & \langle\boxed{4}\rangle \\ \text{vform} & \text{fin} \\ \text{lex} & \text{basic} \\ \text{slash} & \langle\rangle \\ \text{vslash} & \langle\rangle \\ \text{sem} & \boxed{5} \end{bmatrix} \begin{bmatrix} \multicolumn{2}{l}{v} \\ \text{sc} & \langle\rangle \\ \text{subj} & \langle\rangle \\ \text{vform} & \text{fin} \\ \text{slash} & \langle\rangle \\ \text{vslash} & \begin{bmatrix} \multicolumn{2}{l}{vslash} \\ \text{vsc} & \boxed{3} \\ \text{vsem} & \boxed{5} \end{bmatrix} \\ \text{sem} & \boxed{2} \end{bmatrix}$$

Fig. 10. Rules for verb-first and subject first main clauses (y/n questions and simple declarative sentences)

the subject follows the main verb. The *subject-first*-rule introduces phrases of type *root*, in which the subject is first, and the main verb follows the subject. The constraints imply, among others, that the VP must contain a verbal trace, that the SC-information of the main verb is reentrant with VSLASH:VSC of the VP (and thus, indirectly, with the SC-value of the verbal trace), and that the semantics of main verb is shared with the value of VSLASH:VSEM on the VP (and thus, indirectly, with the semantics of the verbal trace). Note also that the VP acts as semantic head of the construction. This is necessary in order to ensure that the effect of verbal modifiers within the VP is properly taken into account. An example derivation of a subject first main clause is given in figure 11.

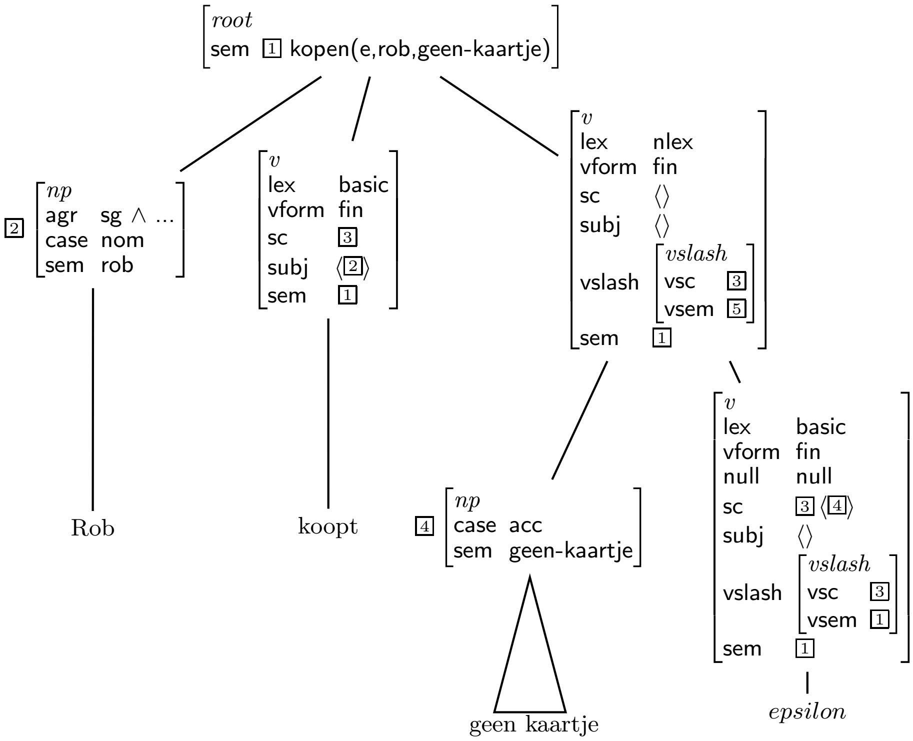

Fig. 11. Rob koopt geen kaartje (Rob does not buy a ticket)

2.5.6 Wh-questions and topicalisation

In the previous section, we have introduced a rule for verb-initial and subject-initial main clauses. The first phrase in a main clause can also be a (non-subject) complement or a modifier. This is typically the case for (non-subject) WH-questions. Sentences with a 'fronted' complement are treated as instances of a non-local dependency construction (where the dependency is mediated through SLASH). In sentences with a fronted modifier, it is assumed that the first element modifies the remainder of the clause, and thus a local treatment can be given.

Examples of sentences with a fronted complement are given in (3).

(3) a. Naar welk station wilt u reizen?
To which station do you want to travel
b. De laatste trein kunt u nog halen.
The last train, you can still catch

These examples are handled by means of a lexical COMPLEMENT-EXTRACTION rule applicable to verbs, and a syntactic HEAD-FILLER-rule for combining the fronted element with a *ques*-phrase containing a non-empty SLASH-value. The COMPLEMENT-EXTRACTION rule can apply in two ways: First, it can take a complement from SC and put it on SLASH (4a). This implies that this complement will not be found locally, but that it will be unified with an element in 'fronted' position. Second, it

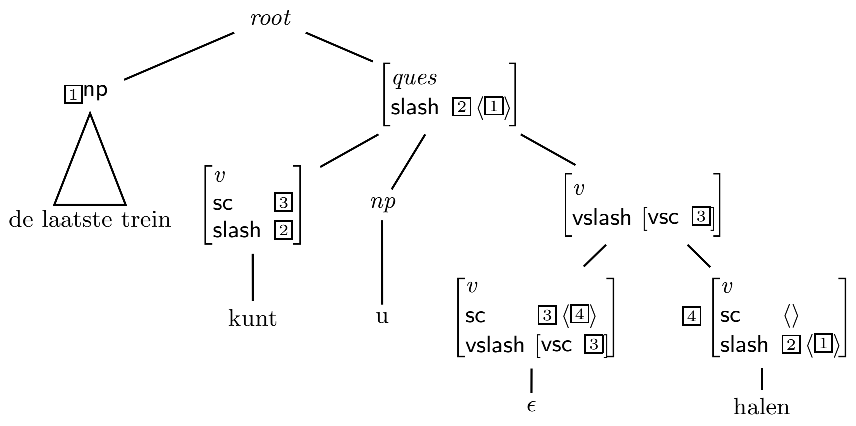

Fig. 12. De laatste trein kunt u halen

can make the SLASH value of a verb reentrant with the SLASH value of one of its complements (4b). This implies that true *non-local* dependencies are possible, as the head of a phrase can pass on information about missing elements from one of its dependents. If the complement-extraction rule does not apply, the SLASH value of the verb, as well as the SLASH value of all its complements, is set to ⟨⟩ (the empty list).

(4) a.
$$\begin{bmatrix} v & \\ \mathsf{sc} & \langle \boxed{1}\ pp \rangle \\ \mathsf{subj} & \boxed{2} \\ \mathsf{vform} & \boxed{3} \\ \vdots & \end{bmatrix} \rightarrow \begin{bmatrix} v & \\ \mathsf{sc} & \langle\rangle \\ \mathsf{slash} & \boxed{1} \\ \mathsf{subj} & \boxed{2} \\ \mathsf{vform} & \boxed{3} \\ \vdots & \end{bmatrix}$$

b.
$$\begin{bmatrix} v & \\ \mathsf{sc} & \langle v \rangle \\ \mathsf{subj} & \boxed{2} \\ \mathsf{vform} & \boxed{3} \\ \vdots & \end{bmatrix} \rightarrow \begin{bmatrix} v & \\ \mathsf{sc} & \left\langle \begin{bmatrix} v & \\ \mathsf{slash} & \boxed{1} \end{bmatrix} \right\rangle \\ \mathsf{slash} & \boxed{1} \\ \mathsf{subj} & \boxed{2} \\ \mathsf{vform} & \boxed{3} \\ \vdots & \end{bmatrix}$$

An example of a derivation involving SLASH is given in figure 12. The COMPLEMENT EXTRACTION rule has applied to *halen* (to catch) to produce a verbal sign with an empty SC-list and an *np* on SLASH. A verbal trace contains a reentrancy between its SC-list and its VSLASH:VSC-list. When the verbal trace combines with *halen*, the information that *halen* has an *np* on SLASH will therefore also be instantiated on VSLASH:VSC. This information is passed up to the resulting verb phrase. The *complement extraction* rule also applies to the finite verb *kunt* (can), but in this case it establishes a reentrancy between the SLASH value of the verb on the SC-list of *kunt* and the SLASH-value of *kunt* itself. The VFIRST rule unifies the SC-list of *kunt* with the VSLASH:VSC-list of the verb phrase *halen* ϵ, and thus, SLASH (of the verb on SC of *kunt*, and thus on *kunt* itself) is instantiated as $\langle np \rangle$. This information is passed on to the resulting *ques* phrase, which can then be combined with the initial *np* using the TOPICALISATION rule in 25.

$$(25)\quad \begin{bmatrix} \textit{root} \\ \mathsf{sem}\ \boxed{1} \end{bmatrix} \rightarrow \boxed{2} \quad \begin{bmatrix} \textit{ques} \\ \mathsf{slash}\ \boxed{2} \\ \mathsf{sem}\ \ \boxed{1} \end{bmatrix}$$

It should be noted that our account of non-local dependencies differs from earlier *slash*-based accounts, such as those in Gazdar et al. (1985) and Pollard and Sag (1994) in that it does not make use of a FOOT FEATURE principle. Instead, we adopt the approach of Sag (1997), who imposes the canonical constraint that the SLASH-value of a head is the set-union of the SLASH-values of its daughters. An EXTRACTION lexical rule can be used to remove an element from SC (COMPS) and to add this element to the set of elements on SLASH. In our implementation, we have made several simplifying assumptions. First, SLASH is not a set, but a list. Second, this list can contain at most one element. This assumption (which has the effect of restricting the number of 'missing' elements from a phrase to at most one) is too restrictive for a highly limited number of cases in English, but appears to be valid for Dutch. Third, instead of imposing a general constraint that SLASH must be the concatenation on the SLASH values of all elements on SC, we allow the COMPLEMENT EXTRACTION rule to unify the value of SLASH with one specific element on SC. We have to make this assumption, as the more general alternative requires the use of *delayed evaluation*, something which we wish to avoid in this grammar, or *difference lists*. While the latter alternative is possible within the present formalism, it also introduces a number of complications which are avoided in the present implementation. The fourth and final simplification is that COMPLEMENT EXTRACTION and SLASH feature passing is only possible for verbs. This is certainly too restrictive, as extraction out of subordinate clauses of type *sbar* (*welke trein zegt Gertjan dat Rob gemist heeft?*, which train does Gertjan say that Rob has missed) and out of *pp*s (*Waar gaat deze trein naar toe?*, Where does this train go to), and a number of other types of phrase is possible as well.

Sentences where the first phrase is a modifier are dealt with without appealing to SLASH. Instead, it is assumed that in sentences such as (5), the fronted elements modify the following *ques* phrase. This requires an additional (mod-topic) rule, given in (26).

(5) a. Hoe laat gaat de volgende trein naar Zwolle?
When does the next train to Zwolle leave?
b. Woensdag moet ik om tien uur in Zwolle zijn.
Wednesday, I must be in Zwolle by ten o'clock.

$$(26)\quad \begin{bmatrix} \textit{root} \\ \mathsf{sem}\ \boxed{1} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{modifier} \\ \mathsf{sem}\ \ \boxed{1} \\ \mathsf{mod}\ \langle\boxed{2}\rangle \end{bmatrix} \begin{bmatrix} \textit{ques} \\ \mathsf{slash}\ \langle\rangle \\ \mathsf{sem}\ \ \boxed{2} \end{bmatrix}$$

Of course, this account rests on the assumption that modifiers of embedded verbs or phrases cannot be fronted, an assumption which is almost certainly false in general (see Hukari and Levine (1995), for instance), but which appears to be rather unproblematic for present purposes.

2.5.7 Special grammar rules

The domain which has been selected for OVIS (information dialogues concerning public transportation) and the fact that OVIS deals with spoken language, imply that it is crucial that a number of grammatical phenomena are described in a robust manner. In particular, temporal expressions, locative expressions (names of cities and stations), and a number of typical spoken language constructions, such as greetings, occur frequently in such dialogues.

The grammar rules and lexical entries for these phenomena make use of the OVIS2 grammar formalism, but are not organised according to the linguistic principles discussed above. This is true not only for the syntax, but also for semantics. The reason for dealing with these phenomena by means of a set of more or less *ad hoc* rules and lexical entries is that the constructions discussed below are often extremely idiosyncratic. At the same time, describing the regularities that can be observed does not seem to require the overhead of the grammar architecture we assume for the rest of the grammar. The most economical and robust solution seemed therefore to encapsulate the grammar for these constructions in relatively independent grammar modules.

2.6 Semantics

The output of the grammatical analysis is a semantic, linguistically motivated and domain-independent, representation of the utterance, in the form of a Quasi Logical Form (QLF). The QLF formalism was developed in the framework of the *Core Language Engine* (CLE, (Alshawi1992; Alshawi and Crouch1992)). Since then, the formalism was used and further developed in projects such as the *Spoken Language Translator* (Agnäs et al.1994), *Clare* (Alshawi et al.1992), in the *Fracas*-project (Cooper et al.1994) and in *Trace & Unification Grammar* (Block1994). In OVIS the QLF is translated into a domain-specific *update* expression, which is passed on to the pragmatic interpretation module and dialogue manager for further processing. The dialogue manager maintains an information state to keep track of the information provided by the user. An *update* expression is an instruction for updating the information state (Veldhuijzen van Zanten1996). Below, we motivate our choice for QLFs as semantic representation language and we discuss how these QLFs are translated into updates.

2.6.1 The semantic representation language

Predicate logic, (sometimes extended with for example *generalised quantifiers* or *discourse markers*), is often used to represent the meaning of sentences. Due to its long tradition in describing semantics of natural languages it is now a well established and well understood technique. The main advantage of artificial languages like predicate logic is that they are unambiguous. An ambiguous natural language utterance will therefore correspond to more than one expression in predicate logic, one for each reading of the utterance. The disadvantage of this approach is that for very ambiguous inputs, expensive computations must be carried out to compute all readings. The alternative adopted in formalisms based on the idea of *monotonic semantic interpretation* ((Cooper et al.1994), see also (Nerbonne1992) and (Pinkal1995)) is to represent ambiguity by means of under-specification and to postpone the computation of individual readings as long as possible.

Representing ambiguity by under-specification, and postponing the computation of individual readings, has at least two computational advantages. First, parsing can benefit significantly from the fact that ambiguities which are only semantic (i.e. do not have a syntactic counterpart) are represented by a single derivation. Second, ambiguity resolution can often proceed without enumerating all possible readings of an input separately. A striking example of the latter situation is the translation of QLF's that are ambiguous with respect to quantifier-scope into a domain-specific meaning representation as it is used by the dialogue manager of the OVIS-system. The utterance in (6a), for instance, gives rise to a single QLF (6b), which could be resolved (ignoring the existential quantification over events and the fact that it is a question) to either (6c) or (6d). The domain-specific reading of (6a) (which corresponds to (6c)) is computed on the basis of (6b) directly, and thus never needs to consider the two different readings of this QLF.

(6) a. Gaat er niet een latere (trein)?
Is there not a later train?

b.
$$\begin{bmatrix} \textsf{pred} & \textsf{not} \\ \textsf{args} & \left\langle \begin{bmatrix} \textsf{pred} & \textsf{leave} \\ \textsf{args} & \left\langle [\textsf{index}\ e_1], \begin{bmatrix} \textsf{index} & \boxed{3} \\ \textsf{res} & \lambda\boxed{4}.\textsf{later\_train}(\boxed{4}) \\ \textsf{q} & \textsf{exist} \end{bmatrix} \right\rangle \end{bmatrix} \right\rangle \end{bmatrix}$$

c. $not(\exists x\ (\text{later\_train}(x) \wedge \text{leave}(e_1, x)))$

d. $\exists x\ (\text{later\_train}(x) \wedge \text{not}(\text{leave}(e_1, x)))$

2.6.2 Quasi logical form

In figure 13 we give a QLF as it is produced by the OVIS-grammar. It is a typed feature-structure, whose main components are predicative forms (*p_form*), representing relations (which may also be higher order, such as *not* and *and*), and terms. Generalised quantifiers are represented by term expressions (*t_expr*). The example in (13) contains two generalised quantifiers, corresponding to the (existentially quantified) event-variables introduced by the two verbal predicates (Davidson1967). Note that these quantifiers appear as arguments of the predicates, and thus are unscoped with respect to each other.

Our implementation of QLF in the OVIS grammar follows roughly the presentation in (Cooper et al.1994), although some of the apparatus supplied for contextual resolution in that work has been omitted. As the OVIS-grammar uses typed feature-structures, QLF's are represented as feature-structures below.

A QLF is either a *qlf-term* or a *qlf-formula*. A *qlf-term* is one of the following:

$$\begin{bmatrix} \textit{p\_form} & \\ \textsf{pred} & \textsf{want} \\ \textsf{args} & \left\langle \begin{bmatrix} \textit{t\_expr} & \\ \textsf{index} & e_1 \end{bmatrix}, \boxed{3}\ \textsf{i}, \begin{bmatrix} \textit{p\_form} & \\ \textsf{pred} & \textsf{and} \\ \textsf{args} & \left\langle \begin{bmatrix} \textit{p\_form} & \\ \textsf{pred} & \textsf{leave} \\ \textsf{args} & \left\langle \begin{bmatrix} \textit{t\_expr} & \\ \textsf{index} & e_2 \end{bmatrix}, \boxed{3} \right\rangle \end{bmatrix}, \begin{bmatrix} \textit{p\_form} & \\ \textsf{pred} & \textsf{at} \\ \textsf{args} & \left\langle \boxed{5}, \textsf{hour(4)} \right\rangle \end{bmatrix} \right\rangle \end{bmatrix} \right\rangle \end{bmatrix}$$

Fig. 13. QLF for 'Ik wil om ongeveer vier uur vertrekken' (I want to leave at about four o'clock)

- a term index,[4]
- a constant term,
- an term-expression of type *t_expr* and containing the features INDEX, RESTR and QUANT[5] (see (13)), where INDEX is a variable, RESTR is an expression of predicate logic (possibly with lambda-abstraction) and QUANT is a generalised quantifier.

A QLF formula is one of the following:[6]

- a predicate-argument formula of type *p_form*, and with features PRED and ARGS (see (13)). Predicates may be higher order, arguments may be formulas or terms,
- a formula of type *v_form* with features VAR and FORM representing a formula with lambda-abstraction (see(14b)). This is an auxiliary level of representation, introduced to facilitate the interaction between grammar-rules and lexical entries,
- a formula of type *s_form* (see(14b)), with features SCOPE and FORM. The value of SCOPE is either a variable or a list of indices indicating the relative scope of term expressions (generalised quantifiers) (see (14c)).

The definitions can best be illustrated with a simple example in which we compare a QLF expression with its corresponding formula in predicate logic. In figure 14 the sentence *Everybody speaks two languages* is given both a translation in QLF and in predicate logic. In the QLF-translation of the full sentence the scope order ($\boxed{5}$) of the two quantifiers is left unspecified. Resolving scope order amounts to instantiating $\boxed{5}$ to $[\boxed{1},\boxed{3}]$ (for everybody there are two languages that s/he speaks) or to $[\boxed{3},\boxed{1}]$ (there are two languages that everybody speaks).

4 In the original formalism *indices* and *variables* are distinguished. An index uniquely identifies a `term` expression. At this moment indices and variables have the same function in our implementation. We may need to distinguish between them later.

5 In chapter 5 of (Cooper et al.1994) term expressions also contain a slot CAT for specifying information about the lexical form and syntactic/semantic type of an expression (e.g. quantifier, pronoun, etc.) and a slot REF for specifying the (contextual) referent of an expression. We do use CAT, but have omitted it from the presentation below. We currently do not use REF.

6 In chapter 5 of (Cooper et al.1994) two more formula constructs are introduced. These are not used in the current implementation.

a. *Everybody*   *two languages*

$$\textbf{qlf}\quad \left[\begin{array}{ll} \textit{t\_expr} & \\ \textsf{index} & \boxed{1} \\ \textsf{restr} & \lambda\boxed{2}.\textsf{person}(\boxed{2}) \\ \textsf{quant} & \textsf{every} \end{array}\right] \qquad \left[\begin{array}{ll} \textit{t\_expr} & \\ \textsf{index} & \boxed{3} \\ \textsf{restr} & \lambda\boxed{4}.\textsf{language}(\boxed{4}) \\ \textsf{quant} & \textsf{two} \end{array}\right]$$

$$\textbf{pl}\quad \forall x.(person(x) \rightarrow ....) \qquad \textsc{two}\ y.(language(y) \rightarrow ....)$$

b. *speaks*

$$\textbf{qlf}\quad \left[\begin{array}{ll} \textit{v\_form} & \\ \textsf{var} & e_1 \\ \textsf{form} & \left[\begin{array}{ll} \textit{s\_form} & \\ \textsf{scope} & \boxed{5} \\ \textsf{form} & \left[\begin{array}{ll} \textit{p\_form} & \\ \textsf{pred} & \textsf{speak} \\ \textsf{args} & \langle e_1, \boxed{6}, \boxed{7}\rangle \end{array}\right] \end{array}\right] \end{array}\right]$$

$$\textbf{pl}\quad speak(e_1, ..., ...)$$

c. *Everybody here speaks two languages*

$$\textbf{qlf}\quad \left[\begin{array}{ll} \textit{s\_form} & \\ \textsf{scope} & \boxed{5} \\ \textsf{form} & \left[\begin{array}{ll} \textit{p\_form} & \\ \textsf{pred} & \textsf{speak} \\ \textsf{args} & \left\langle e_1, \left[\begin{array}{ll} \textit{t\_expr} & \\ \textsf{index} & \boxed{1} \\ \textsf{restr} & \lambda\boxed{2}.\textsf{person}(\boxed{2}) \\ \textsf{quant} & \textsf{every} \end{array}\right], \left[\begin{array}{ll} \textit{t\_expr} & \\ \textsf{index} & \boxed{3} \\ \textsf{restr} & \lambda\boxed{4}.\textsf{language}(\boxed{4}) \\ \textsf{quant} & \textsf{two} \end{array}\right] \right\rangle \end{array}\right] \end{array}\right]$$

$$\textbf{pl}\quad \forall x.(person(x) \rightarrow (\textsc{two}y.language(y) \wedge speak(e_1, x, y)))$$
$$\textsc{two}y.(language(y) \wedge \forall x.(person(x) \rightarrow speak(e_1, x, y)))$$

Fig. 14. The relation between an expression in QLF and a fomula of predicate logic

2.6.3 Construction of QLF*'s*

During grammatical analysis QLFs are constructed compositionally (see also (Alshawi and Crouch1992)). In *head complement structures* the head daughter is the syntactic as well as the semantic head of the structure. This means that the semantic content of the complement constituents is combined with the semantic content of the head. The value of the SEM feature of the head is passed up to the mother (see figure 1).

In *head modifier structures* the modifier is the semantic head. The semantics of the syntactic head of the structure is plugged into the MOD feature of the modifier. Below we will show how the semantics of the modifier is combined with the semantics of the constituent it modifies. The value of the SEM feature of the modifier is passed up to the mother.

We now discuss the semantics of various linguistic categories. *Determiners* subcategorise for a noun (see Figure 15(a.)). The semantics of the noun is unified with restriction of the determiner. Nouns introduce a *v_form* (fig. 15(b.)) Note that it is also assumed that quantifiers may scope at this point. *Adjectives* are modifiers (fig.15(c.)). They operate on structures whose semantic content is of type *v_form.* The lambda variables of the two formulas are unified and the semantic content of the structure is the conjunction of the logical formula of the adjective and the logical formula of the structure it modifies.

$$\text{a.}\quad \texttt{lex(alle,}\left[\begin{array}{ll}\textsf{sc} & \left\langle\left[\begin{array}{ll}\multicolumn{2}{l}{n}\\ \textsf{sem} & \boxed{1}\end{array}\right]\right\rangle\\ \textsf{sem} & \left[\begin{array}{ll}\multicolumn{2}{l}{t\_expr}\\ \textsf{restr} & \boxed{1}\\ \textsf{quant} & \textsf{all}\end{array}\right]\end{array}\right]\texttt{).}$$

$$\text{b.}\quad \texttt{lex(trein,}\left[\begin{array}{ll}\multicolumn{2}{l}{n}\\ \textsf{sem} & \left[\begin{array}{ll}\multicolumn{2}{l}{v\_form}\\ \textsf{var} & \boxed{1}\\ \textsf{form} & \left[\begin{array}{ll}\multicolumn{2}{l}{s\_form}\\ \textsf{form} & \left[\begin{array}{ll}\multicolumn{2}{l}{p\_form}\\ \textsf{pred} & \textsf{train}\\ \textsf{args} & \langle\boxed{1}\rangle\end{array}\right]\end{array}\right]\end{array}\right]\end{array}\right]\texttt{).}$$

$$\text{c.}\quad \texttt{lex(eerste,}\left[\begin{array}{ll}\multicolumn{2}{l}{a}\\ \textsf{sem} & \left[\begin{array}{ll}\multicolumn{2}{l}{v\_form}\\ \textsf{var} & \boxed{1}\\ \textsf{form} & \left[\begin{array}{ll}\multicolumn{2}{l}{s\_form}\\ \textsf{scope} & \boxed{2}\\ \textsf{form} & \left[\begin{array}{ll}\multicolumn{2}{l}{p\_form}\\ \textsf{pred} & \textsf{and}\\ \textsf{args} & \left\langle\left[\begin{array}{ll}\multicolumn{2}{l}{p\_form}\\ \textsf{pred} & \textsf{first}\\ \textsf{args} & \langle\boxed{1}\rangle\end{array}\right], \boxed{3}\right\rangle\end{array}\right]\end{array}\right]\end{array}\right]\\ \textsf{mod} & \left\langle\left[\begin{array}{ll}\multicolumn{2}{l}{v\_form}\\ \textsf{var} & \boxed{1}\\ \textsf{form} & \left[\begin{array}{ll}\multicolumn{2}{l}{s\_form}\\ \textsf{scope} & \boxed{2}\\ \textsf{form} & \boxed{3}\end{array}\right]\end{array}\right]\right\rangle\end{array}\right]\texttt{).}$$

Fig. 15. Examples showing the semantics of determiners, nouns, and adjectives.

The semantics of *verbs* corresponds with a *v_form* (see fig. 16a). The value of VAR is reentrant with the INDEX of the event introduced by the verb. The semantics of the subject is unified with the second element of the argument list of the verb. Intransitive verbs have two semantic arguments, corresponding to the event and subject, respectively. Transitive verbs have three arguments, where the third argument is unified with the semantics of the single element on SC.

Modal verb are *subject-control* verbs. This means that the subject of the VP-complement is controlled by the subject of the modal verb. Semantically, this means that the INDEX of the subject must be unified with the semantics of the subject of the VP-complement. Note also that we assume that assume that the SC-list of a modal verb may contain complements introduced by the VP-complement (as explained in section 2.5.3). These are not relevant for the semantics of the modal verb.

The semantics of *adverbial phrases* resembles that of *adjectives*. In figure (17) the semantics of prepositions heading a PP which acts as a verbal modifier is given. PP-modifiers introduce a conjunction, with the verbal semantics as first argument, and the prepositional semantics as second. The INDEX of the VP is the first argument of

a. $$\texttt{lex(vertrekken,}\begin{bmatrix} \mathit{v} & \\ \textsf{sem} & \begin{bmatrix} \mathit{v\_form} & \\ \textsf{var} & e_1 \\ \textsf{form} & \begin{bmatrix} \mathit{s\_form} & \\ \textsf{form} & \begin{bmatrix} \mathit{p\_form} & \\ \textsf{pred} & \textsf{leave} \\ \textsf{args} & \left\langle \begin{bmatrix} \mathit{t\_expr} & \\ \textsf{index} & e_1 \end{bmatrix}, \boxed{3} \right\rangle \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \textsf{subj} & \left\langle \begin{bmatrix} \mathit{np} & \\ \textsf{sem} & \boxed{3} \end{bmatrix} \right\rangle \end{bmatrix}\texttt{).}$$

b. $$\texttt{lex(willen,}\begin{bmatrix} \mathit{v} & \\ \textsf{sc} & \left\langle \begin{bmatrix} \mathit{v} & \\ \textsf{sc} & \boxed{2} \\ \textsf{vform} & \textsf{inf} \\ \textsf{sem} & \boxed{4} \\ \textsf{subj} & \left\langle \begin{bmatrix} \mathit{np} & \\ \textsf{sem} & \boxed{7} \end{bmatrix} \right\rangle \end{bmatrix} \mid \boxed{2} \right\rangle \\ \textsf{sem} & \begin{bmatrix} \mathit{v\_form} & \\ \textsf{var} & e_3 \\ \textsf{form} & \begin{bmatrix} \mathit{s\_form} & \\ \textsf{form} & \begin{bmatrix} \mathit{p\_form} & \\ \textsf{pred} & \textsf{want} \\ \textsf{args} & \left\langle \begin{bmatrix} \mathit{t\_expr} & \\ \textsf{index} & e_3 \end{bmatrix}, \boxed{6}, \boxed{4} \right\rangle \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \textsf{subj} & \left\langle \begin{bmatrix} \mathit{np} & \\ \textsf{sem} & \boxed{6} \begin{bmatrix} \mathit{t\_expr} & \\ \textsf{index} & \boxed{7} \end{bmatrix} \end{bmatrix} \right\rangle \end{bmatrix}$$

Fig. 16. Verbal semantics.

the predicate introduced by the preposition, the semantics of the NP-object of the preposition corresponds to the second argument.

In Dutch, temporal NP's can act as verbal modifiers:

(7) a Ik wil *zondag* vertrekken
I want to leave on Sunday

b Ik wil *drie januari* naar Amsterdam
I want to go to Amsterdam on the third of January

c Ik wil er uiterlijk *drie uur* zijn
I want to arrive at the latest at three o'clock

As NPs normally do not have a modifier semantics, there is a unary rule that transforms temporal NPs into modifiers (figure 18). The structure that is modified is specified in the MOD feature. The semantic content of the modifier is constructed as if it was a PP with P_FORM *om* (at). The semantic content of the (temporal) NP daughter is plugged into the second position of the argument list of the preposition.

$$\texttt{lex(op,}\ \left[\begin{array}{ll} \multicolumn{2}{l}{\mathit{p}} \\ \textsf{sc} & \left\langle \left[\begin{array}{ll} \multicolumn{2}{l}{\mathit{np}} \\ \textsf{sem} & \boxed{1} \end{array}\right] \right\rangle \\ \textsf{sem} & \left[\begin{array}{ll} \multicolumn{2}{l}{\mathit{v\_form}} \\ \textsf{var} & \boxed{2} \\ \textsf{form} & \left[\begin{array}{ll} \multicolumn{2}{l}{\mathit{s\_form}} \\ \textsf{scope} & \boxed{3} \\ \textsf{pred} & \textsf{and} \\ \textsf{args} & \left\langle \boxed{4}, \left[\begin{array}{ll} \multicolumn{2}{l}{\mathit{p\_form}} \\ \textsf{pred} & \textsf{on} \\ \textsf{args} & \langle \boxed{2},\boxed{1} \rangle \end{array}\right] \right\rangle \end{array}\right] \end{array}\right] \\ \textsf{mod} & \left\langle \left[\begin{array}{ll} \multicolumn{2}{l}{\mathit{v\_form}} \\ \textsf{var} & \boxed{2} \\ \textsf{form} & \left[\begin{array}{ll} \multicolumn{2}{l}{\mathit{s\_form}} \\ \textsf{scope} & \boxed{3} \\ \textsf{form} & \boxed{4} \end{array}\right] \end{array}\right] \right\rangle \end{array}\right]$$

Fig. 17. Adverbial semantics for prepositions.

$$\left[\begin{array}{ll} \multicolumn{2}{l}{\mathit{modifier}} \\ \textsf{sem} & \left[\begin{array}{ll} \multicolumn{2}{l}{\mathit{v\_form}} \\ \textsf{var} & \boxed{1} \\ \textsf{form} & \left[\begin{array}{ll} \multicolumn{2}{l}{\mathit{s\_form}} \\ \textsf{scope} & \boxed{2} \\ \textsf{form} & \left[\begin{array}{ll} \multicolumn{2}{l}{\mathit{p\_form}} \\ \textsf{pred} & \textsf{and} \\ \textsf{args} & \left\langle \boxed{3}, \left[\begin{array}{ll} \multicolumn{2}{l}{\mathit{p\_form}} \\ \textsf{pred} & \textsf{at} \\ \textsf{args} & \langle \boxed{1},\boxed{4} \rangle \end{array}\right] \right\rangle \end{array}\right] \end{array}\right] \end{array}\right] \\ \textsf{mod} & \left\langle \left[\begin{array}{ll} \multicolumn{2}{l}{\mathit{v\_form}} \\ \textsf{var} & \boxed{1} \\ \textsf{form} & \left[\begin{array}{ll} \multicolumn{2}{l}{\mathit{s\_form}} \\ \textsf{scope} & \boxed{2} \\ \textsf{form} & \boxed{3} \end{array}\right] \end{array}\right] \right\rangle \end{array}\right] \rightarrow \left[\begin{array}{ll} \multicolumn{2}{l}{\mathit{np}} \\ \textsf{nform} & \textsf{temp} \\ \textsf{sem} & \boxed{4} \end{array}\right]$$

Fig. 18. Rule `mod_np` to treat temporal noun phrases as modifiers.

2.7 Constructing updates from QLF*s*

The dialogue manager keeps track of the information provided by the user by maintaining an *information state* or *form* (Veldhuijzen van Zanten1996). This form is a hierarchical structure, with slots and values for the origin and destination of a connection, for the time at which the user wants to arrive or depart, etc. An example is given in (8a). Each user utterance leads to an *update* of the information state. An update is an instruction for updating the information in an information state. Updating can mean that new information is added or that given information is confirmed, retracted or corrected. For example, given the information state in (8a), the update in (8b) (which might be the translation of *No, I do not want to travel to Leiden but to Abcoude!*) leads to the information state in (8c). The # -operator in (8b) indicates that the information within its scope (indicated by square brackets) is to be retracted, and the '!'-operator indicates a correction.

(8) a. $$\begin{bmatrix} \textsf{travel} & \begin{bmatrix} \textsf{origin} & \begin{bmatrix} \textsf{place} & [\textsf{town groningen}] \\ \textsf{moment} & \begin{bmatrix}\textsf{at} & \begin{bmatrix}\textsf{time} & [\textsf{clock\_hour 3}]\end{bmatrix}\end{bmatrix} \end{bmatrix} \\ \textsf{destination} & \begin{bmatrix}\textsf{place} & [\textsf{town leiden}]\end{bmatrix} \end{bmatrix} \end{bmatrix}$$

b. travel.destination. ([# place.town.leiden]; [! place.town.abcoude])

c. $$\begin{bmatrix} \textsf{travel} & \begin{bmatrix} \textsf{origin} & \begin{bmatrix} \textsf{place} & [\textsf{town groningen}] \\ \textsf{moment} & \begin{bmatrix}\textsf{at} & \begin{bmatrix}\textsf{time} & [\textsf{clock\_hour 3}]\end{bmatrix}\end{bmatrix} \end{bmatrix} \\ \textsf{destination} & \begin{bmatrix}\textsf{place} & [\textsf{town abcoude}]\end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The result of parsing is a QLF, a linguistically motivated and domain-independent representation of the meaning of a sentence. The translation of a QLF into a domain-specific *update* is done by applying translation-rules to the individual parts of a QLF. These translation rules may be context-sensitive. In particular, some parts of the QLF provide the context which determines how other parts are to be translated. For example, the QLF in (9) (corresponding to the phrase *leave at four o'clock* contains two *p_forms*, one for the predicate *leave* and one for *four o'clock*. The second gives rise to an update expression `moment.at.time.clock_hour.4`. The first provides the contextual information that the moment referred to is a departure-time. The translation can therefore be extended to `origin.moment.at.time.clock_hour.4`. There is no linguistic information which indicates that a special update-operator has to be used. In such cases, it is assumed that the information is new, and thus the assert-operator ('=') can be used, giving rise to the translation for the full phrase: `origin.moment.at.[= time.clock_hour.4]`.

(9) $$\begin{bmatrix} \textit{p\_form} & \\ \textsf{pred} & \textsf{and} \\ \textsf{args} & \left\langle \begin{bmatrix} \textit{p\_form} & \\ \textsf{pred} & \textsf{leave} \\ \textsf{args} & \left\langle \begin{bmatrix} \textit{t\_expr} & \\ \textsf{index} & e_1 \end{bmatrix}, \boxed{3} \right\rangle \end{bmatrix}, \begin{bmatrix} \textit{p\_form} & \\ \textsf{pred} & \textsf{at} \\ \textsf{args} & \langle e_1, \textsf{hour(4)} \rangle \end{bmatrix} \right\rangle \end{bmatrix}$$

Contextual translation is a powerful technique. For instance, the utterance *Groningen Amsterdam* gives rise to a conjunctive QLF, containing two term expressions for locations. Translating each of the conjuncts individually would make it impossible to decide whether an origin or destination location is being specified. By translating the conjunction in one step (and assuming that the order of conjuncts corresponds to the order in the utterance), we can resolve the first locative to origin and the second to destination. As another example, the adverb *graag* is ignored in the translation from QLF to update if it occurs as part of a full sentence (*ik wil graag naar Amsterdam*, 'I would like to go to Amsterdam'), but is translated as 'yes' (i.e. a confirmation of information provided by the system) if it occurs in isolation. Such a translation is motivated by dialogues of the following type:

(10)	[system:]	Dus U wilt van Amsterdam naar Groningen reizen?
		So you want to travel from Amsterdam to Groningen?
	[user:]	Graag.
		Please.

Similarly, the translation of the negations *nee* (no) and *niet* (not) depends on context. If the two occur in isolation, they indicate a denial of information provided by the system. However, if *nee* is followed by another phrase, say a locative, it signals a correction (11a), whereas if *niet* is followed by another phrase, it signals a denial (11b).

(11) a. 'Nee, naar Assen' (*No, to Assen*)
`destination.[!place.assen]`
b. 'Niet naar Assen' (*Not to Assen*)
`destination.[#place.assen]`

It should be noted that the translation of QLF's to updates uses primarily the information provided by NP's, PP's and adverbs. Verbs typically provide the context for translating other parts of the QLF. Also, as quantification plays no role in updates, the scope of generalised quantifiers can be largely ignored. Thus, we are able to translate QLF's into domain-specific meanings without resolving quantifier scope.

3 Robust parsing of word-graphs

3.1 Word-graphs

The input to the NLP module consists of word-graphs produced by the speech recogniser (Oerder and Ney1993). A word-graph is a compact representation for all sequences of words that the speech recogniser hypothesises for a spoken utterance. The states of the graph represent points in time, and a transition between two states represents a word that may have been uttered between the corresponding points in time. Each transition is associated with an *acoustic score* representing a measure of confidence that the word perceived there was actually uttered. These scores are negative logarithms of probabilities and therefore require addition as opposed to multiplication when two scores are combined. An example of a typical word-graph is given as the first graph in figure 19.

At an early stage, the word-graph is normalised to eliminate the *pause transitions*. Such transitions represent periods of time for which the speech recogniser hypothesises that no words are uttered. After this optimisation, the word-graph contains exactly one start state and one or more final states, associated with a score, representing a measure of confidence that the utterance ends at that point. The word-graphs in figure 19 provide an example.

From now on, we will assume word-graphs are normalised in this sense. Below, we refer to transitions in the word-graph using the notation $trans(v_i, v_j, w, a)$ for a transition from state v_i to v_j with symbol w and acoustic score a. Let $final(v_i, a)$ refer to a final state v_i with acoustic score a.

3.2 Parsing word-graphs

The normalized word-graph is parsed by an appropriate parser. Parsing algorithms for strings can be generalized to parse such word-graphs (for some examples cf. van Noord (1995)). In the ideal case, the parser will find a path in the word-graph that

Fig. 19. Word-graph and normalized word-graph for the utterance *Zondag vier februari* (Sunday Februari fourth). The special label # in the first graph indicates a pause transition. These transitions are eliminated in the second graph.

can be assigned an analysis according to the grammar, such that the path covers the complete time span of the utterance, i.e. the path leads from the start state to a final state. The analysis gives rise to an update of the dialogue state, which is then passed on to the dialogue manager.

However, often no such paths can be found in the word-graph, due to:

- errors made by the speech recognizer,
- linguistic constructions not covered in the grammar, and
- irregularities in the spoken utterance.

Even if no full analysis of the word-graph is possible, it is usually the case that useful information can be extracted from the word-graph. Consider for example the utterance:

(12) Ik wil van van Assen naar Amsterdam
I want from from Assen to Amsterdam

The grammar will not assign an analysis to this utterance due to the repeated preposition. However, it would be useful if the parser would discover the prepositional phrases *van Assen* and *naar Amsterdam* since in that case the important information contained in the utterance can still be recovered. Thus, in cases where no full analysis is possible we would like to fall back on an approach reminiscent of concept spotting. The following proposal implements this idea.

Firstly, the grammar is defined in such a way that each *maximal projection* such as S, NP, PP, etc., can be analysed as a top category. This is well-motivated because utterances very often consist of a single NP or PP (section 3.3).

Often, the task of the parser is to discover all instances of the top category from the start state of the word-graph to a final state. But in our case, we require that the parser discovers all instances of the top category *anywhere in the word-graph*, i.e. for all partial paths in the word-graph. This has the desired effect for example (12): both PPs will be found by the parser.

Thus we require that the parser finds all major categories anywhere in the word-graph. If a bottom-up chart parser is used, then we might use the inactive chart items for this purpose. However, since we do not want to be forced to a particular parsing strategy, we have chosen to adopt a different approach. In section 3.4 we show that in a logic programming setting the use of underspecification of the state names associated with the top-most goal obtains the desired effect, without loss of efficiency.

Therefore, after the parser has finished, we have a word-graph annotated with a number of instances of top categories. For each of these categories we are interested in the word-graph state where this category starts (v_i), the word-graph state where this category ends (v_j), the sequence of symbols associated with this category (x), the accumulated acoustic score (a), and the qlf (q). Let $parsed(v_i, v_j, x, a, q)$ refer to such categories.

We are interested in paths from the start state to the final state consisting of a number of categories and transitions in the word-graph (the latter are called *skips*). The problem consists in finding the optimal path, according to a number of criteria. This problem is formalized by defining the annotated word-graph as a directed acyclic graph (section 3.5). The vertices of this graph are the states of the word-graph; the edges are the transitions of the word-graph and the categories found by the parser.

The criteria which are used to favor some paths over other paths are expressed as a weight function on the edges of the graph. The criteria we might take into account are discussed in section 3.6. For instance, a typical criterion will favor paths consisting of a small number of categories, and a small number of skips. The case in which the parser found a full analysis from the start state of the word-graph to a final state then reduces to a special case: the analysis solely consisting of that category will be favored over sequences of partial analyses.

Obviously, it is not a good idea to generate all possible sequences of categories and skips, and then to select the best path from this set: in typical word-graphs there are simply too many different paths. If a certain uniformity requirement on weights is met, however, then efficient graph search algorithms are applicable. The particular algorithm implemented in OVIS2, namely a variant of the DAG-SHORTEST-PATH algorithm (Cormen, Leiserson, and Rivest1990) is discussed in section 3.7.

The criteria used to determine the best path may also include Ngram statistics. It turns out that in those cases some complications arise in the definition of the annotated word-graph. This is explained in section 3.8.

In a previous implementation (Nederhof et al.1997) we used a version of Dijkstra's algorithm. A comparison is presented in section 3.9. Finally, section 3.10 discusses methods in which the parser is applied only to a single path of the word-graph.

3.3 Grammar

We require that grammatical analysis finds all maximal projections anywhere in the input word-graph. This implies that the top category of the grammar should be defined in such a way that it derives each of these maximal projections. For this reason, the grammar contains the declaration:

(27) `top_category(X) :- X => start.`

Furthermore, there are unary rules rewriting this start category into each of the relevant maximal projections. One such rule is:

(28)
```
rule(start_np,Start,[Np]) :-
      Start => start, Np => np,
      Start:sem <=> Np:sem.
```

Similar rules are defined for `pp, sbar, root, advp`, etc.

3.4 Parser

Five different parsing algorithms were implemented and compared (a bottom-up Earley parser, an inactive chart parser, an LR parser, a left-corner parser and a head-corner parser). The most efficient parser (both in terms of CPU-time and memory usage) for this application turned out to be a head-corner parser implemented with goal-weakening and selective memoization. The head-corner parser is presented in detail in van Noord (1997a).

In order to apply this (or any of the other) parser(s) for robust processing, we use underspecification of the state names for the input parse goal in order to parse the start category *anywhere in the word-graph.* Normally the parser will be called using a goal such as the following:

(29) `?- parse(start(Sem),q0,q16).`

indicating that we want to find a path from state `q0` to `q16` which can be analysed as a category `start(Sem)` (a sentence with a semantic representation that is yet to be discovered). If we want to recognize top categories at all positions in the input, then we can simply generalize the parse goal to:

(30) `?- parse(start(Sem),_,_).`

Now it may seem that such an underspecified goal will dramatically slow down the parser, but this turns out to be a false expectation, at least for the head-corner and left-corner parsers. In fact we have experienced no such decrease in efficiency. This can only be understood in the light of the use of memoization: even though we now have a much more general goal, the number of different goals that we need to solve is much smaller.

3.5 Annotated word-graph

An annotated word-graph is a word-graph annotated with the results of the parser. Such an annotated word-graph is defined with respect to an input word-graph (given by the functions *trans* and *final*) and with respect to the results of parsing (given by the function *parsed*).

The *annotated word-graph* is a directed acyclic graph (V, E) where

- V is the set of vertices consisting of the states of the word-graph $v_0 \dots v_n$, and a new vertex v_{n+1}. v_0 is the start state. v_{n+1} is the final state.
- E is the set of edges consisting of:
  1. *skip* edges. For all $trans(v_i, v_j, w, a)$ there are edges $(v_i, v_j, w, a, \epsilon)$.
  2. *category* edges. For all $parsed(v_i, v_j, x, a, q)$ there are edges (v_i, v_j, x, a, q).
  3. *stopping* edges. For all $\text{final}(v_i, a)$ there are edges $(v_i, v_{n+1}, \epsilon, a, \epsilon)$.

3.6 Weights

The weights that are associated with the edges of the graph can be sensitive to the following factors.

- Acoustic score. Obviously, the acoustic score present in the word-graph is an important factor. The acoustic scores are derived from probabilities by taking the negative logarithm. For this reason we aim to minimize this score. If edges are combined, then we have to sum the corresponding acoustic scores.

- Number of 'skips'. We want to minimize the number of skips, in order to obtain a preference for the maximal projections found by the parser. Each time we select a skip edge, the number of skips is increased by 1.
- Number of maximal projections. We want to minimize the number of such maximal projections, in order to obtain a preference for more extended linguistic analyses over a series of smaller ones. Each time we select a category edge, this number is increased by 1.
- Quality of the QLF in relation to the context. We are experimenting with evaluating the quality of a given QLF in relation to the dialogue context, in particular the question previously asked by the system (Koeling1997).
- Ngram statistics. We have experimented with bigrams and trigrams. Ngram scores are expressed as negative logarithms of probabilities. This implies that combining Ngram scores requires addition, and that lower scores imply higher probability.

The only requirement we make to ensure that efficient graph searching algorithms are applicable is that weights are *uniform.* This means that a weight for an edge leaving a vertex v_i is independent of how state v_i was reached.

In order to be able to compute with such multidimensional weights, we express weights as tuples $\langle c_1, \ldots, c_k \rangle$. For each cost component c_i we specify an initial weight, and we need to specify for each edge the weight of each cost component. To specify how weights are updated if a path is extended, we use the function uw that maps a pair of a multidimensional weight and an edge a to multidimensional weight. [7] Moreover, we need to define an ordering on such tuples. In order to experiment with different implementations of this idea we refer to such a collection of specifications as a *method*. Summarizing, such a weight method is a triple $W = \langle ini, uw, \prec \rangle$ where

1. ini is the initial weight;
2. uw is the update weight function;
3. $\prec$ is an ordering on weights

Example: speech method. As a trivial example of such a method, consider the problem of finding the best path through the word-graph ignoring all aspects but the acoustic scores present in the word-graph. In order to implement a method W^{speech} to solve this problem, we define weights using a unary tuple $\langle c \rangle$. The initial weight is $ini = \langle 0 \rangle$ and uw is defined as follows:

(31) $$uw(\langle c \rangle, (v_i, v_j, w, a, q)) = \begin{cases} \langle c + a \rangle & \text{for skip edges} \\ \langle \infty \rangle & \text{for category edges} \\ \langle c + a \rangle & \text{for stopping edges} \end{cases}$$

[7] We do not define a weight function on edges, but we specify how weights are updated if a path is extended, for generality. This approach allows e.g. for the possibility that different cost components employ different operations for combining weights. For example, some cost components may use addition (e.g. for weights which are expressed as negative logarithms derived from probabilities), whereas other cost components may require multiplication (e.g. for probabilities).

Note that we specify an infinite weight for category edges because we want to ignore such edges for this simple method (i.e. we are simply ignoring the results of the parser). We define an ordering $\prec$ on such tuples, simply by stating that $\langle w \rangle \prec \langle w' \rangle$ iff $w < w'$.

Example: nlp_speech method. A more interesting example is provided by the following method which not only takes into account acoustic scores, but also the number of skip edges and category edges. Weights are expressed as $\langle c_1, c_2, c_3 \rangle$, where c_1 is the number of skips, c_2 is the number of categories, and c_3 is the acoustic score.

We define $ini = \langle 0, 0, 0 \rangle$ and uw is defined as follows.

(32) $$uw(\langle c_1, c_2, c_3 \rangle, (v_i, v_j, w, a, q)) : \begin{cases} \langle c_1 + 1, c_2, c_3 + a \rangle & \text{for skip edges} \\ \langle c_1, c_2 + 1, c_3 + a \rangle & \text{for category edges} \\ \langle c_1, c_2, c_3 + a \rangle & \text{for stopping edges} \end{cases}$$

Finally, we define the ordering on such tuples:

(33) $$\langle c_1, c_2, c_3 \rangle \prec \langle c_1', c_2', c_3' \rangle \text{ iff} : \begin{cases} c_1 < c_1' \text{ or} \\ c_1 = c_1' \text{ and } c_2 < c_2' \text{ or} \\ c_1 = c_1' \text{ and } c_2 = c_2' \text{ and } c_3 < c_3' \end{cases}$$

3.7 Search algorithm

The robustness component can be characterised as a search in the annotated word-graph. The goal of the search is the best path from v_0 to v_{n+1}. This search reduces to a well-known graph search problem, namely the problem of finding the shortest path in a directed acyclic graph with *uniform* weights.

We use a variant of the DAG-SHORTEST-PATH algorithm (Cormen, Leiserson, and Rivest1990). This algorithm finds shortest paths for uniformly weighted directed acyclic graphs. The first step of the algorithm is a topological sort of the vertices of the graph. It turns out that the state names of the word-graph that we obtain from the speech recogniser are already topologically sorted: state names are integers, and edges always connect to larger integers. The second step of the algorithm maintains an array A which records for each state v_k the weight associated with the best path known from v_0 to v_k. A similar array, P, is used to represent for each state the history of this best path, as a sequence of QLFs (since that is what we want to obtain eventually).

The first step of the algorithm initialises these arrays such that for each state $v_i (i \neq 0) A[v_i] = \infty$, and $P[v_i] =$ NIL. For v_0 we specify $A[v_0] = ini$ and $P[v_0] = \epsilon$. After this initialisation phase the algorithm treats each edge of the graph in topologically sorted order of the source vertex, as follows:

(34)
```
foreach state v_i (in topologically sorted order)
        do
        foreach edge (v_i, v_j, w, a, q)
                do
                relax (v_i, v_j, w, a, q)
```

The function `relax` is defined on edges and updates the arrays if a better path to a vertex has been found:

(35) **function relax** (v_i, v_j, w, a, q)
  if $uw(A[v_i], (v_i, v_j, w, a, q)) \prec A[v_j]$
  then $A[v_j] =: uw(A[v_i], (v_i, v_j, w, a, q))$
    $P[v_j] =: P[v_i].q$

When the algorithm finishes, $P[v_{n+1}]$ constitutes the sequence of QLFs associated with the best path found. The weight of this path is given by $A[v_{n+1}]$. This algorithm is efficient. Its running time is $O(V + E)$, where V is the number of vertices and E is the number of edges. Therefore, it can be expected that this part of processing should not decrease parsing efficiency too much, since the number of edges is $O(V^2)$.[8] For a more detailed account of the correctness and complexity of this algorithm, see Cormen, Leiserson, and Rivest (1990). [9]

A simple generalisation of the algorithm has been implemented in order to obtain the N best solutions. In this case we maintain in the algorithm for each vertex the N best paths found so far. Such a generalisation increases the worst-case complexity by only a constant factor, and is very useful for development work.

3.8 Complications for Ngrams

In this section we want to extend the *nlp_speech* method to take into account Ngram probabilities. Obviously, we can extend the weight tuple with a further argument which expresses the accumulated weight according to the Ngram probabilities. However, a potential problem arises. If we extend a given path by using the transition labeled w, then we want to take into account the probability of reading this word w given the previous $N-1$ words. However note that in the definition of the annotated word-graph given above these words are not readily available. Even if we make sure that each path is associated with the last words seen so far, we must be careful that weights remain uniform.

The solution to this problem is to alter the definition of the graph, in such a way that the relevant $N-1$ words are part of the vertex. If we want to take into account Ngram probabilities ($N = 2, 3, \ldots$), then the vertices of the graph will be tuples $(v, w_1 \ldots w_{N-1})$ where v is a state of the word-graph as before, and $w_1 \ldots w_{N-1}$ are the previous $N-1$ words. For example, in the case of bigrams ($N = 2$), vertices are pairs consisting of a word-graph state and a word (the previous word). A number of special symbols $y_{N-1} \ldots y_1$ is introduced as beginning-of-sentence markers. The start vertex is now $(v_0, y_{N-1} \ldots y_1)$. The notation $x : k$ is used to refer to the last k elements of x.

[8] This compares well with the $O(V^3)$ complexity which can be obtained for most parsers.
[9] Note that the algorithm is different from the Viterbi algorithm. The latter algorithm finds the best path through a possibly cyclic weighted graph, *for a given sequence of observed outputs*. In the current application we require an algorithm to find the best path in an *acyclic* weighted graph (without an additional observed output sequence).

The *annotated word-graph for Ngrams* is a weighted graph (V, E) and some fixed N, such that:

- V is a set of pairs $(v, w_1 \dots w_{N-1})$ where v is a word-graph state and w_i are labels in the word-graph. The start vertex is $(v_0, y_{N-1} \dots y_1)$; the final vertex is (v_{n+1}, ϵ).
- E is the set of edges consisting of:
  1. *skip* edges. For all $\mathit{trans}(v_i, v_j, w, a)$, and all vertices $V_i = (v_i, x)$ and $V_j = (v_j, xw : N-1)$, there are edges $(V_i, V_j, w, a, \epsilon)$.
  2. *category* edges. For all $\mathit{parsed}(v_i, v_j, x_2, a, q)$, and for all vertices $V_i = (v_i, x_1)$ and $V_j = (v_j, x_1x_2 : N-1)$, there are edges (V_i, V_j, x_2, a, q).
  3. *stopping* edges. For all final(v_i, a) and for all vertices $V_i = (v_i, x)$ there are edges $(V_i, (v_{n+1}, \epsilon), \epsilon, a, \epsilon)$.

Example: nlp_speech_trigram method. The start state of the graph search now is the vertex (v_0, y_2y_1). Weights are expressed as 4-tuples by extending the triples of the *nlp_speech* method with a fourth component expressing trigram weights. These trigram weights are expressed using negative logarithms of (estimates of) probabilities. Let *tri* be the function which returns for a given sequence of three words this number. Moreover, the definition of this function is extended for longer sequences of words in the obvious way by defining $\mathit{tri}(w_0w_1w_2x) = \mathit{tri}(w_0w_1w_2) + \mathit{tri}(w_1w_2x)$.

The initial weight is defined as $\mathit{ini} = \langle 0, 0, 0, 0\rangle$. Weights are updated as follows:

(36) $$\mathit{uw}(\langle c_1, c_2, c_3, c_4\rangle, ((v_i, w_0w_1), (v_j, y), x, a, q)) = \begin{cases} \langle c_1+1, c_2, c_3+a, c_4+\mathit{tri}(w_0w_1x)\rangle & \text{for skip edges} \\ \langle c_1, c_2+1, c_3+a, c_4+\mathit{tri}(w_0w_1x)\rangle & \text{for category edges} \\ \langle c_1, c_2, c_3+a, c_4\rangle & \text{for stopping edges} \end{cases}$$

Finally, we define an ordering on such tuples. The function *total* is defined on tuples as follows. Here k_{nlp} and k_{wg} are constants.

(37) $$\mathit{total}(\langle c_1, c_2, c_3, c_4\rangle) = c_4 + k_{\mathrm{nlp}} * (c_1 + c_2) + k_{\mathrm{wg}} * c_3.$$

We then define the ordering as:

(38) $$\langle c_1, c_2, c_3, c_4\rangle \prec \langle c'_1, c'_2, c'_3, c'_4\rangle \text{ iff } \mathit{total}(\langle c_1, c_2, c_3, c_4\rangle) < \mathit{total}(\langle c'_1, c'_2, c'_3, c'_4\rangle).$$

3.9 Comparison with Dijkstra's algorithm

In a previous version of the implementation we used a generalised version of DIJKSTRA's algorithm (Dijkstra1959), (Nilsson1971), (Cormen, Leiserson, and Rivest1990), instead of the DAG-SHORTEST-PATH presented above. Dijkstra's algorithm is more general in that it is not restricted to acyclic graphs. On the other hand, however, Dijkstra's algorithm requires that weights on edges are positive (paths can only get worse if they are extended). A potential advantage of Dijkstra's algorithm for our purposes is that the algorithm often does not have to investigate all edges. If edges are relatively expensive to compute, then Dijkstra's algorithm might turn out to be faster.

For instance, we can obtain a modest increase in efficiency by exploiting Dijkstra's algorithm if we delay some of the work the parser has to do for some category q, until the robustness component actually has a need for that category q. Since Dijkstra's algorithm will not visit every q, the amount of work is reduced. We exploited this in our implementation as follows. The parser works in two phases. In the first phase (recognition) no semantic constraints are taken into account (in order to pack all ambiguities). In the second phase semantic constraints are applied. This second phase can then be delayed for some category q until Dijkstra's algorithm uses an edge based on q. For a number of categories, therefore, this second phase can be skipped completely.

However, we had three reasons for preferring the DAG-SHORTEST-PATH algorithm given above. Firstly, this algorithm is simpler than Dijkstra's algorithm. Secondly, negative weights do show up in a number of circumstances. And thirdly, the expected efficiency gain was not observed in practice.

An example where negative weights may show up is the following. Suppose we define a method which takes into account Ngram scores but nothing else, i.e. all other sources of information such as acoustic scores are ignored. It turns out that a straightforward implementation of such a method is non-optimal since it will favour paths in the word-graph consisting of a small number of long words over paths (of the same duration) consisting of a larger number of smaller words, only because more scores have to be added. A simple and effective way to eliminate this effect, is to subtract a constant from each score. However, this subtraction may yield negative numbers.

3.10 Best-first methods

Rather than integrating parsing and disambiguation of the word-graph as a single procedure, as we proposed above, it is also possible to try to disambiguate the word-graph first, and then use the parser to parse the resulting path in the word-graph.

We have implemented two versions of this approach. Both versions use the search algorithm discussed above, by applying a method which takes into account the acoustic scores and Ngram scores. One version uses $N = 2$, the other version uses $N = 3$. In section 4 we refer to these two methods as *best_1_bigram* and *best_1_trigram* respectively.

We have experimented with such methods in order to evaluate the contribution of grammatical analysis to speech recognition. If, for instance, the integrated method *nlp_speech_trigram* performs significantly better than *best_1_trigram* then we can conclude that grammatical analysis improves speech recognition. The results below, however, do not permit this conclusion.

Table 1. **Characterization of test set (1).**

	graphs	transitions	words	t/w	w/g	t/g	max(t/g)
input	1000	48215	3229	14.9	3.2	48.2	793
normalized	1000	73502	3229	22.8	3.2	73.5	2943

4 Evaluation

We present a number of results to indicate how well the NLP component currently performs. In the NWO Priority Programme, two alternative natural language processing modules are developed in parallel: the 'grammar-based' module described here, and a 'data-oriented' (statistical, probabilistic, DOP) module. Both of these modules fit into the system architecture of OVIS. The DOP approach is documented in a number of publications (Scha1990; Bonnema, Bod, and Scha1997; Bod and Scha1997).

In order to compare both NLP modules, a formal evaluation has been carried out on 1000 new, unseen, representative word graphs (obtained using the latest version of the speech recognizer). Full details on the evaluation procedure, and all evaluation results, are described elsewhere (van Noord1997b; Bonnema, van Noord, and van Zanten1998). For these word graphs, annotations were provided by our project partners consisting of the actual sentences ('test sentences'), and updates ('test updates').

The Ngram models used by our implementation were constructed on the basis of a corpus of almost 60K user utterances (almost 200K words).

Some indication of the difficulty of the test-set of 1000 word-graphs is presented in table 1, both for the input word-graphs and for the normalised word-graphs. The table lists the number of transitions, the number of words of the actual utterances, the average number of transitions per word, the average number of words per graph, the average number of transitions per graph, and finally the maximum number of transitions per graph. The number of transitions per word in the normalized word-graph is an indication of the additional ambiguity that the parser encounters in comparison with parsing of ordinary strings.

A further indication of the difficulty of this set of word-graphs is obtained if we look at the word and sentence accuracy obtained by a number of simple methods. The string comparison on which sentence accuracy and word accuracy are based is defined by the minimal number of substitutions, deletions and insertions that is required to turn the first string into the second (Levenshtein distance d). Word accuracy is defined as $1 - \frac{d}{n}$ where n is the length of the actual utterance.

The method *speech* only takes into account the acoustic scores found in the word-graph. The method *possible* assumes that there is an oracle which chooses a path such that it turns out to be the best possible path. This method can be seen as a natural upper bound on what can be achieved. The methods *bigram* (*trigram*) report on a method which *only* uses a bigram (trigram) language model. The methods

Table 2. **Characterization of test set (2).**

method	WA	SA
speech	69.8	56.0
possible	90.4	83.7
bigram	69.0	57.4
trigram	73.1	61.8
speech_bigram	81.1	73.6
speech_trigram	83.9	76.2

speech_bigram (*speech_trigram*) use a combination of bigram (trigram) statistics and the speech score.

4.1 Efficiency

Table 3 reports on the efficiency of the NLP components for the set of 1000 word-graphs and test utterances. The first two rows present the results for sentences; the remaining rows provide the results for word-graphs. Listed are respectively the average number of milliseconds per input; the maximum number of milliseconds; and the maximum space requirements (per word-graph, in Kbytes).

For most word-graphs we used the nlp_speech_trigram method as described above. For large word-graphs (more than 100 transitions), we first selected the best path in the word-graph based on acoustic scores and N-gram scores only. The resulting path was then used as input for the parser. In the case of these large word-graphs, N=2 indicates that bigram scores were used, for N=3 trigram scores were used.

CPU-time includes tokenizing the word-graph, removal of pause transitions, lexical lookup, parsing, the robustness/disambiguation component, and the production of an update expression. [10]

For word-graphs the average CPU-times are actually quite misleading because CPU-times vary enormously for different word-graphs. For this reason, we present in table 4 the proportion of word-graphs (in %) that can be treated by the NLP component within a given amount of CPU-time (in milliseconds).

4.2 Accuracy

The results for word accuracy given above provide a measure for the extent to which linguistic processing contributes to speech recognition. However, since the main task of the linguistic component is to analyze utterances semantically, an equally important measure is *concept accuracy*, i.e. the extent to which semantic

[10] For the grammar-based methods, CPU-time was measured on a HP 9000/780 machine running HP-UX 10.20, with SICStus Prolog 3 patch level 3. The statistics for the data-oriented module were obtained on a Silicon Graphics Indigo with a MIPS R10000 processor, running IRIX 6.2.

Table 3. **Efficiency (1).**

input	method	mean msec	max msec	max Kbytes
test sentence	data-oriented	91	8632	14064
test sentence	grammar-based	28	610	524
word graphs	data-oriented	7011	648671	619504
word graphs	grammar-based N=2	298	15880	7143
word graphs	grammar-based N=3	1614	690800	34341

Table 4. **Efficiency (2).**

method	100	500	1000	5000	10000
data-oriented	52.7	70.8	76.6	90.6	94.2
grammar-based N=2	58.6	87.0	94.6	99.5	99.8
grammar-based N=3	58.5	78.9	87.3	96.7	98.7

analysis corresponds with the meaning of the utterance that was actually produced by the user.

For determining concept accuracy, we have used a semantically annotated corpus of 10K user responses. Each user response was annotated with an *update* representing the meaning of the utterance that was actually spoken. The annotations were made by our project partners in Amsterdam, in accordance with the existing guidelines (Veldhuijzen van Zanten1996).

Updates take the form described in section 2.6. An update is a logical formula which can be evaluated against an information state and which gives rise to a new, updated information state. The most straightforward method for evaluating concept accuracy in this setting is to compare (the normal form of) the update produced by the grammar with (the normal form of) the annotated update. A major obstacle for this approach, however, is the fact that very fine-grained semantic distinctions can be made in the update-language. While these distinctions are relevant semantically (i.e. in certain cases they may lead to slightly different updates of an information state), they can often be ignored by a dialogue manager. For instance, the two updates below are semantically not equivalent, as the ground-focus distinction is slightly different.

(39)
```
user.wants.destination.place.([# town.leiden];[! town.abcoude])
user.wants.destination.([# place.town.leiden];[! place.town.abcoude])
```

However, the dialogue manager will decide in both cases that this is a correction of the destination town.

Since semantic analysis is the input for the dialogue manager, we have measured concept accuracy in terms of a simplified version of the update language. Inspired

Table 5. **Accuracy.**

Input	Method	String accuracy WA	SA	Semantic accuracy match	precision	recall	CA
test sentence	data-oriented	N/A	N/A	93.0	94.0	92.5	91.6
test sentence	grammar-based	N/A	N/A	95.7	95.7	96.4	95.0
word-graph	data-oriented	76.8	69.3	74.9	80.1	78.8	75.5
word-graph	grammar-based N=2	82.3	75.8	80.9	83.6	84.8	80.9
word-graph	grammar-based N=3	84.2	76.6	82.0	85.0	86.0	82.6

by a similar proposal in Boros et al. (1996), we translate each update into a set of *semantic units*, where a unit in our case is a triple ⟨`CommunicativeFunction, Slot, Value`⟩. For instance, the two examples above both translate as
⟨ `denial, destination_town, leiden` ⟩
⟨ `correction, destination_town, abcoude` ⟩
Both the updates in the annotated corpus and the updates produced by the system were translated into semantic units.

Semantic accuracy is given in table 5 according to four different definitions. Firstly, we list the proportion of utterances for which the corresponding semantic units exactly match the semantic units of the annotation (*match*). Furthermore we calculate *precision* (the number of correct semantic units divided by the number of semantic units which were produced) and *recall* (the number of correct semantic units divided by the number of semantic units of the annotation). Finally, following Boros et al. (1996), we also present concept accuracy as

$$CA = 100\left(1 - \frac{SU_S + SU_I + SU_D}{SU}\right)\%$$

where SU is the total number of semantic units in the translated corpus annotation, and SU_S, SU_I, and SU_D are the number of substitutions, insertions, and deletions that are necessary to make the translated grammar update equivalent to the translation of the corpus update.

We achieve the results listed in table 5 for the test-set of 1000 word-graphs. String accuracy is presented in terms of word-accuracy (WA) and sentence accuracy (SA).

Conclusion

The results given above lead to the following conclusions.

- Sophisticated grammatical analysis is fast enough for practical spoken dialogue systems.
- Moreover, grammatical analysis is effective for the purposes of the present application. Almost all user utterances can be analysed correctly. This is somewhat surprising. Typically, linguistic ambiguities are a major obstacle

for practical NLP systems. The current application is very simple in the sense that such linguistic ambiguities do not seem to play a significant role. The ambiguities introduced by the speech recognizer (as multiple paths in the word-graph) are a far more important problem.

- Grammatical analysis does not seem to help much to solve the problem of disambiguating the word-graph. The best method incorporating grammatical analysis performs about as well as a method which solely uses N-gram statistics and acoustic scores for disambiguation of the word-graph. However, in the latter case grammatical analysis of the type proposed here is useful in providing a robust analysis of the best path.

We have argued in this paper that sophisticated grammatical analysis in combination with a robust parser can be applied successfully as an ingredient of a spoken dialogue system. Grammatical analysis is thereby shown to be a viable alternative to techniques such as concept spotting. We showed that for a state-of-the-art application (public transport information system) grammatical analysis can be applied efficiently and effectively. It is expected that the use of sophisticated grammatical analysis will allow for easier construction of linguistically more complex spoken dialogue systems.

Acknowledgements

This research was carried out within the framework of the Priority Programme Language and Speech Technology (TST). The TST-Programme is sponsored by NWO (Dutch Organization for Scientific Research).

References

Agnäs, M-S., H. Alshawi, I. Bretan, D. Carter, K. Ceder, M. Collins, R. Crouch, V. Digalakis, B. Ekholm, B. Gambäck, J. Kaja, J. Karlgren, B. Lyberg, P. Price, S. Pulman, M. Rayner, C. Samuelsson, and T. Svensson. 1994. Spoken Language Translator: First-year report. Technical report, SICS and SRI Cambridge, Cambridge Computer Research Centre. Available as crc043/paper.ps.Z from http://www.cam.sri.com/tr/.

Allen, J.F., B.W. Miller, E.K. Ringger, and T. Sikorski. 1996. A robust system for natural spoken dialogue. In *34th Annual Meeting of the Association for Computational Linguistics*, pages 62–70, University of California, Santa Cruz.

Alshawi, Hiyan, editor. 1992. *The Core Language Engine*. ACL-MIT press, Cambridge Mass.

Alshawi, Hiyan, David Carter, Richard Crouch, Steve Pulman, Manny Rayner, and Arnold Smith. 1992. CLARE: A contextual reasoning and cooperative response framework for the Core Language Engine. Technical report, SRI International, Cambridge Computer Research Centre. Available as cmp-lg/9411002 from http://xxx.lanl.gov/cmp-lg.

Alshawi, Hiyan and Richard Crouch. 1992. Monotonic semantic interpretation. In *30th Annual Meeting of the Association for Computational Linguistics*, pages 32–39, Newark, Delaware.

Aust, H., M. Oerder, F. Seide, and V. Steinbiss. 1995. The Philips automatic train timetable information system. *Speech Communication*, 17:249–262.

Block, Hans Ulrich. 1994. Compiling trace & unification grammar. In Tomek Strzalkowski, editor, *Reversible Grammar in Natural Language Processing*. Kluwer Academic Publishers, Dordrecht, pages 155–174.

Bod, Rens and Remko Scha. 1997. Data-oriented language processing: An overview. Technical Report 38, NWO Priority Programme Language and Speech Technology.

Bonnema, Remko, Rens Bod, and Remko Scha. 1997. A DOP model for semantic interpretation. In *35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, Madrid.

Bonnema, Remko, Gertjan van Noord, and Gert Veldhuizen van Zanten. 1998. Evaluation results NLP components OVIS2. Technical Report 57, NWO Priority Programme Language and Speech Technology.

Boros, M., W. Eckert, F. Gallwitz, G. Görz, G. Hanrieder, and H. Niemann. 1996. Towards understanding spontaneous speech: Word accuracy vs. concept accuracy. In *Proceedings of the Fourth International Conference on Spoken Language Processing (ICSLP 96)*, Philadelphia.

Bouma, Gosse. 1992. Feature structures and nonmonotonicity. *Computational Linguistics*, 18(2).

Briscoe, Ted and John Carroll. 1993. Generalised probabilistic lr parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, 19(1):25–60.

Briscoe, Ted, Claire Grover, Bran Boguraev, and John Carroll. 1987. A formalism and environment for the development of a large grammar of english. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, pages 703–708, Milan.

Carpenter, Bob. 1992. Skeptical and creduluous default unification with applications to templates and inheritance. In Ted Briscoe, Anne Copestake, and Valerie de Paiva, editors, *Default Inheritance within Unification-Based Approaches to the Lexicon*. Cambridge University Press, Cambridge.

Carroll, John. 1993. *Practical unification-based parsing of natural language*. Ph.D. thesis, Cambridge University.

Cooper, Robin, Richard Crouch, Jan van Eijck, Chris Fox, Josef van Genabith, Jan Jaspars, Hans Kamp, Manfred Pinkal, Massimo Poesio, and Stephen Pulman. 1994. Fracas deliverable d8: Describing the approaches. Technical report, Centre For Cognitive Science, Edinburgh. Available by ftp from ftp.cogsci.ed.ac.uk, pub/FRACAS/.

Cormen, Leiserson, and Rivest. 1990. *Introduction to Algorithms*. MIT Press, Cambridge Mass.

Davidson, Donald. 1967. The logical form of action sentences. In Nicholas Rescher, editor, *The Logic of Decision and Action*. University of Pittsburgh Press.

Dijkstra, E.W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271.

Frank, Annette. 1994. Verb second by lexical rule or by underspecification. Technical report, Institute for Computational Linguistics, Stuttgart.

Gazdar, Gerald, Ewan Klein, Geoffrey Pullum, and Ivan Sag. 1985. *Generalized Phrase Structure Grammar*. Blackwell.

Hukari, Thomas E. and Robert D. Levine. 1995. Adjunct extraction. *Journal of Linguistics*, 31(2):195–226.

Jackson, E., D. Appelt, J. Bear, R. Moore, and A. Podlozny. 1991. A template matcher for robust NL interpretation. In *Speech and Natural Language Workshop*, pages 190–194, Pacific Grove, California, February.

Johnson, Mark and Jochen Dörre. 1995. Memoization of coroutined constraints. In *33th Annual Meeting of the Association for Computational Linguistics*, pages 100–107, Boston.

Koeling, Rob. 1997. Moving on the dialogue game board. In *Second Tbilisi Simposium Language, Logic and Computation*, Tbilisi State University.

Mellish, C.S. 1988. Implementing systemic classification by unification. *Computational Linguistics*, 14(1):40–51.

Moore, R., F. Pereira, and H. Murveit. 1989. Integrating speech and natural-language processing. In *Speech and Natural Language Workshop*, pages 243–247, Philadelphia, Pennsylvania, February.

Nederhof, Mark-Jan, Gosse Bouma, Rob Koeling, and Gertjan van Noord. 1997. Grammatical analysis in the ovis spoken-dialogue system. In *Proceedings of the ACL/EACL Workshop on Spoken Dialog Systems*, pages 66–73, Madrid, Spain.

Nerbonne, John. 1992. Constraint-based semantics. In P. Dekker and M. Stokhof, editors, *Proceedings of the Eight Amsterdam Colloquium*, pages 425–44, ITLI Amsterdam.

Netter, Klaus. 1992. On non-head non-movement. In G. Görz, editor, *KONVENS 92*. Springer-Verlag.

Nilsson, Nils. 1971. *Problem Solving Methods in Artificial Intelligence*. McGraw-Hill.

van Noord, Gertjan. 1995. The intersection of finite state automata and definite clause grammars. In *33th Annual Meeting of the Association for Computational Linguistics*, pages 159–165, MIT Cambridge Mass. Available from http://www.let.rug.nl/~vannoord/papers/.

van Noord, Gertjan. 1997a. An efficient implementation of the head corner parser. *Computational Linguistics*, 23(3):425–456.

van Noord, Gertjan. 1997b. Evaluation of OVIS2 NLP components. Technical Report 46, NWO Priority Programme Language and Speech Technology.

van Noord, Gertjan and Gosse Bouma. 1994. Adjuncts and the processing of lexical rules. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING)*, pages 250–256, Kyoto. Available from http://www.let.rug.nl/~vannoord/papers/.

van Noord, Gertjan and Gosse Bouma. 1997a. Dutch verb clustering without verb clusters. In Patrick Blackburn and Maarten de Rijke, editors, *Specifying Syntactic Structures*. CSLI Publications / Folli, Stanford, pages 123–153.

van Noord, Gertjan and Gosse Bouma. 1997b. Hdrug, a flexible and extendible development environment for natural language processing. In *Proceedings of the EACL/ACL workshop on Environments for Grammar Development, Madrid*.

Oerder, Martin and Hermann Ney. 1993. Word graphs: An efficient interface between continuous-speech recognition and language understanding. In *ICASSP Volume 2*, pages 119–122.

Pereira, Fernando C.N. and David Warren. 1980. Definite clause grammars for language analysis — a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13.

Pinkal, Manfred. 1995. *Logic and Lexicon*. Kluwer.

Pollard, Carl and Ivan Sag. 1994. *Head-driven Phrase Structure Grammar*. University of Chicago / CSLI.

Pulman, Steve. 1996. Unification encodings of grammatical notations. *Computational Linguistics*, 22(3):295–328.

Sag, Ivan. 1997. English relative clause constructions. *Journal of Linguistics*. to appear.

Scha, Remko. 1990. Taaltheorie en taaltechnologie; competence en performance. In *Computertoepassingen in de Neerlandistiek*. Landelijke Vereniging van Neerlandici (LVVN Jaarboek), Almere.

Veldhuijzen van Zanten, Gert. 1996. Semantics of update expressions. Technical Report 24, NWO Priority Programme Language and Speech Technology. http://odur.let.rug.nl:4321/.

Ward, W. 1989. Understanding spontaneous speech. In *Speech and Natural Language Workshop*, pages 137–141, Philadelphia, Pennsylvania, February.