Quantification and Scoping: Representing scope-ambiguity by means of underspecification

Rob Koeling
koeling@let.rug.nl

Abstract

In this paper we look at a proposal by Nerbonne (1993) to deal with the notorious scope-ambiguity problem by means of assigning semantic representations to sentences that are underspecified with respect to the scope of their quantificational expressions. First, we compare this proposal with other approaches that are suggested to solve this problem. Second, we describe an implementation of this proposal in Carpenter's Attribute Language Engine (ALE, (1994)). In this implementation we look at some classic problematic examples, like scope-islands caused by e.g. relative clauses, intensional verbs and control verbs. Finally, we look at possibilities to explore this idea in a wider range of ambiguity problems.

1 Introduction

Disambiguation is a classic problem and turns out to be one of the more challenging puzzles in the field of natural language processing. Ambiguities arise in many situations, and information from various sources must be considered to decide what is meant in a particular situation. Assigning different scopings to various quantificational expressions in a sentence is an instance of this problem. It gives rise to multiple readings for sentences that are assigned only one syntactic structure, turns out to be a major source of ambiguity itself, and illustrates most of the difficult problems which arise in disambiguation.

1.1 Scope ambiguity

Scope ambiguity crops up in many situations. We do not intend to give here a complete overview of situations in which scope ambiguity arises (for a good overview the reader is referred to (Carpenter, 1993)), we just illustrate the problem here with some examples. Later in this paper when we describe the implementation some more cases are discussed.

(1) a. A student read each book

b. An expert in every field read most books

The most obvious case is the subject/object scope-ambiguity in (1a). This sentence can either mean that there exists a student who has read every book (available in that specific context) or that for every book a student can be found that has read that particular book. These two readings can be expressed in the first order predicate-logic formulae (2a) and (2b) respectively. These formulae also illustrate that the ambiguity is caused by assigning different scopes to the quantifiers associated with the determiners *each* and *a*.

(2) a. $\exists x\ (student(x) \wedge \forall y\ (book(y) \rightarrow read(x,y)))$

b. $\forall y\ (book(y) \rightarrow \exists x\ (student(x) \wedge read(x,y)))$

Example (1b) contains three quantifiers. A naive algorithm for generating all the possible quantifier-scopings would generate six (all permutations of the three quantifiers) readings (see (Hobbs and Shieber, 1987) for discussion). This illustrates two aspects of the problem. First, the exponential growth of the number of readings with the number of quantifiers in the sentence. Computation of all the possibilities can become rather expensive. The second aspect that can be mentioned, is that not all the readings generated by the naive algorithm are legitimate. We will come back later to that issue.

In most situations it is clear for the participants of a dialog which of the possible readings is meant by the one who uttered the ambiguous sentence. Although the issue of scope determination is still a difficult and open research question, it is clear that it is not purely determined by the syntactic structure of a sentence (Vanlehn, 1978). The choice of the preferred reading is influenced by lots of factors (syntactic, lexical and pragmatic). It might be possible that at a given moment in a conversation the information that is needed to choose the right reading is already available, but also that this information will be added at a later stage. In both cases all the possible readings might be considered. The extra (context-)information constrains the number of possibilities.

A theory for processing natural language needs a component that can cope with this problem. When a sentence is processed, these kinds of systems must first be able to recognise the various possible readings. Secondly, they need some way to decide whether a reading is compatible with the context-information or not. And finally, in case there are more potential readings, there must be a possibility of choosing a preferred reading (e.g. on linguistic or pragmatic grounds).

1.2 Approaches to quantifier scoping in natural language processing

Many proposals regarding the characterisation of scope ambiguities have been made. An important aspect of a proposal is naturally the ability of predicting correctly all the possible scopings (completeness) without generating incorrect readings (soundness). This goal is the same for every approach and it is possible (and useful) to classify the proposals in terms of success. A second way of classifying the approaches is looking at the strategy they follow. We distinguish three strategies:

- Multiple representations (one representation for each reading): e.g. Montague's PTQ (Gamut, 1991), Cooper Storage (Cooper, 1983; Keller, 1988), the deductive approach of Pereira (1990) or Carpenter (1993)
- Single (unscoped) representation for all the possible readings:
  - Special representational level (in algorithmic relation to representation of each reading): e.g. (Hobbs and Shieber, 1987; Gerdemann and Hinrichs, 1990), the Core Language Engine (Alshawi, 1992)
  - Underspecified representation on the same representational level as the (fully scoped) representations of the different readings (in subsumption relation to representation of each reading): e.g. (Nerbonne, 1993; Reyle, 1993)

The reason to make such a classification is the computational attractivity of the algorithm. As we have pointed out, the number of readings grows quickly with the number of quantifiers involved in the sentence. It might therefore be attractive to be able to postpone the computation of all the possible readings until the moment that they are really needed. In this paper we have a closer look at the last strategy. This is done by explaining Nerbonne's proposal in the next section and by describing an implementation of this approach in Section 3.

2 A feature based syntax-semantics interface

In the paper *A Feature-Based Syntax/Semantics Interface* (1993), Nerbonne argues that in unification-based approaches of natural language processing syntactic processing not only harmonises well with the way in which syntax is normally described, but that it further provides the possibility of coupling syntactic and semantic processing as tightly as one wishes. Combining syntax and semantics in feature-based formalisms has become popular practice in the field of computational linguistics. Well known examples are described in (Shieber, 1986) and the HPSG books by Pollard and Sag (1987; 1994).

The rules for combining constituents must, apart from constraints on syntax, also include constraints on how the semantics of those parts may be combined. This way of integrating syntax and semantics provides the major advantage that not only syntactic constraints on semantics are allowed, but that also other information, e.g. context-information, can be employed to compute the ultimate sentence-meaning. A simple example of how this works is given in Figure 1:

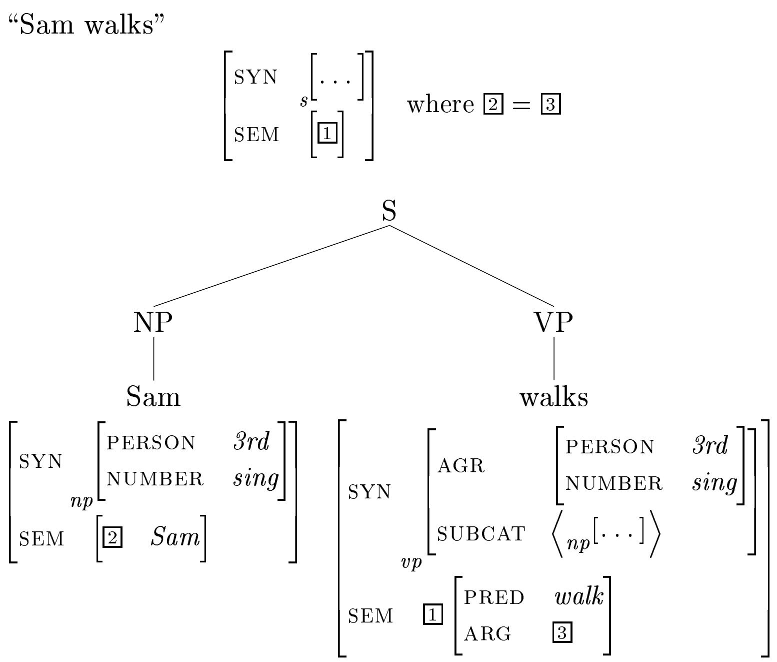


Figure 1: Combining syntax and semantics in a feature structure.

Here, feature structures are employed to describe the logical form of the meaning of the sentence. Nerbonne argues that this way of describing semantics provides the possibility of defining constraints on logical form as well as the opportunity of underspecifying meanings "in ways difficult to reconcile with non-constraint-based accounts". He favours the view of using an intermediate level of logical form for describing semantics above the simpler view in which representations described by feature structures may be interpreted as directly denoting elements within a semantic model.

The feature-based approach provides us with much flexibility regarding the choice of a semantic representation language. Feature structures, as a formalised metalanguage for a semantic representation logic, allow us to describe any logic we want. Here we have chosen to adopt the language of generalised quantifiers (LGQ). This well-known and well-understood language has become popular in the field of computational linguistics.

2.1 Scope Underspecification

The expressive power of the syntax/semantics-interface proposed by Nerbonne, is illustrated by a characterisation of the classical scope ambiguity problem. He claims that semantics conceived along these lines provides the possibility of characterising scope-ambiguities completely within the formalism. So, it is not necessary to make an appeal to auxiliary notions like *Quantifier Store* (Cooper, 1983; Keller, 1988) or *Quasi Logical Form* (Alshawi, 1992). A second aspect that is worth noting here, is that this framework provides a good basis for postponing the computation of all scoping-possibilities. Considering the fact that a syntax/semantics-interface provides the possibility of using information from other sources, which may constrain the number of readings of a sentence, the scoping-problem may be resolved without putting such a heavy computational burden on the formalism.

Even when you want to characterise the different quantifier scopings by means of underspecification, you still need the information that describes the relation between the quantifiers involved and their scope. The challenge now is to express it in a more general way, without losing correct or adding incorrect readings. In the remainder of this section we will give the idea of how we can represent scope-ambiguities in this framework. In the section on the implementation we will explain in more detail how to put constraints on these feature structures to ensure that all and only the right solutions are described by this representation. The first question we need to answer is what is meant with *scope*. Nerbonne's observation with respect to the representation of the quantifier scopings in CLE's Quasi Logical Form is that they do not try to describe that a particular quantified wff has a particular scope, but that *either* a particular quantified wff has a particular scope *or* that its scope has a particular scope, *or* that its scope's scope has a particular scope, etc. Let us call this piece of information the *nuclear scope*. We know that this scope occurs somewhere in the formula, but we do not want to say something about where it is found at this moment. Now we need to express this nuclear scope, as general as possible. This is done by adding the feature SCOPE*. The relation between the nuclear scope and the ultimate sentence-scope (that is given in words above), can easily be expressed in the feature description language (see (3)).

(3) $$\begin{bmatrix} \text{QUANT} & & \\ \text{SCOPE} & \boxed{1} & \begin{bmatrix} \text{SCOPE}^* & \boxed{2} \end{bmatrix} \\ \text{SCOPE}^* & \boxed{3} & \end{bmatrix}_{q\_wff} \quad \text{Condition: } \boxed{3} = \boxed{1} \vee \boxed{3} = \boxed{2}$$

The scope is only half the information we need to represent the possible scopings. The other required information is, of course, the quantifiers that are involved. We have just defined SCOPE* as the nuclear scope of a complex formula. This feature can be seen as denoting a path. The quantifiers involved can be encountered somewhere on the path that SCOPE* denotes. To keep it as general as needed here, we simply have to represent these quantifiers without saying anything about the order. Therefore once again an extra feature is introduced: QUANT*. QUANT* is a feature that contains the *set of quantifiers* that are encountered along the path denoted by SCOPE*.The relation between the features QUANT and QUANT* is similar to the one we defined for SCOPE* and is given in (4).

(4) $$\begin{bmatrix} \text{QUANT} & \boxed{1} \\ \text{SCOPE} & \begin{bmatrix} \text{QUANT*} & \boxed{2} \end{bmatrix} \\ \text{QUANT*} & \boxed{3} \end{bmatrix}_{\textit{q\_wff}} \quad \text{Condition: } \boxed{3} = \left\{ * \ \boxed{1} \ * \right\} \cup \boxed{2}$$

Nerbonne claims that with these two extra features a characterisation of scope-ambiguity (of simple sentences like (1a))[1] can be given. That the desired subsumption relation holds can easily be illustrated by (5–7).

(5) $$\begin{bmatrix} \text{QUANT} & \\ \text{QUANT*} & \left\{ * \ \forall x \ student(x) \ , \ \exists y \ book(y) \ * \right\} \\ \text{SCOPE} & \\ \text{SCOPE*} & \textit{read(x,y)} \end{bmatrix}_{\textit{q\_wff}}$$

subsumes:

(6) $$\begin{bmatrix} \text{QUANT} & \exists x \ student(x) \\ \text{QUANT*} & \left\{ * \ \exists x \ student(x) \ , \ \forall y \ book(y) \ * \right\} \\ \text{SCOPE} & \begin{bmatrix} \text{QUANT} & \forall y \ book(y) \\ \text{QUANT*} & \left\{ * \ \forall y \ book(y) \ * \right\} \\ \text{SCOPE} & \textit{read(x,y)} \\ \text{SCOPE*} & \textit{read(x,y)} \end{bmatrix}_{\textit{q\_wff}} \\ \text{SCOPE*} & \textit{read(x,y)} \end{bmatrix}_{\textit{q\_wff}}$$

and:

[1]For the (slightly) more difficult case of (1b) we must account for the fact that the restriction of a generalised quantifier might be a quantified wff. This means that we have to introduce the feature REST* and define the relation between REST and REST*. This relation is defined in Section 3.5. It also complicates the relation between QUANT, SCOPE and QUANT* and we will have to redefine this relation. For details the reader is refered to (Nerbonne, 1993) or (Koeling, 1994).

$$(7)\qquad {}_{q\_wff}\left[\begin{array}{ll} \text{QUANT} & \forall y\ book(y) \\ \text{QUANT}^* & \left\{^*\ \exists x\ student(x)\ ,\ \forall y\ book(y)\ ^*\right\} \\ \text{SCOPE} & {}_{q\_wff}\left[\begin{array}{ll} \text{QUANT} & \exists x\ student(x) \\ \text{QUANT}^* & \left\{^*\ \exists x\ student(x)\ ^*\right\} \\ \text{SCOPE} & read(x,y) \\ \text{SCOPE}^* & read(x,y) \end{array}\right] \\ \text{SCOPE}^* & read(x,y) \end{array}\right]$$

He shows (1993, pp. 124–125), that *all* and *only* the possible scopings are captured by this representation with underspecified scope.

3 The implementation

Before we can describe the implementation of the grammar based on the ideas we presented in Section 2, we have to say something about the grammar development environment we used. We demand a certain expressive power to be able to express what we need. The grammar is described by presenting the four main components of the grammar in some detail. These are the definition of the *type-hierarchy*, the *phrase structure rules*, the *lexical entries* and the *definite clauses*. We explain the most important part of these components (for a full explanation the reader is refered to (Koeling, 1994)) and we demonstrate how particular scope-phenomena are handled.

3.1 Ale

We need a formalism in which we can express constraints on the terms we have defined in feature structures. This yields problems in most feature-based formalisms. Recursion for example, is often only allowed in the characterization of the grammar-rules and not, as we need, in the description of the feature structures itself. This can be illustrated by the impossibility of expressing the constraints that we need (like the one in Figure 3) in a system like PATTR-II (Shieber, 1986). HPSG, however, has adopted a richer view of employing feature structures. It allows us to express the required constraints, as well as to put restrictions on the appropriateness of attributes and values by means of a type-hierarchy. This richer view provides further the possibility of defining *relations* between terms and making reference to *sets*.

The *Attribute Language Engine* (ALE) is a grammar development environment in which phrase structure parsing and definite clause programming are integrated. It supports the richer view on feature structures that we just described. This means in particular that grammars written in ALE are *strongly typed* (i.e. every feature structure is obligatory associated with a type). It further means that it is possible to define relations between terms by definite clauses. Definite clauses can be associated with phrase structure rules, which gives us great flexibility in manipulating feature structures. The way feature structures are employed for defining and parsing phrase structure rules is (apart from typing) fairly similar to for example PATR-II. Most of the material we used here can be found in the ALE users guide (1994) and more theoretical details in (Carpenter, 1992).

3.2 The type-system

We have mentioned before that HPSG provides the right environment to give a characterization of Nerbonne's syntax/semantics-interface. Although the semantics for this approach are completely different from those developed for HPSG, we gratefully made use of the grammar of HPSG that Bob Carpenter has coded in ALE. We have focussed on the semantic part of the grammar and therefore we kept the syntactic part as simple as possible while giving a correct account of the phenomena we wanted to describe by means of a small fragment of English. This resulted in some modifications in the syntactic part of the HPSG-grammar. In Figure 2 (a part of) the type hierarchy is given for the semantic part of the grammar. It is a straightforward implementation of a part of the Language of Generalized Quantifiers. For every type the features are specified so that for any type we see which features are appropriate (notation: *feature : type*). It should be clear to the reader that this hierarchy can easily be extended in many ways.

There are a few things to say about this definition. First, we want to note the reliance on type inheritance in the definition of RELATION and WFF. Second, the existence of type IND_EXPR was motivated by the fact that we needed the feature INDEX at the level of WFF (we will explain why in Section 3.3). We had to introduce an index anyway for the type TERM. One may not introduce the same feature twice, so we needed a *most general unifier* of TERM and WFF. It was of course possible to introduce this feature at the level of EXPR, but that would result in an obsolete feature INDEX on GEN_QUANT, RELATION, etc. Finally, FREEVARS and QUANT* are defined as set bearing features. ALE, however, does not provide the possibility of using sets. Although we have defined the type *set*, sets are simulated by lists and the definition is thus just a copy of the list-definition. A set consists therefore (as in Prolog) of a head element (*elt*) and a rest-set of elements (*elts*).

3.3 Rules

First we describe the phrase structure rules that are necessary to give an account of the theory described in Section 2. Later we will also show how this theory is exploited to characterise other quantification phenomena. We

```
EXPR
    IND_EXPR
      index : name
        TERM
          VAR
            index : var_name
          CONST
            index : const_name
        WFF
          free_vars : set_var
            A_WFF
              relation : relation
            Q_WFF
              quant : gen_quant
              scope : wff
              quant* : set_quant
              scope* : wff
    GEN_QUANT
      qdet : qdet
      qvar : var_name
      q_freevars : set_var
      rest : wff
      rest* : wff
    RELATION
        UN_RELATION
          first : term
            STUDENT
            ...
        BIN_RELATION
          second : term
            READ
            ...
    QDET
        FORALL
        EXISTS
```

Figure 2: The type-hierarchy for the semantic part of the grammar. Under the types are given the features that are introduced at that type.

have kept the number of grammar rules as small as possible. It is not our intention to cover the largest possible fragment of English; we just want to demonstrate how quantification phenomena are represented in this framework.

We will focus on those parts of the grammar rules that are of primary interest for our problem. That is why we do not say anything about syntactic phenomena other than subcategorization. Our main interest here is the interaction between the features QUANT*, SCOPE* and REST*: how these features receive values and how their contents flow in the derivation tree.

Most of the work presented here is in the spirit of HPSG, but we deviate from this framework in the definition of the rules in the implementation. Even though the lexical entries bear full subcategorization information, we have spelled out the phrase structure rules completely in our grammar. As a consequence of this, the grammar contains redundant information. Of course, we did not design for redundancy, but it evolved while developing the grammar. It is still possible to adapt the grammar in such a way that we can rely on HPSG's principles without spelling out all the phrase structure rules. Most rules we present in this section, for example, are instances of the single HPSG-rule:

(8) $\text{X} \rightarrow \text{H C}^*$

And also the principles we present later in this section for passing up the semantic contents of constituents in the analysis tree do not contradict HPSG. Even though they are listed on every single rule and are not represented by a principle that must be obeyed by the grammar.

The Quantifier Principle

We need some principle for distributing the contents of the features QUANT* and SCOPE*. Recalling that we have defined the feature QUANT* as bearing the quantifiers that occur on the path denoted by SCOPE* (containing the nuclear scope of the formula), we let the head daughter provide the scope-information and we collect all the quantifiers involved in the formula. This is formulated in the principle in Figure 3. All the phrase structure rules that we have defined in our grammar are instances of this general rule. Consider the rule for the last step to form a sentence in Figure 4. Although not all the information is given in this figure, most aspects of the grammar we want to mention here are shown. First, it is clear that this rule follows the quantifier principle which we have just defined. The first goal takes care of collecting the quantifiers by joining the sets of quantifiers of the daughter nodes. This rule shows further that during the derivation the feature structures remain underspecified with respect to scope. We can specify the scope at any time we want by calling the goal `solve_underspec/5`. For this grammar we have chosen to do that at sentence level. In Section 3.5 we give details of how

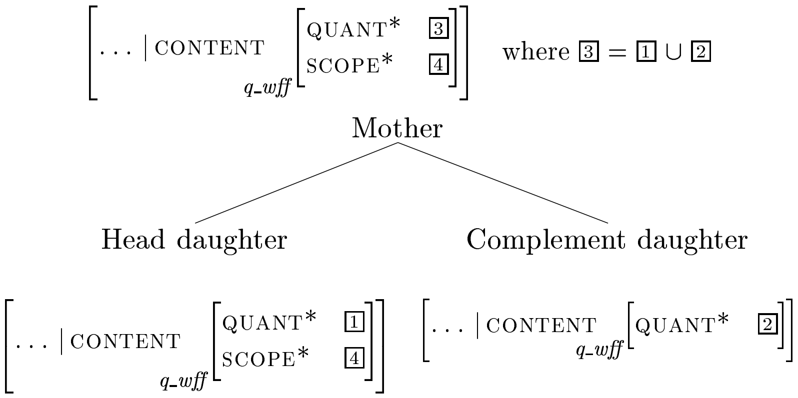
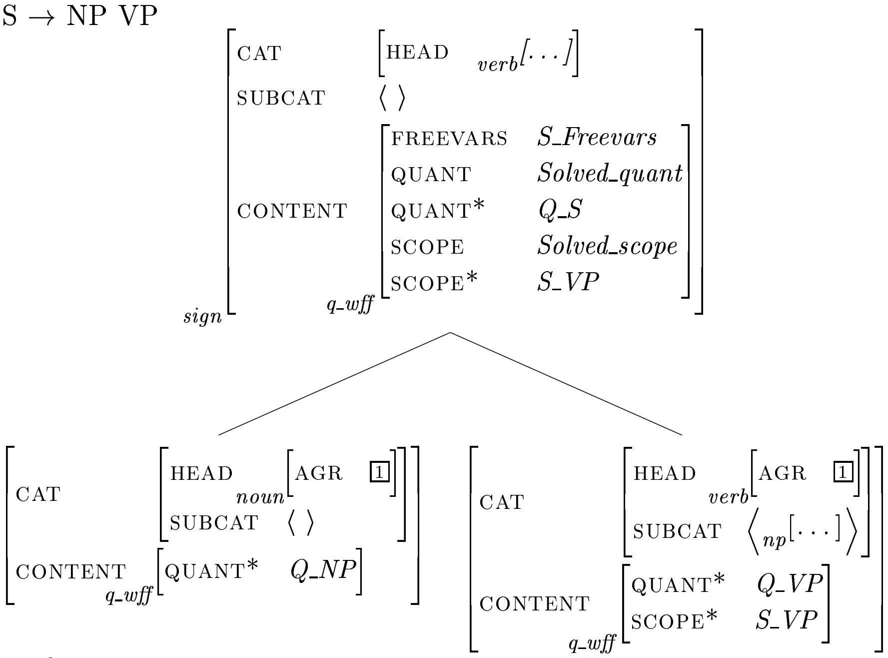


Figure 3: The Quantifier Principle

S $\rightarrow$ NP VP



goal:

```
( union(Q_NP, Q_VP, Q_S),
  solve_underspec(
      Q_S, S_VP, Solved_quant, Solved_scope,
      (S_Freevars, e_set)
  )
)
```

Figure 4: Sentence formation

this goal is evaluated. The feature FREEVARS remains also underspecified during parsing. This feature, which is used as a constraint on the well-formedness of formulae, is instantiated during the evaluation of the goal `solve_underspec/5`. It is associated with the *empty set* at sentence level to exclude open formulae. The rule for constructing the semantic contents of a NP is given in Figure 5. Note that this rule follows the earlier formulated principle for distributing the contents of the QUANT* and the SCOPE* feature. The contents of the determiner consists of a generalized quantifier whose variable is restricted by the contents of the NP. This is done by assigning the value of SCOPE* of the Nbar to the REST* of the determiner. It might be possible that the Nbar is complex. In that case the QUANT* of the Nbar will not be empty and those quantifiers are added to the QUANT* of the NP. A consequence of this is that the quantifiers inside the Nbar are not restricted to the restriction of this quantifier, but that they are also allowed to take wide scope over this noun phrase.

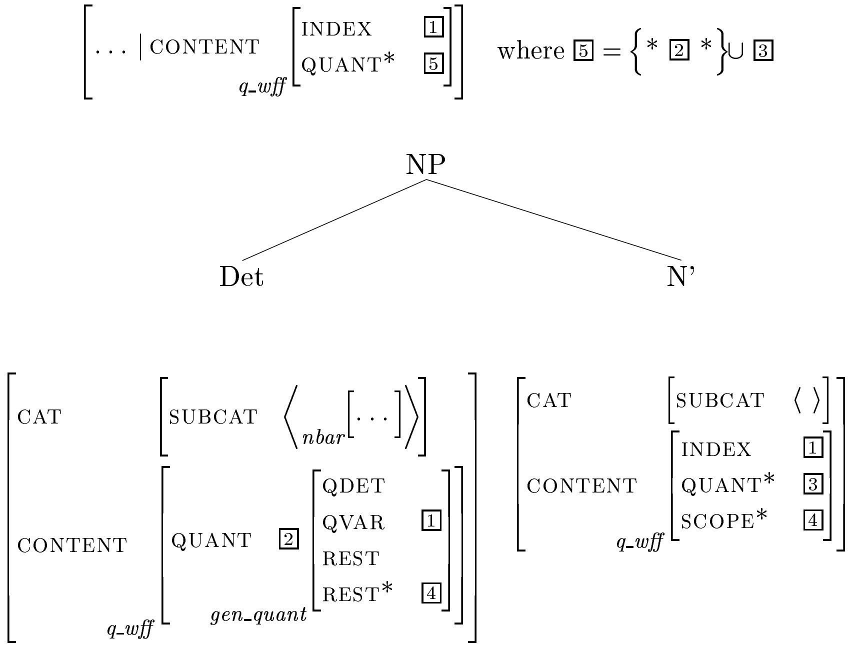


Figure 5: The construction of the semantic contents of a noun phrase

If the Nbar is complex, the feature INDEX will bear the value of the argument position of SCOPE* that must be bound by the quantifier of the determiner the Nbar combines with in this rule. The INDEX of the resulting NP will get the same value. The QUANT* of the NP contains two quantifiers in this case. When the NP combines with a VP, the free argument position of the VP must be bound by the variable associated with the last encountered determiner. This information would normally be hidden somewhere in the (set-based) feature QUANT*, but is now accessible through INDEX.

The subject of sentence (9a) is a complex NP containing two quantifiers. It must be possible for the quantifier inside a prepositional phrase to take wide scope over the whole NP. This is required by the general principle that quantifiers may scope freely (in head-complement structures) and it is implemented by the Det/Nbar rule shown in Figure 5. In this rule, the quantifiers that occur within the Nbar contribute to the NP in the same manner as the quantifier that is introduced by the determiner. One of the six readings of (9b) is successfully ruled out by the free-variable constraint.

(9) a. An expert in every field attended the meeting

b. An expert in every field read most books

Relative Clauses

Unlike the quantifiers that occur within a prepositional phrase, quantifiers inside relative clauses are (generally[2]) not allowed to take wide scope over the NP.

(10) An ice-cream that everyone likes is sold out

So we need some way to prevent the quantifier inside the relative clause to take wide scope over the NP. As we have seen in the characterization of the NP-rule, quantifiers that occur within the QUANT* of the Nbar are allowed to take wide scope (which is normally desired in case of complex noun phrases). Figure 6 shows how the semantic contents of the relative clause contribute to the contents of the Nbar. The contents of the QUANT* of the relative clause is not joined with the QUANT* of the N', but forms together with the SCOPE* of the relative clause a local underspecified WFF.

This might at first blush seem to contradict the general quantifier principle that we have formulated before. But we formulated that principle for sentences of the head/complement-type. In HPSG, however, relative clauses are not considered to be complement-daughters but adjunct-daughters, which means that this rule does not have to follow this principle. Stating it this way, the universal quantifier of the relative clause contributes to the restriction of the quantified noun phrase, without being able to take wide scope over the noun phrase.

3.4 Lexical entries

In this section we will not say much about what the lexical entries in general look like. We will jump directly to the treatment of some special cases. For a description of the other lexical entries the reader is refered to (Koeling, 1994).

[2]Pereira (1990) presents some examples that contradict this.

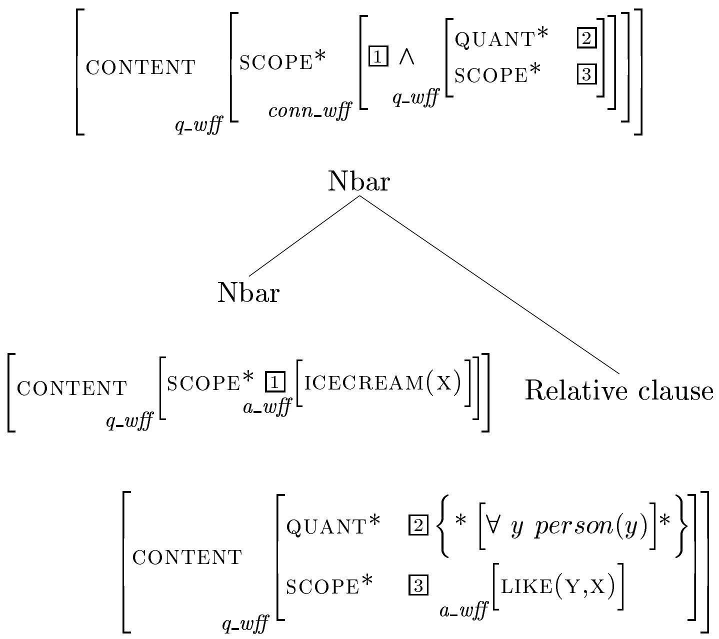


Figure 6: Restricting the scope of a quantifier inside a relative clause

Intensional verbs

Intensional verbs introduce a so-called *de dicto/de re* ambiguity. Sentences like (11)a can be read in two ways. The *de re* reading implies the existence of a unicorn, while the *de dicto* reading does not. We treat intensional verbs in the Montagovian style (Gamut 1991 p. 185). That is, the *de dicto* reading is obtained by allowing the object position of *seek* to take a generalized quantifier as argument. The two readings are then translated in the following formulae:

(11) a. John seeks a unicorn.

b. $\exists\ (x)\ (unicorn(x) \wedge seeks(John, x)\ )$

c. $seeks(John, \exists\ (x)\ unicorn(x))$

Since we have defined arguments (of type TERM) to be *constants* or *variables*, we need to modify the type-hierarchy slightly to be able to obtain the second reading. That is, we need to allow some relations to take either *terms* or *gen_quants* as arguments. The verb *seeks* subcategorizes for two NPs:

(12) *seek*:

$$\left[\text{SUBCAT}\ \left\langle {}_{np}\begin{bmatrix}\text{INDEX} & j\\ \ldots & \end{bmatrix}, {}_{np}\begin{bmatrix}\text{INDEX} & \boxed{1}\ x\\ \text{QUANT*} & \left\{*\ \boxed{2}\ *\right\}\end{bmatrix}\right\rangle\right]$$

The *de dicto/de re* ambiguity is characterized by allowing the second argument position to take values of both types.

(13) a. de re:

$$\left[\text{CONTENT} \quad \left[\begin{array}{ll} \text{QUANT*} & \left\{ * \ \exists x \ unicorn(x) \ * \right\} \\ \text{SCOPE*} & \left[\text{RELATION} \quad \left[\begin{array}{ll} \text{FIRST} & John \\ \text{SECOND} & \boxed{1} \ x \end{array}\right]_{seek} \right]_{a\_wff} \end{array}\right]_{q\_wff} \right]$$

b. de dicto:

$$\left[\text{CONTENT} \quad \left[\begin{array}{ll} \text{QUANT*} & \left\{ * \quad * \right\} \\ \text{SCOPE*} & \left[\text{RELATION} \quad \left[\begin{array}{ll} \text{FIRST} & John \\ \text{SECOND} & \boxed{2} \left[\exists x \ unicorn(x)\right]_{gen\_quant} \end{array}\right]_{seek} \right]_{a\_wff} \end{array}\right]_{q\_wff} \right]$$

Control verbs

(14) a. John believed everyone to be quiet.
$\forall x(person(x) \rightarrow believe(John, quiet(x)))$
$believe(John, \forall x(person(x) \rightarrow quiet(x)))$

b. John persuaded everyone to be quiet.
$\forall x(person(x) \rightarrow persuade(John, x, quiet(x)))$

Raising-control verbs (like *believe* in (14a)) allow quantified formulae as arguments.

- TERM
  - S_TERM
    - VAR
      - index : var_name
    - CONST
      - index : const_name
  - C_TERM
    - form : wff

Figure 7: The revised definition of the type TERM.

This may be contrasted with equi-control verbs (like *persuade* in (14b)) whose second argument position is restricted to only (simple) terms. This means that a quantifier that binds a variable at the second argument position must always have wide scope over the expression. To obtain the second reading of the raising-case, we need to extend the type-hierachy to allow terms also to be well-formed formulae instead of just simple terms (Figure 7). The different behavior of these control verbs can be characterized lexically. We can account for the ambiguity that crops up in the raising-case by allowing the object-position to take arguments of type *a_wff* as well as arguments of type *q_wff*. The lexical entry for a raising-verb will then be:

(15) *believe*:

$$\left[\begin{array}{ll} \ldots|\text{SUBCAT} & \left\langle \ldots, {}_{fin\_S}\left[\text{CONTENT} \quad {}_{wff}\left[\text{SCOPE*} \quad \boxed{1}\right]\right]\right\rangle \\ \ldots|\text{CONTENT} & {}_{a\_wff}\left[\text{RELATION} \quad {}_{believe}\left[\begin{array}{ll}\text{FIRST} & John \\ \text{SECOND} & {}_{wff}\left[\text{SCOPE*} \quad \boxed{1}\right]\end{array}\right]\right]\end{array}\right]$$

This means however that we need to extend the type-hierarchy to allow *atomic wff's* to bear a value for SCOPE*. The information that SCOPE* bears is in case of atomic wff's equal to the information the atomic wff bears itself. This is characterized by the constraint on the value of SCOPE* in atomic wff's, as in (16) below:

(16)

$$\boxed{1} \; {}_{a\_wff}\left[\begin{array}{ll}\text{RELATION} & \\ \text{SCOPE*} & \boxed{1}\end{array}\right]$$

The ambiguity is still characterized by means of underspecification. If the second argument is taken to be of type *q_wff* then the QUANT* feature remains unspecified. The value of QUANT* is determined by the constraint we have given in Section 2 (Figure 4). The fact that *persuade* is unambiguous can be enforced by restricting the type of the second argument to *simple terms.*

3.5 Definite Clauses

In our grammar, definite clauses are used in two ways. First, we need a tool for dealing with set-expressions. QUANT* is a set-based feature. In the section about the grammar-rules we have pointed out that the content of the QUANT*-feature of the mother-node is composed of those of the daughter-nodes. Union of sets is however not a standard operation in feature-based grammar formalisms. Recalling that sets are simulated by lists, the definite clause for this operation consists of a straightforward Prolog-goal for the concatenation of lists.

With the grammar as we have developed it now, we can give a characterization of scope-ambiguities. The various readings however are not computed. Normally this is the desired situation. Computation of all the

```
solve_underspec(Quant_star,Scope_star,Quant,Scope,Freevars) if
/*1*/   remove(Quant,Quant_star,Rest_quant_star),
/*2*/   gq_constraint(
            (Quant,@gen_quant(_Qdet,Qvar,Q_freevars,_Rest,_Rest*))
        ),
/*3*/   solve_scope(Rest_quant_star,Scope_star,Quant,Scope),
/*4*/   solve_rest_underspec(Quant),
/*5*/   solve_scope_underspec(Scope),
/*6*/   scope_fv(Scope,S_freevars),
/*7*/   collect_free_vars(S_freevars,Q_freevars,Qvar,Freevars).
```

Figure 8: 'Solving' underspecification in ALE

readings might be computationally expensive and the unscoped representation suffices for linguistic purposes. As we have pointed out in Section 2, the subsumption relation holds (which is in many cases[3] enough for deciding whether a reading is compatible with the currently available information about the situation) and is it possible to add information from various sources. But sooner or later, we might want the fully specified characterizations of the possible scopings. So we have to create the possibility of computing all the readings. We have defined definite constraints for this purpose.

The clause for 'solving' the underspecification is shown in Figure 8[4] . Here, `Quant_star` and `Scope_star` are the variables that are associated with the values of the features QUANT* and SCOPE* respectively. These features bear the information of the quantifiers and the scope involved in the formula. `Quant` and `Scope` will get a value when this clause is evaluated. The object of the feature FREEVARS is to constrain the number of readings by excluding ill-formed formulae (i.e. open formulae and formulae in which vacuous binding occur). At sentence level no more free variables are allowed (variables that are free at sentence level fall outside the scope of a quantifier), we have already seen that these formulae are excluded by associating FREEVARS with the empty set after evaluation of this clause.

In Clause 1, one quantifier is taken out of the set of quantifiers (QUANT*) that occur on the path denoted by SCOPE*, and is assigned to the feature `Quant`. The backtracking mechanism takes care of trying all the available quantifiers at this position. Clause 2 is the implementation of the constraint

[3]But *not* for example in case of negation for which additional reasoning is needed.

[4]We make intensive use of *templates* (called *macro's* in ALE) and *variables* in the characterization of the definite clauses. For example the clause `gq_constraint/1` takes a generalized quantifier as an argument. Therefore the template `@gen_quant/5` is used. This quantifier is just taken out of the set of quantifiers denoted by QUANT* (by clause 1 in Figure 8) and is associated with the variable `Quant` (which received its value on line 1).

that defines the relation between the features REST and REST*:

(17) $${}_{gen\_quant}\begin{bmatrix} \textsc{rest} & \boxed{1}\begin{bmatrix}\textsc{scope}^* & \boxed{2}\end{bmatrix} \\ \textsc{rest}^* & \boxed{3} \end{bmatrix} \quad \text{Condition: } \boxed{3} = \boxed{1} \vee \boxed{3} = \boxed{2}$$

Clause 3 takes care of the distribution of the contents of the QUANT* and the SCOPE*-feature. Therefore it has just to obey the constraints on this distribution that we described in Section 2.

(18)
```
solve_scope(Quant_star_4,Scope_star,Quant_1,Scope) if
          q_wff_quant_constraint(
               Quant_star_4,Scope_star,Quant_1,Scope),
          q_wff_scope_constraint(Scope_star,Scope).
```

The clauses 4 and 5 check whether the restriction and the scope of the quantifier involved at this level are complex or not. In case one of them is complex, the formula will be underspecified with respect to scope. So `solve_underspec` must be called recursively. This recursion will stop when no more quantifiers are available. Here we gratefully make use of templates. Particularly clause 5 makes clear that, in order to express the notion of *nuclear scope* (a quantifier has a particular scope, or its scope has a particular scope, or its scope scope etc.) we must be able to construct feature structure like in (19).

(19) $${}_{q\_wff}\begin{bmatrix} \textsc{index} & \ldots \\ \textsc{quant} & \ldots \\ \textsc{scope} & {}_{q\_wff}\begin{bmatrix} \textsc{index} & \ldots \\ \textsc{quant} & \ldots \\ \textsc{scope} & {}_{q\_wff}\begin{bmatrix} \ldots\ \textsc{etc.}\ \ldots \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The variable `Scope` in clause 5 is associated with a feature structure of type *wff*. When it bears a quantified wff (underspecified with respect to scope), `solve_underspecific/5` is called recursively. When the clause returns from its recursive loop, the unification of the value of the SCOPE feature with the original feature structure (which was not specified for scope), is performed by just associating the variable for `Scope` in the template with the returned value for the SCOPE feature.

To evaluate clause 4, the variable in the restriction that is bound by the variable of the generalized quantifier (`Qvar`) must be removed from the set of free variables. In order to be able to remove a variable from a list of variables we need the possibility to check the equality of variables (`==` in Prolog). The set of free variables is computed in the last clause (7). This set is made up by the union of the set of free variables that occur in the generalized quantifier and those from the scope, with the variable bound by the quantifier removed.

4 Conclusions

In this paper, we have described an implementation of an approach to the quantifier scope ambiguity problem in which it is possible to represent scope ambiguities by means of underspecification on the same representational level as the fully scoped representations of the different readings. We have shown that the framework that Nerbonne provides in (Nerbonne, 1993) can be implemented in a straightforward manner in a grammar development environment like ALE. We have added analysis of some more cases in which scope ambiguities occur. It might be interesting to investigate how other phenomena can be described in this framework, like interaction with other operators (e.g. negation or modals). Even more interesting seems to be to investigate how this work can be used in disambiguation. The fact that the underspecified representations are part of the formalism, provides good opportunities to postpone the computation of all readings and perorm further action on basis of the underspecified representations (e.g. similar to those described in (Reyle, 1995)).

References

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

Carpenter, Bob. 1992. *The Logic of Typed Feature Structures.* Cambridge University Press.

Carpenter, Bob. 1993. Quantification and scoping: A deductive account. unpublished ms., Carnegie Mellon University, Pittsburgh.

Carpenter, Bob. 1994. The attribute logic engine user guide. Technical report, Laboratory for Computational Linguistics, Carnegie Mellon University, Pittsburgh.

Cooper, Robin. 1983. *Quantification and Syntactic Theory.* Reidel. volume 21 of Synthese Language Library.

Gamut, L. T. F. 1991. *Logic, Language and Meaning, book II: Intensional Logic and Logical Grammar.* The University of Chicago Press.

Gerdemann, Dale and Erhard W. Hinrichs. 1990. A unification-based approach to quantifier scoping. In *Proceedings of the 9th European Conference on Artificial Intelligence*, pages 292–297.

Hobbs, Jerry and Stuart M. Shieber. 1987. An algorithm for generating quantifier scopings. *Computational Linguistics*, 13.

Keller, William R. 1988. Nested cooper storage: The proper treatment of quantification in ordinary noun phrases. In U. Reyle and C. Rohrer, editors, *Natural Language Parsing and Linguistic Theories*. Reidel, Dordrecht, pages 221–242.

Koeling, Rob. 1994. Quantification and scoping: Representing scope-ambiguities by means of underspecification. Master's thesis.

Nerbonne, John. 1993. A feature-based syntax/semantics interface. *Annals of Mathematics and Artificial Intelligence*, 8:107–132.

Pereira, Fernando C.N. 1990. Categorial semantics and scoping. *Computational Linguistics*, 16(1):1–10.

Pollard, Carl and Ivan Sag. 1987. *Information Based Syntax and Semantics, Volume 1*. Center for the Study of Language and Information Stanford.

Pollard, Carl and Ivan Sag. 1994. *Head-driven Phrase Structure Grammar*. Center for the Study of Language and Information Stanford.

Reyle, Uwe. 1993. Dealing with ambiguities by underspecification: Construction, representation and deduction. *Journal of Semantics*, 10:123–179.

Reyle, Uwe. 1995. On reasoning with ambiguities. In Kees van Deemter, editor, *Semantic Ambiguity and Underspecification*. CSLI Publications.

Shieber, Stuart M. 1986. *Introduction to Unification-Based Approaches to Grammar*. Center for the Study of Language and Information Stanford.

Vanlehn, K.A. 1978. Determining the scope of english quantifiers. Technical report, M.I.T. Cambridge. Master's thesis, Published as Report AI-TR-483.