PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1988,
edited by ICOT.

OVERVIEW OF THE CORE LANGUAGE ENGINE

H. Alshawi, D.M. Carter, J. van Eijck, R.C. Moore*, D.B. Moran*

SRI International
Cambridge Computer Science Research Centre,
23 Millers Yard, Cambridge CB2 1RQ, U.K.

S.G. Pulman

Cambridge University
Computer Laboratory,
Pembroke Street,
Cambridge CB2 3QG, U.K.

ABSTRACT

The Core Language Engine (CLE) is a domain independent system for translating natural language (English) sentences into formal representations of their literal meanings which are capable of supporting reasoning. It is designed to be used as a major component of interactive advisor systems such as interfaces to database management systems and diagnostic expert systems. The main contribution of the CLE is intended to be substantial coverage of English constructions in both syntax *and* semantics that is well motivated and hence extensible. Interactive facilities are provided to allow users to extend the system vocabulary. The CLE has a modular architecture with well defined interfaces between the various stages of linguistic processing. Unification is used as the mechanism for rule application and passing information in each of morphology, parsing, interpretation, and selectional filtering, so the rules for these components are expressed declaratively. A compact representation of local ambiguities is applied systematically in syntax and semantics, allowing us to adopt the modular staged approach without sacrificing computational efficiency.

1 DESIGN GOALS AND ARCHITECTURE

The CLE is a domain independent system for natural language processing which can be used as the basis for building natural language applications. It is primarily aimed at interactive advisor systems such as interfaces to database management systems and diagnostic expert systems. English sentences are translated by the CLE into logical form expressions which represent their literal meaning. The motivation for choosing such representations as the output of the system is that they are capable of supporting reasoning which is necessary both for natural language disambiguation and for the application task of typical advisor systems.

The main contribution of the CLE is intended to be substantial coverage of in both syntax *and* semantics. This coverage is achieved by taking advantage of recent advances in linguistics giving analyses that are well motivated and hence extensible. Transportability to new applications is also enhanced by providing interactive facilities to allow users to extend the system vocabulary. Current CLE coverage of English syntax and semantics can be summarized roughly as follows:

> Major clause types: declaratives, imperatives, wh- and yes-no questions, relatives, passives, clefts, there-clauses.
>
> Verb phrases: complement subcategorization, control verbs, verb-particles, auxiliaries, tense operators, some adverbials.
>
> Noun phrases: prenominal and postnominal modifiers, lexical and phrasal quantifiers/specifiers.
>
> Coordination: conjunctions and disjunctions of a wide class of noun phrases, verb phrases, and clauses; some common comparatives.
>
> Morphology: inflectional morphology, simple productive cases of derivational morphology.
>
> Core lexicon: 1200 function words and content word stems, 1800 senses and their selectional restrictions.

The CLE has a modular architecture. Sentence processing is performed in the following stages:

- Morphological analysis and sense derivation.
- Syntactic parsing.
- Semantic interpretation and selectional filtering.
- Quantifier scoping.
- Reference resolution.

The current CLE prototype has development versions of components implementing the first four phases

*current address: Artificial Intelligence Center, SRI International, Menlo Park, California 94025, USA

as well as the various sets of rules and lexical entries encoding the linguistic knowledge required by these components. Semantic interpretation results in a level of representation we call 'quasi logical form'. This may be regarded as the natural level of sentence representation resulting from linguistic analysis that applies compositional semantic interpretation rules independently of the influence of context. It differs from fully specified logical form in that it will typically contain quantifiers and operators whose scope has not yet been determined, and also 'anaphoric terms' which stand for entities and relations to be determined by reference resolution. Having quasi logical form as a well-defined level of representation allows the problems of compositional semantics to be tackled separately from the problems of scoping and reference resolution.

There are two important themes that recur throughout the design of the CLE. One is that unification is used as the mechanism for passing information and rule application in each of morphology, parsing, interpretation, and selectional filtering, allowing the rules for these components to be expressed declaratively. (This reliance on unification was the main reason for choosing Prolog as the CLE implementation language.) The other theme is a technique for compact representation of local ambiguities which is applied in a systematic way to arbitrarily complex syntactic and semantic structures. This way of handling ambiguities means that we can adopt the modular staged approach without sacrificing computational efficiency.

2 CATEGORY AND FEATURE SYSTEM

Information about the syntactic and semantic properties of linguistic constituents is represented in the CLE using complex categories that include a principal category symbol and specifications of constraints on the values of syntactic and semantic features – cf. the feature systems of GPSG (Gazdar, et al, 1985) and PATR-II (Shieber, 1986). Categories appear in syntax rules, semantic interpretation rules, and lexical entries. Two categories can be unified if the constraints on their feature values are compatible. Typically, categories appearing in a rule have shared variables which are used to pass information between the categories and other rule components (Sections 3 and 4).

A category consists of a category symbol and a set of feature-value pairs (or feature specifications) represented as a list. Each pair consists of an atomic feature name and a value, which can be an arbitrary Prolog term. In particular, feature values may contain lists or other categories recursively. For example, the category shown below has category symbol `s`, a variable for the feature `type`, and a list structure containing the category `np:[whgap=W]` as the value of `gapsSoughtIn`.

```
s:[type=Type,gapsSoughtIn=[np:[whgap=W]],
   form=tnsd,agr=(\(third/\sing))]
```

Feature values can also be boolean combinations of elements from finite sets; an example shown above is the value 'not third person singular' for the feature `agr`. These values are compiled into terms which unify if and only if the expressions are compatible (see Mellish, 1987). More generally, category compilation in the CLE ensures that category unification is implemented efficiently as Prolog term unification. The values of features not mentioned in a category are compiled according to feature default declarations.

3 SYNTACTIC INFORMATION

Lexical entries consist of a word form, optionally followed by its stem form (for irregular words), then a list of syntactic categories for the word. Regular morphological variants are computed at parse time and cached.

Syntactic and morphological rules consist of an identifier, followed by a list of categories or terminals, the first of which is the mother. Variables over whole categories (as well as category valued features) are allowed. Here is a sample rule:

```
syn(s_np_vp_Normal,...
 [s:[agr=Ag, type=T,...
     gapsSoughtIn=Gi, gapsSoughtOut=Go,...],
  np:[agr=Ag, type=T,  nform=Sfm,...],
  vp:[agr=Ag, subcat=[], subjform=Sfm,
      gapsSoughtIn=Gi,
      gapsSoughtOut=Go,...]]).
```

As can be seen, our formalism is a prototypical unification grammar consisting of a context-free skeleton enriched with features.

There are three noteworthy features of the current syntactic description.

1. Subcategorisation by verbs or adjectives of their arguments is handled by giving them a lexical entry in which the value of a `subcat` feature is a list of the required complements in the order expected. The word is then combined into a phrase with its complements by applications of a rule having roughly the structure:

```
vp:[subcat=Rest, ...] -->
    vp:[subcat=[First|Rest], ...]  First
```

where `First` is a variable over a whole category. Thus a phrase like *give a book to me* will have a structure like:

$$[[[give]_{vp}[a\ book]_{np}]_{vp}[to\ me]_{pp}]_{vp}$$

2. Syntactically predictable alternations are handled by rules which capture the effect of what in Standard Theory Transformational Grammar (Chomsky, 1965) were 'lexically governed' transformations. Thus a rule like:

```
vp:[subcat=[], subjform=it,...] -->
     vp:[subcat=[], extraposes=y,...]
     s:[...]
```

will allow verbs and adjectives like *bother* and *obvious* to appear in a frame like *it [bothers him / is obvious] that S* as well as *that S [bothers him / is obvious]*. The existence of the former is predictable from that of the latter.

3. Unbounded dependency constructions like wh-questions and relative clauses are treated by list valued features which 'thread' the dependency through a tree (see Pereira and Shieber, 1987). A rule introducing such a dependency will 'push' a 'gap' onto the head of the `GapsSoughtIn` list, and a rule discharging such a dependency will 'pop' a gap present on the `GapsSoughtIn` list, resulting in a `GapsSoughtOut` list with one fewer gap on it. All the gaps must be found for the sentence to be grammatical. For example, in the wh-question *Who did John give a book to?* the gap is threaded through the constituents in the order shown below with numbered brackets:

$$who[_1\ did\ john[_2[_3 give[_4 a\ book]][_5 to[_6 \emptyset]]]]$$

The advantage of this treatment is that the same rules as are used for the normal case can be used to build phrases with missing constituents.

The current grammar covers most of the subcategorisation types of English, and handles all types of wh-constructions, (questions, relatives, clefts), passives, existentials, 'transformations' like Dative, Particle movement, Extraposition etc, nominal and verbal pre- and post-modification, and conjunctions, with about 50 rules

4 SEMANTIC INFORMATION

Sense entries, the semantic counterparts to the lexical entries in the syntactic component of the CLE, link every basic word form to a featured category and a logical form that translates the word. Here is a (considerably simplified) example:

```
sense(design,
      vp:[arglist=[(B,comp:[])],eventvar=E,
          subj=A,gapValsIn=G,gapValsOut=G],
      [design1,E,A,B]).
```

This entry shows that the logical form for the verb *design* is a functor with three argument slots, `[design1,E,A,B]`, where `E` is a variable for the event denoted by the verb, and `A` and `B` are variables for the subject and direct object. The argument slots are kept track of by means of verb phrase features. The entry specifies, for instance, that the object argument slot `B` is unified with the meaning `B` of the VP complement. The features `gapValsIn` and `gapValsOut` are used for gap filler threading.

Morphological derivation rules provide senses for regularly formed combinations of basic word forms and one or more affixes. For regular verbs, the basic word form is the infinitive, and the senses of the other verb forms are derived. Morphological derivation also deals with simple cases of agentive *er*-nominalization. The meaning of *designer* is derived from `[design1,E,A,B]` by filling two of the argument slots with suitable forms, and leaving the subject argument slot open.

Semantic interpretation rules provide interpretations for phrases that are the result of syntactic rule applications. Every syntax rule has one or more corresponding semantic rules. Each semantic rule gives the name of the syntax rule that it corresponds to, and specifies how the semantic features and the interpretation of the mother node depend on features and interpretations of the daughter nodes. Rather than using lambda expressions in the way they are traditionally employed in compositional semantics (Montague 1974), we normally employ unification to compress the work of functional application and lambda reduction into one step. Thus arguments are immediately plugged into slots in the logical form that are marked by Prolog variables. The following (simplified) rule for the semantics of $S \rightarrow NP\ VP$ illustrates this:

```
sem(s_np_vp_Normal,
    [(Vp,s:[gapValsIn=Gin,gapValsOut=Gout]),
     (Np,np:[]),
     (Vp,vp:[subj=Np,gapValsIn=Gin,
```

```
gapValsOut=Gout])]).
```

This says that the meaning of the mother is the meaning of the VP daughter with the NP meaning plugged into its subject slot.

The application of the semantic interpretation rules results in logical form expressions that are unresolved as to quantifier scopes and anaphoric possibilities. The unscoped LF that translates *A bishop wanted to visit every college* is:

```
[past,quant(exists,A,[event,A],
        [want1,A,qterm(a1,B,[bishop1,B]),
         quant(exists,C,[event,C],
          [visit1,C,B,
            qterm(every1,D,[college1,D])])])]
```

The two qterms in this LF appear in argument positions that in a scoped LF would be occupied by quantified variables. Note that the qterm variable B occurs not only inside the qterm but also in the subject slot of visit1. Only after scoping will this variable be properly bound by a quantifier.

In the previous example no reference resolution as to the subject of *visit* had to take place: the subject of the matrix sentence is the only candidate. LFs for sentences containing pronouns or definite descriptions, however, will contain unresolved anaphora. *Wren said that he would design a college* will translate into a LF with

```
a_term(pro,C,[and,[human,C],[male,C]])
```

in the subject slot of *design*.

The semantics of long distance dependencies is handled by a mechanism analogous to the gap threading in syntax: the meaning of a displaced constituent is put on a gap-values list and threaded until the gap component is encountered. The interpretation rule for the gap pops this list, as follows:

```
sem(np_WhGap,
   [(Np,np:[gapValsIn=[Np|Rest],
            gapValsOut=Rest])]).
```

The rule shows how the meaning of the phrase, Np, is taken from the top of the list of gap values.

5 SORTAL RESTRICTIONS

When a semantic rule or sense entry is used during the semantic interpretation phase, sortal information is introduced into the logical form for the interpretation of the constituent under analysis. This sortal information is attached to a component of the logical form with the operator ';'. For example, the logical form expression [build1,A,B,trinity1] might become

```
([(build1;([event,human,object],
           proposition)),
  (A;event),
  (B;human),
  (trinity1;object)]
 ;proposition)
```

where event, human, and object, are sorts associated with the arguments of the predicate build1. If B had already been given a sort restriction conflicting with human, e.g., inanimate, then the attempted use of the semantic rule or sense entry would fail, ruling out the interpretation with incompatible sorts.

The sort associated with a functor of arity n is an ordered pair consisting of a list of sortal restrictions on the arguments and a sort for the expression resulting from the application of the functor to its arguments:

$$([\langle sort_1\rangle,\ldots,\langle sort_n\rangle],\langle expression\text{-}sort\rangle)$$

Thus the sort ([event,human,object],proposition) is associated with the predicate build1 in the example.

Sorts used in the CLE are in fact more complex than the atomic ones such as human shown above. In the general case, the sort of each argument in a predication must unify (as a Prolog term) with the sortal restriction that the predicate imposes on that argument. The term for an argument sort is an encoding of the sort of a class of individuals according to a classification hierarchy (Mellish 1987).

Mutually exclusive classes are encoded as terms with different functors, ensuring that they do not unify:

```
abstract(_)          object(_,_,_)
```

Further instantiation of these terms gives finer degrees of classification: the term for a 'human animate object' might be

```
object(animate(human,_),_,_).
```

Functors with more than one argument represent classes having non-exclusive subclassifications. For example, objects in the class 'animate' can be classified with respect to sex or whether they are human. The two-argument functor `animate` has `human` and `animal` as possible values for its first argument, and `male` and `female` as possible values for its second argument. This allows the sort for 'female animate object' to unify with the one shown above.

The sortal restriction for a word sense is not entered directly in the fixed-position format shown above, but rather is compiled from constraints specified for that word sense in the lexicon. The sort hierarchy used in this compilation process is generated from declarations stating subsumption and disjointness relations between entity classes.

6 LEXICAL ACQUISITION TOOL

The lexical acquisition tool VEX (Vocabulary EXpander) allows the creation of CLE lexicon entries by users with knowledge both of English and of the application domain, but not of linguistic theory or of the way lexical entries are represented in the CLE.

VEX is provided with pointers to entries in a 'paradigm' lexicon for a number of representative word usages, and declarative knowledge of the range of sentential contexts in which these usages can occur. It elicits grammaticality judgments from the user to determine which paradigm (or set of paradigms) occurs in the same contexts as the word being defined, and then constructs the new entries by making substitutions in these paradigm entries.

An alternative to this *copy and edit* strategy would be to use knowledge of the function of every feature and other construct in the representation, but this approach would lead to lengthy interaction with the user and make it more difficult to keep up with developments in the feature system. The 'copy and edit' approach, on the other hand, makes VEX independent of most changes to the representation. Furthermore, the fact that its knowledge is specified at the level of word behaviours, means that as the CLE's coverage increases, modifications to this knowledge are easy to make. It also makes robust interaction with the user much easier to achieve.

The process of defining a new word or phrase specified by the user is as follows. First, the user is asked for the gross syntactic category or categories of the new item (noun, verb, etc; no further grammatical knowledge is assumed). The rest of the definition process takes place separately for each category.

After eliciting any irregular inflectional forms, VEX uses its knowledge of the category of the new item and the number of words it consists of to select a subset of the sentence patterns it knows about. Redundancy in this subset is then removed by eliminating patterns whose grammaticality can be deduced from that of other patterns in the subset. The remaining sentences, with forms of the item being defined substituted in, are presented to the user, who states which of them are grammatical.

VEX then tries to find a minimal set of paradigms which, together, occur in all and only the contexts the user has marked as grammatical. If no such set exists, the user is asked to accept one of several additions to, or deletions from, the grammatical set. Such negotiation is needed because it is quite common for users to ignore sentences, to misread them, or simply to have different intuitions on them from those embodied in the CLE's data.

Thus if the user asks to define the phrasal verb 'use up', VEX selects the following sentence patterns to be judged:

```
1 The thingummy used up.
2 The thingummy used the whatsit up.
3 The whatsit was used up by the thingummy.
4 The thingummy used the boojum up very good.
5 The boojum was used up the whatsit by the
    thingummy.
6 The whatsit was used up for the boojum by
    the thingummy.
7 The thingummy used up existing.
8 The thingummy used up the whatsit that the
    boojum existed.
9 The whatsit was used up by the thingummy
    to exist.
```

(Content-free nouns such as 'thingummy' are used to prevent the user being sidetracked into judging semantic acceptability). The user replies that sentences 2, 3 and 9 are grammatical. VEX then asks whether the 'to' in 9 must mean 'in order to'; since it must, sentence 9 can be treated merely as sentence 3 with an optional modifier. This information allows VEX to identify 'use up' as having a single paradigm, that of transitive particle verb.

When, finally, a paradigm set is established, a number of senses of the user's word are derived from them (typically one per paradigm). The user is asked then to

provide sortal information, appropriate to the domain, for each such sense.

7 AMBIGUITIES AND PACKING

Local ambiguities arise from prepositional phrases, compound nominals, multiple word senses, and so on. The CLE 'packing' technique for compact representation of local ambiguities was used by Tomita (1985) and is also implicit in the Earley and CKY parsing algorithms. We have generalised it, however, so that it can be used with categories represented by arbitrary term structures containing variables and we have applied it in the semantic phases of processing as well as parsing.

Each analysis of a constituent can be thought of as having three components: the segment of input text spanned by the constituent, a category, and an internal structure. The basis of the packed representation is that we can abstract away from alternative internal structures of constituents that have the same category because the application of our syntactic and semantic rules depends only on the categories of constituents and not on their internal structures.

Thus the prepositional phrase attachment ambiguity in the sentence *Wren designed the library in Trinity* yields two analyses for the verb phrase following *Wren*. Instead of building explicit trees for these analyses during parsing, the CLE parser maintains two types of record. 'Constituent' records state that a particular segment of input has been analysed as a particular category, whereas 'analysis' records give possible local syntactic structures of such constituents. In the above example, the verb phrase will have two analysis records but only one constituent record; this constituent record is then used to build a single analysis record for the sentence category spanning the whole input.

Packed syntax records are translated directly into packed semantic structures using two additional record types, 'semantic-constituent' records and 'interpretation' records. Interpretation records contain logical form templates with 'references' to daughter constituents so that completed logical forms can be recovered by selecting among the interpretation records for the daughter constituents. In the example above, the alternative semantic interpretations of the verb phrase would be expressed with multiple interpretation records.

In order to handle the more general case of CLE categories that contain arbitrary terms as feature values, we can no longer simply check that categories are identical for packing to take place. Instead, we must check whether the category in a constituent record subsumes the categories in the corresponding analysis or interpretation records. The alternative syntactic analyses or logical forms can then be recovered by unification rather than replacement.

8 PARSING AND INTERPRETATION

Morphological processing takes place in three subphases: word segmentation, word structure analysis, and sense derivation. Segmentation applies spelling rules to a morphologically complex word so that the stem and affixes can be identified. Word structure analysis parses these word components using a simple left corner parser (see below) in order to assign a category to the word. Sense derivation applies another set of rules in order to derive new sense entries from the senses of the stem. Word parsing and sense derivation are similar to the processes of sentence parsing and interpretation, but simpler because packing is not used at the sub-lexical level of analysis.

The parser in the CLE uses a 'left-corner' parsing strategy with top-down filtering (Rosenkrantz and Lewis 1970). This is primarily a bottom-up strategy, but it does a limited amount of top-down processing in order to use the left context to decide whether a particular constituent could occur at a given position in the input. The idea is that, having parsed a constituent bottom-up, the system selects a rule in which that constituent could be the left-most daughter, and it postulates that this rule provides the analysis of the immediately following input, predicting the remaining daughters in the rule top-down. In parsing the next segment of the input, it considers only analyses that are consistent with the selected rule, by checking that each constituent it builds bottom-up could be a 'left corner' of the next daughter needed to satisfy the rule.

Pure top-down parsing has the problem that left-recursive rules lead to infinite recursion. Pure bottom-up parsing has the disadvantage that every possible empty category (or 'gap') has to be proposed at each point in the input string, since the parser cannot tell from the input string where gaps exist. A left-corner parser avoids the left-recursion problem because it is fundamentally bottom-up, and it can be made to solve the gap-proliferation problem by using the top-down filtering to make sure that gaps are proposed only in places they can actually occur.

The information for testing whether one constituent can be a left corner of another constituent is precom-

puted into the rule tables used in the prediction steps of the parser. A well-formed substring table is also used to avoid re-analysis of already parsed constituents after backtracking. The implementation of these elaborations of the basic algorithm allows for the possibility of fully general CLE categories containing variables (see Alshawi, et al, 1988).

Another parser based on Prolog unification and a bottom-up strategy is the BUP system (Matsumoto, et al, 1983). Our parser is an extension of this in that its top-down filtering uses feature values as well as major categories, and its well-formed substring table construction is sound since it is based on subsumption checking.

The process of applying semantic interpretation rules to produce unscoped logical forms is much simpler than parsing. The system merely traces the syntactic analysis records down from the start symbol to look up the word senses of all the lexical items and apply the semantic interpretation rules to all the complex constituents that appear in complete sentence analyses.

To process a constituent, the system finds a syntactic analysis record for the constituent and recursively computes an interpretation for all the daughter constituents in that analysis. It then looks for a semantic rule that corresponds to the syntax rule for the analysis and is compatible with the interpretations chosen for the daughter constituents. After unification, the relevant instance of the mother logical form is extracted from the rule and a sorted version of the form is built, making sure that it does not violate any sortal restrictions.

9 QUANTIFIER SCOPING

An algorithm for generating the possible quantifier scopings for a sentence, in order of preference, has been developed and implemented (Moran, 1988). Quantifier scoping generates the two possible readings for *A bishop wanted to visit every college* from the unscoped LF (given in Section 4 above) by replacing the `qterms` with quant expressions corresponding to generalized quantifiers. In this case, the preferred scoping preserves surface order:

```
quant(exists,A,[bishop1,A],
  quant(forall,B,[college1,B],
    [past,quant(exists,C,[event,C],
            [want1,C,A,
              quant(exists,D,[event,D],
                [visit1,D,A,B])])])).
```

The scoping assigned to a quantifier is determined by its interactions with other quantifiers, logical operators (e.g., modals and negation), and boundaries of certain syntactic constituents (e.g., major clauses and relative clauses). When a potential scoping is logically equivalent to another, one is discarded.

The algorithm is intended as the first stage of a two-stage process for determining the preferred quantifier scoping for a sentence. The first stage generates the scopings and orders them using linguistic criteria; the second stage (future work) would use pragmatic information (e.g., functional dependencies and discourse criteria) to modify the initial ordering. The motivation for this division can be seen in the two closely related sentences *John visited every house on a street* and *John visited every house on a square*; both have a preferred quantifier scoping of *on a particular street/square*. However, the latter has a secondary scoping (*any house on any square*) which is very hard to get for the former. We attribute this difference to an application of pragmatic information: the typical house is on a street, but not on a square.

Our algorithm, which is an extension of the one described by Hobbs and Shieber (1987), traverses the unscoped logical form, collecting the quantifier terms into a 'store'; then as the scoping for each quantifier term is determined, it is 'pulled' out of the store, producing a scoped logical form. Unlike the flat stores of Cooper (1983) and the LUNAR system (Woods, 1977), our algorithm uses a store in which the structure of the quantifier terms reflects their relative positions in the unscoped logical form, so we can apply order-dependent linguistic preferences.

The selection of a preferred quantifier scoping for the whole sentence is the result of a sequence of pairwise comparisons between the individual quantifier terms, logical operators, and constituent boundaries. The relative scoping preferences of the individual quantifiers are not embedded in the algorithm, but are specified by a set of rules. Many of these rules have appeared in the linguistics literature and have been used in various natural language processing systems. However, the coordination of these rules and the resulting coverage represents a significant contribution. Because experimental data on human quantifier-scoping preferences are still fragmentary, we chose to design a system in which the set of preference rules could be easily modified and expanded.

An improved scoping algorithm has recently been included into the CLE by Fernando Pereira. The format of binding operators can now be specified declaratively, and cases not handled in previous work, such

as conjoined quantified noun phrases, are now treated. For example, for sentences involving noun phrases such as *most doctors and some engineers* the algorithm produces distributive readings which respect the constraint that both quantifiers must have 'parallel' scopes.

10 FURTHER RESEARCH

We intend to extend the treatment of tense, aspect, events and temporal reference, and of collective readings of noun phrases. Work has started on the design of the reference resolution component and on a more detailed treatment of noun subcategorization, including acquisition of words formed by nonproductive derivations. In the longer term, we would like to develop the CLE in a number of directions including the provision of capabilities for language generation, for the interpretation of abbreviated and elliptical expressions, and for using reasoning to aid interpretation and application interfacing.

ACKNOWLEDGEMENTS

Development of the CLE has been carried out as part of a research programme in natural language processing supported by the UK Department of Trade and Industry under an Alvey grant and by members of the NATTIE consortium (British Aerospace, British Telecom, Hewlett Packard, ICL, Olivetti, Philips, Shell Research, and SRI). We would like to thank the Alvey Directorate and the consortium members for this funding, and Fernando Pereira for valuable criticisms and suggestions while guiding the research.

REFERENCES

Alshawi, H., D. M. Carter, J. van Eijck, R. C. Moore, D. B. Moran, F. C. N. Pereira, S. G. Pulman and A. G. Smith. (1988) *Interim Report on the SRI Core Language Engine.* Technical Report CCSRC-5, Cambridge Computer Science Research Centre, SRI International, Cambridge, England.

Chomsky, N. (1965) *Aspects of the Theory of Syntax.* MIT Press, Cambridge, Massachusetts.

Cooper, R. (1983) *Quantification and Syntactic Theory.* D. Reidel, Dordrecht, Holland.

Gazdar, G., E. Klein, G. K. Pullum, and I. A. Sag (1985) *Generalised Phrase Structure Grammar.* Blackwell, Oxford.

Hobbs, J. R., and S. M. Shieber (1987) An Algorithm for Generating Quantifier Scopings. *Computational Linguistics*, Vol. 13, no 1–2.

Matsumoto, Y., H. Tanaka, H. Hirakawa, H. Miyoshi, and H. Yasukawa (1983) BUP: a bottom-up parser embedded in Prolog. *New Generation Computing*, Vol. 1, no 2, pp. 145–158.

Mellish, C. S. (1987) Implementing Systemic Classification by Unification. *Computational Linguistics*, Vol. 14, no 1, pp. 40-51.

Montague, R. (1974) *Formal Philosophy: Selected Papers of Richard Montague.* Ed. by Richmond Thomason, Yale University Press, New Haven.

Moran, D. B. (1988) Quantifier Scoping in the SRI Core Language Engine. *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, Buffalo, New York.

Pereira, F. C. N., and S. M. Shieber (1987) *Prolog and Natural-Language Analysis.* Center for the Study of Language and Information, Stanford.

Rosenkrantz, D. J., and P. M. Lewis (1970) Deterministic Left Corner Parsing. *Conference Record of the 11th Annual Symposium on Switching and Automata Theory*, IEEE, pp. 139–152.

Tomita, M. (1985) An Efficient Context-Free Parsing Algorithm for Natural Languages. Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Los Angeles, California, pp. 756–764.

Woods, W. A. (1977) Semantics and Quantification in Natural Language Question Answering. In: *Advances in Computers, Volume 17*, Academic Press, New York, New York: 1-87.