UNIVERSITY OF DURHAM: DESCRIPTION OF THE LOLITA SYSTEM AS USED IN MUC-6.

Richard Morgan, Roberto Garigliano,
Paul Callaghan, Sanjay Poria, Mark Smith, Agnieszka Urbanowicz,
Russell Collingham, Marco Costantino, Chris Cooper, and the LOLITA Group.

Laboratory for Natural Language Engineering
Department of Computer Science
University of Durham
South Road
Durham, DH1 3LE, United Kingdom.
email: R.G.Morgan@durham.ac.uk

INTRODUCTION

This document describes the LOLITA system and how it was extended to run the four MUC tasks, discusses the resulting system's performance on the required "walk-through" article, and then considers the performance of this system on the final evaluation set.

Because of the design of LOLITA, the code implementing the four MUC tasks is almost trivial, since it is built using complex facilities provided by a 'core' system. Hence, after some background to the LOLITA project, the 'core' of LOLITA is described without much reference to the MUC tasks. Then, the analysis of the walk-through article is presented and the implementation of each task discussed using examples from the walk-through article. This is followed by an overall view of the system's performance and some conclusions.

BACKGROUND

The LOLITA (Large-scale, Object-based, Linguistic Interactor, Translator, and Analyser) system is designed as a general purpose Natural Language Processing (NLP) system and has been under development at the University of Durham since 1986. The system is designed to provide NLP capabilities to support many applications in multiple domains. It attempts to do this by providing a *core* platform upon which different applications can be built. This core platform provides two main facilities: analysis, which converts text to a logical representation of its meaning, and generation, which expresses information represented in this logical form as text. Unlike many of its contemporary NLP systems, the LOLITA system is not designed as a framework that can be tailored to specific domains, but as system that brings its knowledge of specific domains to bear as and when appropriate.

The Laboratory for Natural Language Engineering (LNLE) at the University of Durham is focussed on developing this core. Prototype applications have been built using the core facilities; some of them are listed below:

- Information extraction: production of summary and other templates.
- Simple meaning-based translation: currently Italian to English.
- Natural language query: supplying information to LOLITA and then asking questions about this information.
- Dialogue: a model of dialogue has been implemented.
- Chinese language tutoring: a mixed English and Chinese grammar allows detection of students of Chinese using English constructions, and diagnosis of problems.

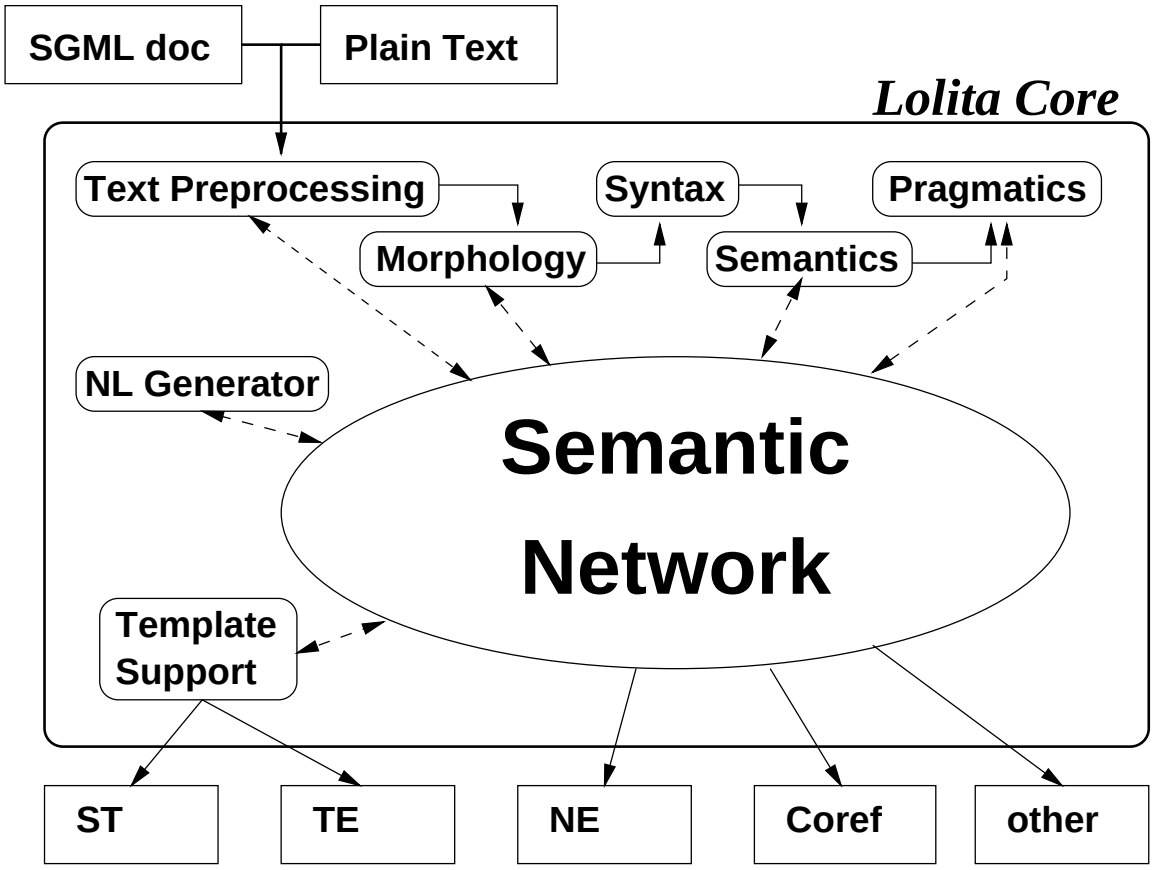

Figure 1: Block Diagram of the LOLITA Core plus some applications

Work is also in progress to integrate the text-based core with a speech recogniser. The use of the core to improve determination of prosody in synthetic speech is also being investigated.

The MUC-6 competition has provided an opportunity for the Laboratory for Natural Language Engineering to evaluate the approach used in the LOLITA system on some very specific tasks as well as a chance to strengthen the system's performance in the domain of newspaper articles. Given the wider aims of the project, the approach taken was to put minimal effort into the development of the four new applications needed for the MUC-6 tasks and maximum effort into the development and improvement of the core system (although work was concentrated to some extent in areas of the core stressed by the MUC tasks).

Of the 35 person-months spent in preparing the system for MUC, we estimate that 95% was on the core system. This work included providing some completely new (but general) functionality to the core as well as improving existing functionality and performance. A considerable amount of integration work was performed to utilise the Brill tagger [1], and to make use of tables of data such as the MUC gazetteer and a large list of common companies. However, neither was used in the formal evaluation due to their limited effect on performance.

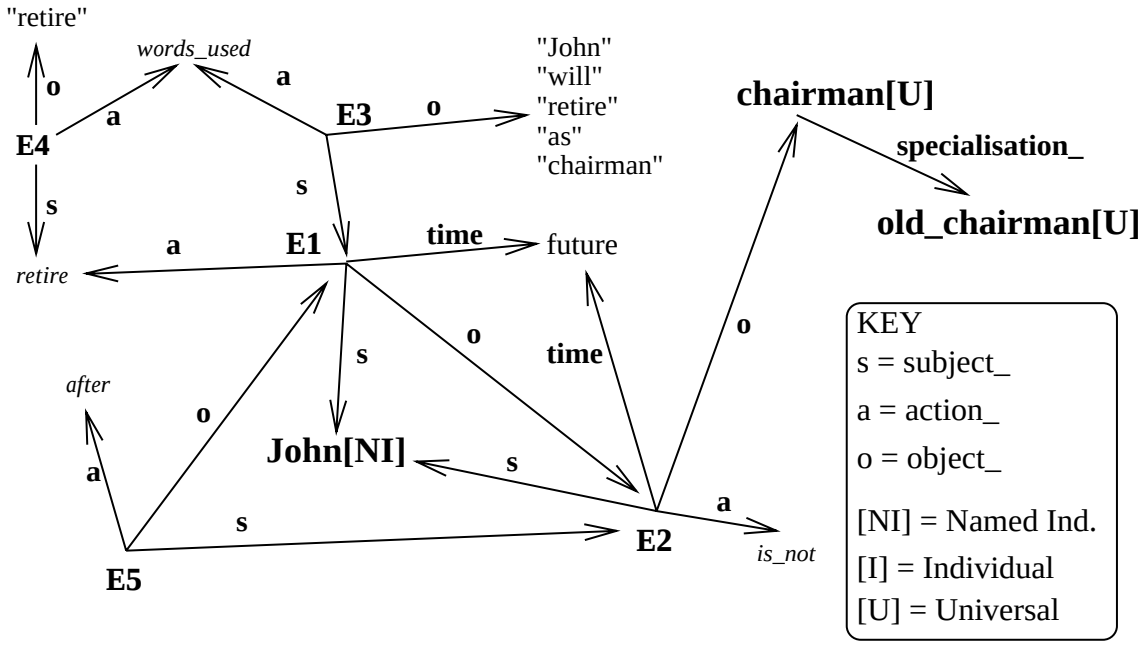

Figure 2: Example piece of Semantic Net, for the sentence "John will retire as chairman". It is given here as an example of SemNet structure, and its meaning is discussed in the section on the Semantic Net. The full structure is not shown, for reasons of space.

ARCHITECTURE

Overview

LOLITA is designed as a core system supplemented with a set of small applications, the former supplying basic NL facilities to the latter. Figure 1 shows the MUC-relevant parts. The most important part of the core is the large knowledge-base, which we call the Semantic Network, SemNet or net, for short. It is heavily used in most stages of analysis, and the results of analysis are added to it, as a disambiguated logical representation of the input. The analysis stages are fairly standard, and are arranged in a pipeline. Each is implemented in a rule-based way. We do not currently use any form of stochastic or adaptive techniques in the main system.

Applications can then read the results of analysis from the SemNet, and generally interrogate the contents of the SemNet. Some central "support" facilities are provided to aid application writing, such as the general template mechanism and the NL generator - which translates pieces of the SemNet into English (or, recently added, into Spanish). More detail on the architecture of LOLITA can be found in [6].

The Semantic Network

The SemNet is a 100,000 node, directed hyper-graph. Each node has a set of links, plus a set of "control variables" (or *controls*). Some nodes have an associated "name": this is usually a single word which characterises the meaning of the node. Each link has an arc and a set of targets. Targets are other nodes, and the arc too is just a node. Nodes correspond to concepts of entities or events. Links correspond to relationships between nodes. Since an arc is also a node, the concepts of the different kinds of relationship possible between nodes can be represented in the same formalism as more concrete concepts. In this system, the "meaning" of any particular node is given by its connections, its relative position in the net.

Controls indicate basic information about a node, such as its type (e.g. event, entity, relation), its family (e.g. human, inanimate, food, organisation), its lexical type (e.g. noun, preposition, adverb) - as appropriate. An important control is a node's *rank*: this encodes quantification information. Concepts of general sets

have a *Universal* rank, specifically named objects have a *Named Individual* rank, and general individuals an *Individual* rank. There are several other less important ranks, used for things like encoding script-like information or existential quantification. Controls could be represented using links, but for efficiency reasons this more compact form is used.

There are approximately 60 different arcs. The arcs `subject_` `action_`, and `object_` are used to represent the basic roles of an event. Events can have other arcs, such as those indicating temporal information, the status of the information (e.g., known fact, hypothesis, etc), or arcs that indicate the source of the information. Most arcs also have inverses: e.g. the `subject_` arc has the inverse `subject_of_`, which allows determination of the events in which a particular concept played the subject role.

Concepts are connected with arcs such as `specialisation_` (and its inverse, `generalisation_`), or `instance_` (inverse `universal_`). Specialisation links a set to one possible subset; for example, in figure 2, `chairman[U]` represents the set of all possible chairmen, and `old_chairman[U]` the set of all possible old chairmen. Between the former and the latter is a `specialisation_` link, indicating that old chairmen are a subset of chairmen. Conversely, the latter is linked to the former with a `generalisation_` link, representing a superset. Using the `specialisation_` link, hierarchies of concepts are specified. The `instance_` arc connects a concept to an instance of that concept: e.g., a particular chairman `chairman1[I]` would be linked to `chairman[U]` by an `instance_` link. Other links between concepts include `synonym_` and `antonym_`

The SemNet is used to hold several kinds of information:

- concept hierarchies: built with arcs such as `generalisation_` concept hierarchies encodes knowledge like "man is_a mammal is_a vertebrate" etc. They prevent duplication of information by allowing information to be inherited within the hierarchy.
- lexical information: actual words are represented in the net, and their properties are stored in the net, as opposed to having a separate lexicon. The lexical-level nodes are indexed via a simple dictionary: ie, a mapping from root words to all the senses of that word. Note that the lexical forms are distinct from the concepts - they are linked by a `concept_` arc. Concepts are linked to lexical forms by a link named after the language of interest. E.g., `dog[U]` has a link `english` to the noun form of "dog", and a link `italian` to the Italian word "cane".
- prototypical events: these define restrictions on events by providing 'templates' for events, e.g. by imposing selectional restrictions on the roles in an event. "Human owners own things" says that only humans can take the subject role in 'ownership' events.
- general events: other kinds of information. For example, the content of a MUC article would come in this class, when analysed.

The bulk of the net (70%) comes from WordNet, a database containing lexical and semantic information about word forms in English [5]. More details about the formalism used in the net can be found in [4].

Referring back to the Original Text

Before MUC, LOLITA did not have a method of referring back to its input: the previous orientation was to move from language-dependent surface forms to a language-independent logical representation. Therefore, information about the surface form was discarded. Since the ability of reference has many uses outside of the MUC tasks, a more general mechanism was designed and added to the core. It allows fine-grain connection of the analysis results to the sections of the document giving rise to those results. The system allocates new SemNet nodes to components of the document (words, phrases, sentences, ...), which act as references into the document. This is called the 'Textref' system and has several uses:

- It allows the core to analyse input which talks about surface components of the input text. For example, a user might be able to ask 'What is meant by "organisation" in the second paragraph of the document?', or make statements such as 'When I wrote "pointing", I was referring to brickwork'.

- It enables applications to produce output which is highly related to the original text. Clearly, the MUC tasks are an example of this, since they require exact phrases. Another possibility is the provision of hypertext-style links to the relevant parts of the original documents in information extraction or summarisation tasks.
- Previous LOLITA applications have relied on the core system's generator [6] to produce output. This generator relies heavily on the core analysis, and although it performs well given a correct analysis, errors in the analysis can produce very strange output and a drastic reduction in the perceived performance of the system. Textrefs enable more robust reporting of results, as witnessed in a significant performance improvement in our non-MUC template generation applications.
- The Textref system can also be used to provide convenient debugging information, since they allow developers to relate internal structures produced by the system to the portions of the text from which they were derived.

Textrefs allow the document structure to be fully represented in the net, and represented uniformly with the other information in the system. At the word level, a Textref signifies a specific *occurrence* of a word at a certain position in the input, and is distinct from the nodes representing the lexical or semantic forms of its root form. It is an `instance_` of the universal concept of all occurrences of that word. Concept nodes and Textref nodes are linked by an event with the internal action `words_used`. Two examples may be seen in figure 2: single words are attached to the "key" words of the sentence (only 'retire' is shown), and all of the Textrefs in the sentence are attached to the node representing the whole event.

Text Pre-processing

Core analysis of textual input starts from a LOLITA-specific SGML representation of the input (called an SGML tree). Individual applications must convert from their own formats (e.g. plain text, MUC WSJ articles, LaTeX, HTML, ...) into this internal format. The MUC converter is just a simple SGML parser. The preprocessor then adds additional structure to the internal SGML tree where necessary. In particular the following structures are handled in the order given: reported speech, paragraphs, sentences and words. Markers for reported speech are distributed over all sentences inside the quotes. Lastly, each word is allocated a Textref.

Morphology

Morphology is applied to an SGML tree whose leaves are individual word tokens, and whose nodes represent the structure of the document. A few transformations are done on this structure to unpack contractions (e.g. "I'll" expanded to "I will"), expand monetary and numeric expressions (eg "$10 million" to "10 million dollars"), and to transform certain surface-level idiomatic phrases (eg "in charge of"). Some splitting of hyphenated words is also done. Then, the basic morphology function is mapped on to all leaves (with additional treatment provided for sentence initial words).

Lookups in the dictionary are done with the root forms suggested by affix stripping. If successful, a word is linked to lexical and semantic nodes, allowing access to lexical and semantic information during the rest of morphology, parsing, and semantics. Affix stripping loses information such as number and case, so this information is represented using a Feature system. Features are used in parsing (described below). Other Features include word class (Noun, Verb, ...) and some semantic-based Features. Finally, possible syntactic categories for a word are determined from the lexical (and sometimes semantic) node information. Thus, each leaf is mapped to a set of alternatives, varying in category and Features, which represent all possible interpretations of that leaf.

Parsing

There are four stages in parsing:

```
sen                              -- sentence branch
 full_propernoun                 -- proper noun phrase
   propernoun JOHN [Sexed]
 neg_copula                      -- copula verb phrase
   asCopulaN RETIRE [Fut]        --
   comnoun CHAIRMAN [Sing,Per3]
```

Figure 3: Parse tree for "John will retire as chairman".

- A pre-parser which identifies and provides structure for monetary expressions. This stage is currently underused, and would provide a measure of robustness for the kind of expressions used in Named Entity, should parsing fail. It is implemented using a simple grammar of low ambiguity and a parser which attempts to find the largest non-overlapping sequences which match the grammar (working from left to right).

- Parsing of whole sentences using the Tomita algorithm [7]. The main system grammar is large and highly ambiguous, so a powerful algorithm is required. Our grammar is written in a context-free style, using a simple feature system to parametrise pieces of grammar, and contains some rules for handling non-grammatical input. It is transformed into approximately 1900 rules of the type `A -> X` or `A -> X Y`, where `A` is a non-terminal, and `X`,`Y` can be terminals or non-terminals. The result of this stage is a "parse forest", a directed acyclic graph which indicates all possible parses. Due to the complexity of the grammar, this forest is frequently very large, implying many possible parses.

- Decoding of the parse forest. The forest is selectively explored from the topmost node, using heuristics such as Feature consistency and hand-assigned likelihoods of certain grammatical constructions. Feature errors and unlikely pieces of grammar involve a cost: the aim of the search is to extract the set of lowest-cost trees.

- Selection of best parse tree: subsequent analysis operates on a single tree. The lowest cost set is ordered on the basis of several heuristics on the form of the tree - for example, preferring a deeper tree. It is possible for the subsequent analysis to reject suggested trees, and try the next best, but this option is not used in our MUC system. Work is underway to improve the handling of structural ambiguity, possibly by passing a graph structure to subsequent analysis.

- Normalisation: syntax-based, meaning-preserving transformations are applied to the trees to reduce the number of cases required in semantics. A prime example of this is passive to active, ie "I was bitten by a dog" changed to "A dog bit me". Another class involves transformations such as "You are surprised" to "*SOMETHING* surprised you", which makes explicit the object doing the surprising.

Parsing can sometimes fail on very large forests: decoding these requires a lot of resources (time, memory). Rather than cause a crash due to overrunning limits, the parse is abandoned. This is implemented by fixing a time-limit on the process - resource usage being proportional to time: we refer to expiry of the time limit as a 'timeout'. It is also possible for parses to fail if the sentence can't be analysed with the main grammar. If the parse fails, analysis is discontinued on that sentence - so no semantic result is produced.

An example parse is given in figure 3. Note that 'will' and 'as' are missing. As so-called function words, they don't carry much inherent semantic meaning, so the tense information of 'will' is transferred to the Features of the main verb, and the copula function of 'as' is transformed into a syntactic construct. This simplifies the semantic rules.

Analysis of Meaning

This section describes how the parse tree is converted to a disambiguated piece of SemNet. There are two stages, which we call 'Semantic' and 'Pragmatic'. The Semantic analysis is compositional in general:

the meaning of a tree is built from the meanings of its subtrees. A mechanism goes through the parse tree in depth-first, post-order traversal, applying semantic rules mainly on the basis of the syntactic phrase type of the current tree node. If the meaning of a particular subtree is unambiguous in role, the Textrefs for the text in that subtree are connected to that meaning. Since the meanings can be nodes which already have Textrefs connected, then particular nodes can collect Textrefs for all occurrences of their mention. This Textref handling is completely invisible to the semantic rules.

A state value, the "context", is passed around during traversal: this holds possible referents in order of occurrence, and is used to resolve anaphoric expressions. Use of this context prevents the semantics being purely compositional.

The 'meaning' of most leaves is the semantic node associated with the word at the Morphology stage. The node is passed to the leaf's parent in the form of a 'role' structure, which indicates the role the node may play in the semantics of the parent. Often this is unknown, but in cases like verbs, it can be determined as the `act`. The actual role structure allows for representation of semantic ambiguity.

Branches are associated with rules for combining the semantics of the subtrees. The following example rule handles phrases like "Alan Gottesman, an analyst with PaineWebber", which is a propernoun phrase followed by a noun phrase describing the propernoun. Reading from the bottom, it labels the role from the left as `subject_`, and the right role becomes `object_` Then, add a role indicating the `action_` of 'equality'. Tense information about the phrase is added from the Features `fs`, and an "internal" event built from the roles collected: this links Gottesman with the concept of the individual who is an analyst with PaineWebber. Finally, the `subject_` of the event is returned as the meaning of that subtree.

```
> meta_branch na fs
>   | na `is_in` ["full_propernoun_description ","description "]
>     = extractrole Subj `compose` newnodes_InternalEvent
>       `compose` add_tense_info fs
>       `compose` addroles (sourcerole `join_arl_f` is_a_role na)
>       `compose` labelboth Subj Obj
```

The main task of the Pragmatic stage is disambiguation and type checking. Lexical ambiguities and anaphora are resolved using a series of preference heuristics which are first applied to disambiguate the action of the event. Once the action is known, any knowledge available from the prototype event associated with that action can be used to rule out pragmatically implausible readings, as well as to aid disambiguation of the remaining elements of the event (in the spirit of [8]).

The contents of the current context together with the topic of the text (the latter is given to the system in advance) influence the choice of word senses: those meanings are preferred which are semantically closer to the meanings present in the context or the topic, where semantic closeness is computed on the basis of the distance between nodes in the network. Other factors may cause one concept to be preferred over others, such as the amount of knowledge the system has about a given concept, or the concept's frequency of use.

Each heuristic eliminates any meanings which are not the 'preferred' ones. Given that the less favoured meanings are rejected at once and no backtracking mechanism is used at present, the order of application of heuristics can have a big effect on the final interpretation. The order used has been developed by trial and error to get the desired meaning in the majority of cases in a small test set. In general, the cheaper heuristics are applied first, before using the more powerful but more expensive deep heuristics.

Once an event is disambiguated, the system attempts to establish plausible connections between it and the previously processed discourse. Co-reference links between entities and events are made and a set of new elements is added to the global context.

Reference Resolution

An initial stage of this is done "on the fly" in semantics. The context structure holds possible referents, ordered by recency of occurrence and with semantic and feature information attached to aid disambiguation.

Anaphoric expressions result in this context structure being examined for possible candidates which have appropriate feature and semantic information attached; if more than one candidate exists, then a new node is created to represent the alternatives and is linked to each of them. This new node is then returned as the meaning of the anaphor; later stages (eg Pragmatics) will attempt to disambiguate the reference, and will replace the new node with the chosen node.

A later stage of analysis examines the recently built pieces of net and attempts to unify those which are similar. This makes correspondences which were not picked up during the semantic analysis of individual sentences. A similar stage was added to help unify certain occurrences of proper names - cases such as 'FAA' and 'Federal Aviation Authority', and abbreviated forms such as 'PanAm' and 'Pan American'. In brief, the method looks for correspondences in the surface text attached to Named Individual nodes (ie, resulting from proper nouns). Furthermore, this process is used on titles: the grammar of article titles is quite different from that for normal text, so we avoided full analysis of titles and joined title Textrefs to nodes when a surface match was found.

Template Support

The processes involved in producing templates can be generalised, hence the core contains a mechanism to help write templates at an abstract level. This mechanism handles search through the net, use of inference rules to derive implicit facts, and general output formatting. A fairly sophisticated facility existed pre-MUC (as had a few template applications); for MUC, support for *hyper-templates* (templates that can refer to other templates) was added, and - via the Textref system - the ability to reproduce surface text.

A Template contains a predefined set of slots with associated fill-in rules that direct the search for appropriate information in the net. The slot fill-in rules are predicates that check node controls, or use the inference functions available in the core. There are currently four slot types, distinguished by how the slot output is produced:

- Concept Slot. This type of slot has associated with it a rule which produces a list of concept nodes with which the slot should be filled. Each concept node represents one slot fill and the generator, or the Textref system, is used to express them in English.
- Textref Slot. Some concept nodes may have more than one related Textref. In concept slots, some default rules are used to pick the most appropriate one, but for situations in which more control is required, the Textref slot allows its associated rule to define precisely the Textref to be used.
- String Slot. The slot fill rule directly produces a list of the strings to fill the slot.
- Template Reference Slot. The output consists of a reference to another template, enabling *hyper-templates*.

Currently, three classes of template exist:

- event-based templates - where one clearly identifiable event is the subject of the article. For example, a template regarding a *"takeover"* will include all the information (separated in different slots), referring to the takeover itself which represents the main event of the template.
- summary-templates - where the article does not contain a prominent event. The summary template is thus a collection of different kinds of information extracted from the source article. A summary template, for example, can consist of the slots: personal names, organisations, numeric expressions etc., found in the source article.
- hyper-templates. Hyper-templates are structures whose slots can refer to other templates, thus creating a graph of templates. Hyper-templates have been used for MUC-6 scenario templates.

Implementation and Operating Details

LOLITA is written mostly in Haskell, a non-strict functional programming language [3]. Two resource-critical sections are written in C - the parser and the SemNet data structure and its access functions. Haskell has some similarity to LISP, such as building programs by writing functions, a garbage-collected heap, lists as a basic type, and full higher-order use of functions. However, it provides excellent support for modern Software Engineering, such as modularity, constrained polymorphism, a strong but flexible type system. It also enforces referential transparency and allows coding in a 'lazy' style, which means code is not executed unless needed. Thus, whilst our system has the external appearance of a pipeline architecture, the evaluation of individual pieces of code need not occur in that strict order. We argue that Haskell allows us to write complex code much more easily than, say, C or LISP.

LOLITA occupies approx. 45,000 lines of Haskell, plus some 6,000 lines of C, in 300 modules. It can run on many machines (ie, those for which there is a Haskell compiler), and is normally given a heap size of 60 megabytes. To give an idea of speed, the final evaluation, if run on a single 70MHz SUN machine, took approximately 24 hours. This performance has been achieved using the Glasgow Haskell compiler[2], and with a considerable amount of assistance from the Glasgow Haskell Group, for which we are grateful.

The heap limit has been mentioned as one resource limit. The other one was the 'timeout' value, which is a seconds-per-word limit for parsing. This was set quite conservatively for the final run: too high and we risked a crash due to garbage collection being unable to reclaim enough heap, too low and complex parses are rejected. We had no heap failures during the evaluation.

IMPLEMENTATION OF THE MUC TASKS - WITH WALK-THROUGH EXAMPLES

In the following section, we use examples from the walk-through article to show how the four MUC tasks have been implemented on top of the LOLITA core. Unfortunately the walk-through article contained several features which highlighted some bugs in the core analysis. As a result the scores for this article were well below the formal evaluation average, and in some cases we have described how the task works given output from a corrected version of the core system. It is an interesting example, however, because it demonstrates how small errors can have a wide impact when attempting to perform a 'deep' analysis of the article's meaning.

The LOLITA architecture means that if core analysis is faulty, there is little that can be done in the MUC task modules to correct it. Hence, how the core performs is of prime interest. Several aspects of the core analysis for walk-through article are presented first, followed by the performance of the four tasks, then a few comments on how we managed to improve scores on the walk-through article.

Reported speech

The walk-through article contained a higher proportion of reported speech than most of the formal training articles as well as some other uses of quotation marks. This caused several problems for the LOLITA system and was the principal factor in the low walk-through article score.

The most severe problem was associated with reported speech that covered more than one sentence. The LOLITA system handles these by creating a series of sentences, each of which is enclosed in quotation marks. Unfortunately this processing occurs before the document is stored in the SemNet, and as a result, additional quotation marks are added to the final output. Since the NE and CO scorers take notice of exact positions within the text, they were confused by these additional symbols and many of the markups after the first occurrence of these additional quotes, were scored as incorrect. Unfortunately neither scorer reported an error, and so the problem remained undetected until after the formal evaluation (and indeed until after the MUC-6 conference itself).

Further problems were created by the presence of a typographical error in the original text. In the fourth from last paragraph, one of the sections of reported speech contained two closing quotation marks:

```
Mr. James ... says that because he "had a great time" in advertising," ...
```

Unfortunately the system had no mechanism for recovering from this situation, and so the second closing quote was considered as an opening quote. This resulted in a reversal of what was considered as normal and reported speech in the remainder of the document, causing numerous problems.

The Textref System

The Textref System has been difficult to implement due to the complexity of passing this additional information through all of the processing stages without introducing errors. As a result it has consumed a significant part of the development effort and has also contributed to a significant proportion of the errors in core analysis.

One such problem that has been identified in the analysis of the walk-through article is the copying of the Textrefs from one concept to another. As analysis progresses, the system may change a decision about what concept a particular part of the text referred to. In this case it should simply move the corresponding Textrefs from one concept to another, but in some instances it was copying them. This resulted in a semantic output which indicates that a particular phrase referred to more than one concept, causing problems for the Coref task.

Parsing

For the walk-through article, visual inspection shows that approximately half of the parses are reasonable. Unfortunately, many of these are unimportant, and frequently short, sentences. Many of the important sentences are long, and some cause our parse decoding stage some problems, in that they 'time out'. There were three such failures, and four "completely ungrammatical" failures - quite significant in an article of 50 sentences (14%), since no analysis is produced for these failed parses. One of these failures was due to the quote imbalance in the 'taskmaster' sentence. A more serious loss for the scenario template task was the sentence about Mr. James stepping down, which timed out.

Of the sentences that parse, particular problems were:

- Quotes and reported speech: Not withstanding the previously mentioned problems with reported speech, the parser still had additional problems because the current grammar is quite restrictive about what can appear inside quotation marks.
- Errors in the SemNet: if the basic lexical information is wrong, then a word may not be parsed with the right category. For example, a word like 'now' has an adverbial sense, but only the sub-conjunction and temporal noun sense were available. Such errors have the effect of corrupting the remainder of the parse, as the grammar has to use a marginal interpretation to work round the miscategorisation, resulting in very strange parses.
- Inadequacies in the methods for selecting the best parse.
- Unfamiliar grammatical constructions.
- Full parsing can force poor local decisions. In several cases, clearly identifiable named entities are not recognised as such because the parser is attempting to produce a full parse of the sentence despite the fact that some of the grammar is missing, and this can only be done by taking an alternative parse for the named entity. Further work on the pre-parser for named entities would have reduced this problem.

Another way of looking at parsing performance is examining the sequences of Textref that get attached to net nodes. This is a good debugging aid, showing what text gave rise to particular nodes, and also allowing us to trace the semantics produced from certain parts of the text. Figure 3 shows the semantics created for "Mr. James". The first column shows what other concepts are linked to the concept of "Mr. James", the second shows the text sequences that are considered as referring to "Mr. James", and the third gives a broad grammatical classification for the text sequence.

Particular points of interest are marked X1 and X2. X1 is parsed as some kind of NP, but the associated surface text is not understandable as such: clearly, a parsing error. X2 shows an unusual way of bracketing:

basically, '(Mr) (James, 47 years old,)' is produced instead of the expected '(Mr James,) (47 years old,)'. Some repetition and overlap will be seen in the Textref results: these are known bugs.

Semantics

Even if the whole parse was problematic, semantics is sometimes able to extract reasonable analyses for some sub-trees, providing some measure of robustness. There is some evidence of it happening in the Walk-through Article. Proper nouns and referents are at least recognised, if not correctly interpreted. Several key points of the article are also identified despite weak parses.

For example, 'Mr Dooner' is identified as the subject of 'to succeed' with 'Mr James' as the object, and the sense of the verb is correctly disambiguated because of the pre-defined topic of the article. 'Mr James' is recognised as the subject of the 'retiring' event; however, the system has problems in deciding that the vacated posts will be taken over by 'Mr Dooner' due to the failure in the unification of the 'retiring' and 'succeeding' events. Other events which are erroneously connected with 'Mr James' or 'Mr Dooner' include 'guiding' attributed to 'Mr James' (figure 4) and 'acquiring', 'materialising' to 'Mr Dooner'.

```
   * james: 97751 *
universal_:
  james - 94505 - rank: [U]
subject_of:
  event - 99808 - rank: [U] (keep)
  event - 99700 - rank: [I] (leave)
  event - 97897 - rank: [I] (step)
  event - 97868 - rank: [U] (is_a)
  event - 97768 - rank: [I] (prepare)
  event - 97757 - rank: [U] (sail)
  event - 99845 - rank: [U] (say)
  event - 99774 - rank: [I] (control)
  event - 99723 - rank: [I] (control)
  event - 99691 - rank: [U] (choose)
  event - 99684 - rank: [U] (is_a)
  event - 99661 - rank: [I] (compete)
  event - 99611 - rank: [I] (say)
  event - 99346 - rank: [U] (choose)
  event - 98021 - rank: [U] (say)
  event - 97921 - rank: [I] (is_a)
  event - 97922 - rank: [I] (retire)
  event - 97914 - rank: [U] (is_a)
  event - 97805 - rank: [I] (guide)
object_of:
  event - 97928 - rank: [U] (succeed)
  event - 99718 - rank: [U] (concentrate)
  event - 99691 - rank: [U] (choose)
  event - 99604 - rank: [U] (sit)
  event - 99437 - rank: [U] (relate_)
  event - 99356 - rank: [U] (relate_)
  event - 99346 - rank: [U] (choose)
  event - 99322 - rank: [U] (hunt)
mistake_of:
  event - 99346 - rank: [U] (choose)
```

```
textrefs:
  S-3, W:11, his              [97713] Ok Anaphor
  S-3, W:4, he                [97705] Ok Anaphor
  S-3, W:1-2 = Mr. James      [97702, 97703] Ok NounPhrase
  S-3, W:2, James             [97703] Ok NounPhrase
  S-1, W:7, his               [97651] Ok Anaphor
  S-1, W:2, he'll             [97643] Ok Anaphor
  S-2, W:4, his               [97620] Ok Anaphor
  S-1, W:16, he               [97611] Ok Anaphor
  S-1, W:4, James             [97597] Ok NounPhrase
  S-1, W:3, he                [97552] Ok Anaphor
  S-1, W:1, He                [97550] Ok Anaphor
  S-7, W:14, his              [97515] Ok Anaphor
  S-7, W:5-10 = great highs , says Mr. James            (X1)
                              [97504...97510] Ok NounPhrase
  S-7, W:10, James            [97510] Ok NounPhrase
  S-3, W:22, James            [97451] Ok NounPhrase
  S-1, W:17, James            [97357] Ok NounPhrase
  S-1, W:9-10 = while he      [97348, 97349] Ok PrepPhrase
  S-1, W:10, he               [97349] Ok Anaphor
  S-1, W:3, he                [97339] Ok Anaphor
  S-3, W:1-2 = Mr. James      [97309, 97310] Ok NounPhrase
  S-3, W:2, James             [97310] Ok NounPhrase
  S-3, W:13, he               [96435] Ok Anaphor
  S-3, W:1, He                [96347] Ok Anaphor
  S-2, W:12-18 = Mr. James , 57 years old ,             (X2)
                              [96303...96312] Ok NounPhrase
  S-2, W:13-18 = James , 57 years old ,                 (X2)
                              [96304...96312] Ok NounPhrase
  S-2, W:13, James            [96304] Ok NounPhrase
  S-2, W:12-18 = Mr. James , 57 years old ,
                              [96303...96312] Ok NounPhrase
  S-2, W:13-18 = James , 57 years old ,
                              [96304...96312] Ok NounPhrase
  S-2, W:13, James            [96304] Ok NounPhrase
  S-1, W:3-4 = Mr. James      [96237, 96238] Ok UnknownClass
  S-1, W:4, James             [96238] Ok NounPhrase
```

Figure 4: Example Net Node (text version): Mr. James, in the Walk-through Article

Named Entity

The algorithm works by examining the concepts created in the SemNet following semantic analysis of the article by the core system. The algorithm selects all new nodes which have a Named Individual rank

control, which corresponds to all proper names and numerals (eg money or percentages). The family type control is also used: determined during semantics by inference, possible values include *human organisation*, *temporal quantity*, and *location*, hence these values are used to distinguish the type of concept which has been created (and subsequently the kind of markup added to the input text).

To each node will be attached zero or more Textref sequences, which must be filtered on the basis of markability (eg, only proper nouns are markable), and then overlaps removed (it is possible to find something markable inside a larger entity). These Textrefs are then combined with the entity type and finally added to the SGML tree.

Although some experiments with substantial lists of company and place names were tried, these produced little improvement and were therefore not used in the formal evaluation. However, a small amount of information on common human names was already available in the semantic network.

The walk-through article originally scored P&R = 58.16, 2P&R = 63.27 and P&2R = 53.81. LOLITA's analysis of the *numex* (money and percent) category is better than the other categories because of the relative simplicity of the grammars for these expressions. The worst category is *enamex*, where the *person* slot scores only 43% recall.

On the full formal evaluation set, the named entity task scored P&R = 67.62, 2P&R = 71.62, P&2R = 64.05 which is significantly higher than the scores for the walk-through article.

The poor Named entity scores are caused by a number of factors, the most significant of which are:

- The lack of a backup in the case of parsing failures ensures that 6% of the score is lost immediately. Although a backup strategy for named entity would be relatively simple, finding a more general strategy is difficult, and so none has been implemented.
- As previously noted, even when a parse is produced, it may have missed easy named entities because of the full parsing strategy. Further work on the pre-parser should improve this situation.
- Since the named entity task is based on the output of the full core analysis of a text, errors in phases such as semantic analysis can result in the loss of named entities already clearly identified by previous phases. Although it might help to produce named entity results directly from parser output, it would not help with other tasks and applications in which it is important that the named entities are treated correctly by the whole analysis.
- Correction of the bug which gave rise to additional quotation marks vastly improves performance on the walk-through article, but would probably have a much smaller affect on the formal evaluation results.

Co-reference

Like the Named Entity, the Coref task begins with the set of all nodes created or modified during analysis. To some of these nodes will be attached a number of Textref sequences. These are 'raw' Corefs, that is, they correspond to several pieces of text referring to the same concept. Then:

- The Textref sequences are filtered to leave only properly markable ones
- Nodes connected by 'is_a' (or, identity) links are merged, copying all of the Textrefs from the `object_` to the `subject_`, and discarding the `object_` These cases cause particular problems for the LOLITA system because the MUC-6 definition of what is co-referential differs from what the LOLITA system considers as co-referential. Some 'is_a' events need to be treated as co-referential, others do not and the distinction is often based on the type of surface form that produced the event.
- Remove the nodes which are partial heads: this prevents linking of 'cars' in the NP 'red cars and blue cars', but has to allow a link between 'sugar' in 'I like sugar manufacturers because I like sugar'. This rule was non-trivial to implement because the task definition was not clear.

- Intersections between Textref sequences of a particular concept are removed. This is a robustness measure for when the core produces duplicated textrefs.
- Concepts with less than two remaining Textref sequences are discarded. Two references are needed to form a chain.
- The remaining concepts are converted in to chains of markups, and then added to the SGML Tree.

The co-reference performance on the walk-through article was badly affected by some of the problems already mentioned. The insertion of additional quotes problem caused loss of more than half the correct co-references and more than doubled the number of incorrect ones.

The duplication of textrefs resulted in the loss of many of the co-references involving Mr. Dooner. This was because the Coref algorithm ignores Textrefs that refer to more than one concept in case they create an unintentional linking of two Textref chains (duplicated textrefs should never occur, and so any co-reference based on them is likely to be wrong).

The original coref scores for this article were 14% recall and 20% precision, however, with the correction of the bugs mentioned above, this score was improved to 45% recall and 56% precision. This is somewhat better than the formal evaluation scores which were 36% recall and 44% precision.

Template Elements

Using the general template facility, the ORGANIZATION template and the PERSON template are defined as *event-based* templates, since it is possible to find a clear underlying concept (person or organisation) from which to produce a template.

In the ORGANIZATION template the ORG_NAME and ORG_ALIAS slots are filled by using Textrefs attached to the concept which are classified as "full_propernoun", the longest one is taken as the name and the remainder as the aliases. The ORG_DESCRIPTOR slot is filled with any Textrefs that are noun phrases and not in the above slots. Since every concept in the system should be connected to some point in the SemNet hierarchy, the core inference functions are used to check if the organisation concept from which the template is derived is an instance of a company or government organisation and the results used to fill in the ORG_TYPE slot.

Similar rules are defined for the PERSON template slots. For example, 'Mr James' in the walk-through article is selected because the node 97751 belongs to the human family type, and has the Named Individual rank (see figure 4 for the semantic information about 'Mr James').

Like Named Entity, this task has a simple implementation and depends critically on the core analysis. Problems with this have been discussed in the previous sections. The original template-element scores for the walk-through article were P&R = 48.65, 2P&R = 52.02, P&2R = 45.69. These are considerably lower than the scores for the formal evaluation: P&R = 53.80 2P&R = 57.34 P&2R = 50.67

Scenario Templates

The scenario management template is defined using the *hyper-template* mechanism. A succession event is identified if the event has an action that can be generalised to a set of predefined "succession actions" (e.g. to dismiss, to fire etc.) or can be itself identified as a succession event (e.g. appointment, promotion).

The remainder of the succession event information is then established using one of two techniques. The high precision technique is used first, and looks for specific relationships between the event node and nodes that are connected to it. The type of relationship may vary depending on which of several categories the event belongs too. If the output from the core analysis is correct the related nodes will then be used to fill in the appropriate templates or slots (for example, the subject of a 'sacking' event would be used to fill the SUCCESSION_ORG slot). On it's own, this technique currently produces a low recall, largely because the core analysis may not produce exactly the correct relationships between the relevant nodes. To overcome this, a high recall technique is used when the expected relationships are not present. In this technique,

appropriate concepts that are closely related to the event are considered as candidates, and the closest concept below a threshold value is picked. For example if no company is subject of a 'sacking' event but one is an object of such an event, then this will be picked.

For the walk-through article, the succession event:

```
<SUCCESSION_EVENT-9402240133-97791> :=
    POST: "chief executive officer on July 1"
    IN_AND_OUT: <IN_AND_OUT-9402240133-198450097791>
                <IN_AND_OUT-9402240133-198740097791>
                <IN_AND_OUT-9402240133-977780097791>
    VACANCY_REASON: OTH_UNK
```

is identified on the basis of the event below, which has action *succeed* which is recognised by the fill-in rule as relevant for the management template.

```
    * event: 97791 *
 generalisation_: event - 7688 - rank: universal  (happen_)  - definition_
 subject_: Dooner - 97764 - rank: named individual - family: inanimate manmade
 action_:  succeed - 26048 -
```

Despite the lack of any analysis of the 'stepping down' sentence, the system scored 35% recall and 40% precision on the formal evaluation of the walk-through article. This compares with formal evaluation scores of 33% recall and 34% precision.

Subsequent Improvements to the Walk-through Article Performance

After a thorough analysis of the problems in the walk-through article had been carried out, several of the problems discovered were fixed and a new set of results for this article were produced. The following is a list of the changes made:

- fixed problems with reported speech
- prevented duplication of Textrefs
- allowed additional time and memory for parsing (which enables the parsing of the 'step down' sentence).
- added "to hire" as possible succession action.
- altered a heuristic which caused categorisation problems for sentence-initial words, which was originally too strong: it had been reducing morphological possibilities to a single item.
- fixed a bug in the semantics which caused a poor analysis of some verbs which were keywords for the scenario template

The resulting scores are shown below. There are significant increases in all the tasks other than Template Element.

```
CO   Recall: 60/132 = 0.45   Precision: 60/108 = 0.56
  F-MEASURES          P&R      2P&R      P&2R
NE                  86.08     87.4     84.79
TE                  55.70     56.99     54.46
ST                  55.86     53.82     58.05
```

CONCLUSIONS

The LOLITA system was entered in MUC-6 mainly because of the importance of evaluation. There is no clear methodology for evaluation in the NLP field; however, a well-established and well-known event such as MUC presents an excellent challenge and provides important resources for evaluation. We wanted to see if our system could be adapted to perform the MUC tasks, and then to see how well it could do them. Clearly our general purpose 'deep analysis' approach to the tasks did not produce scores that compare well with the best systems, however there are some general reasons why we believe this is the case. Firstly, the use of a system which aims to be general purpose rather than generic, means that it is not possible to start from a 'clean slate' and populate the system with a set of rules ideally suited to just the MUC evaluation. Any modifications to rule bases in the system's core must be carried out with a careful view to their effect on all aspects of the core. Given that the MUC tasks only test certain aspects of the core system, this means that much effort is expended on issues that will not affect the MUC performance. Secondly, the nature of the MUC-6 tasks is such that only a small percentage of the marks are available for 'deep' analysis and so such an analysis is counter productive unless it achieves an extremely high level of robustness. We are working towards such a level of robustness, but our MUC-6 results make it clear that we are not there yet.

As well as providing impetus to develop the core system, the experience has taught us much about testing and evaluation. This will help in subsequent development. Code-wise, several major extensions have been added, and much existing code has been improved. Very little of this work is MUC-specific, so the amount of reuse is high. Evaluation-wise, we have a set of measures with which to evaluate at least some aspects of our progress.

We do not see the scores as a refutation of our approach. As is to be expected in a system of LOLITA's size and complexity, we see the effects of several small bugs in the analysis which obscure the potential scores: witness our recent improvement in the walk-through article. It is clear that increasing robustness, for example by providing backup strategies when the main analysis fails, is a good idea. We also plan to improve our parsing and grammar techniques. During development, we have seen several examples of good scores being obtained when the system works to its full potential, and we are much encouraged by it.

In summary, we are pleased with our first participation in MUC. Not only have we successfully implemented all four tasks on our first attempt at MUC, but we managed to produce a deep analysis of a good part of the text in the formal evaluation set. Despite the hard work, MUC has been an extremely useful and enjoyable experience, and we look forward to MUC-7.

REFERENCES

[1] Eric Brill. Rule-based tagger, 1995. contact: brill@cs.jhu.edu.

[2] Glasgow. The university of glasgow functional programming group, 1995. URL http://www.dcs.glasgow.ac.uk/fp/.

[3] P Hudak, S Peyton Jones, and P Wadler. Report on the functional programming language haskell, version 1.2, March 1992. URL ftp://ftp.dcs.gla.ac.uk/pub/haskell/report.

[4] Derek Long and Roberto Garigliano. *Reasoning by Analogy And Causality: A Model and Application.* Ellis Horwood, 1994.

[5] George Miller. Wordnet: An online lexical database. *International Journal of Lexicography*, 3(4), 1990.

[6] Mark H. Smith. *Natural Language Generation in the LOLITA System: An Engineering Approach.* PhD thesis, Department of Computer Science, University of Durham, 1995. submitted for the degree of Ph.D.

[7] M. Tomita. *Efficient Parsing of NL: A Fast Algorithm for Practical Systems.* Kluwer Academic Publishers, Boston, Ma, 1986.

[8] Yorick Wilks. Preference semantics. In Edward L. (ed.) Keenan, editor, *Formal Semantics of Natural Language*, pages 329–348. Cambridge University Press, 1975.