

Number 373



UNIVERSITY OF  
CAMBRIDGE

Computer Laboratory

## Natural-language processing and requirements specifications

Benjamín Macías, Stephen G. Pulman

July 1995

15 JJ Thomson Avenue  
Cambridge CB3 0FD  
United Kingdom  
phone +44 1223 763500  
<https://www.cl.cam.ac.uk/>

© 1995 Benjamín Macías, Stephen G. Pulman

Technical reports published by the University of Cambridge  
Computer Laboratory are freely available via the Internet:

*<https://www.cl.cam.ac.uk/techreports/>*

ISSN 1476-2986

DOI *<https://doi.org/10.48456/tr-373>*

# Natural-Language Processing and Requirements Specifications

Benjamín Macías  
Stephen G. Pulman

March 1995

This document reports on our participation in the MORSE (“A Method for Object Re-use in Safety-critical Environments”) project. Our brief in the project was to investigate the role that natural-language processing (NLP) techniques can play in improving any of the aspects linking natural-language requirements specifications and formal specifications. The contents are as follows. We begin with a brief introduction to NLP in the context of requirements tasks, followed by an examination of some strategies to control the form of requirements specifications. We continue by describing an interface designed to correct some of the problems with known methods to control specifications, while employing current NLP to maximum advantage. We then show how to build a natural-language interface to a formal specification, and some aspects of the problem of paraphrasing formal expressions. We finish with the conclusions reached at the end of our participation in the project.

The work reported here was sponsored by Engineering and Physical Sciences Research Council and the U. K. Department of Trade and Industry under the Department’s Safety Critical Systems Advanced Technology Programme (SCSATP), project no. IED4/1/9001 (MORSE).



# Contents

<b>Preface</b>	<b>4</b>
<b>1 A Brief Introduction to Natural-Language Processing</b>	<b>6</b>
1.1 Natural-language Processing Systems . . . . .	6
1.2 Structure of a NLP System . . . . .	7
1.2.1 Morphology . . . . .	8
1.2.2 Syntax . . . . .	9
1.2.3 Semantics . . . . .	12
1.2.4 Reference Resolution . . . . .	13
1.2.5 Reference Resolution Problems . . . . .	14
1.2.6 Processes of Reference Resolution . . . . .	16
1.3 Conclusion . . . . .	19
<b>2 Controlling the Form of Requirement Specifications</b>	<b>20</b>
2.1 Introduction . . . . .	20
2.2 Some Features of Natural-Language Specifications . . . . .	20
2.2.1 Sentence Length . . . . .	21
2.2.2 Punctuation . . . . .	21
2.2.3 Presentational Units . . . . .	22
2.3 Use of a controlled vocabulary . . . . .	23
2.4 Enhancing a simple style of writing . . . . .	23
2.5 Keeping track of the information in the specification . . . . .	24
2.6 Designing an English-like specification language . . . . .	26
2.7 Evaluation of these proposals . . . . .	28
2.8 Conclusions . . . . .	29
<b>3 A Window-Based Interface for Specifications in Natural-Language</b>	<b>30</b>
3.1 Introduction . . . . .	30
3.2 An Example . . . . .	31
3.3 Lists . . . . .	33
3.4 Defining local identifiers . . . . .	34
3.5 Storing, modifying, and reusing a specification . . . . .	36
3.6 Ambiguous statements and paraphrasing . . . . .	38
3.7 Evaluation of the system . . . . .	39
3.8 Further and Related Work . . . . .	41

3.9	Conclusion . . . . .	42
<b>4</b>	<b>Natural-Language Interfaces to Formal Specifications</b>	<b>43</b>
4.1	Introduction . . . . .	43
4.2	Processing natural-language queries . . . . .	44
4.3	Formal specification of a valve . . . . .	45
4.3.1	Processing of queries in English . . . . .	46
4.3.2	Representing the formal specification . . . . .	47
4.3.3	Direct questions about the specification . . . . .	47
4.3.4	Question about properties of the domain . . . . .	50
4.3.5	Questions about functions and their properties . . . . .	51
4.4	Conclusion . . . . .	52
4.5	Appendix 1: Interpretation of Queries . . . . .	54
<b>5</b>	<b>Generating English-Like Paraphrases of Formal Expressions</b>	<b>56</b>
5.1	Introduction . . . . .	56
5.1.1	Logical Expressions . . . . .	56
5.1.2	Set-theoretic expressions . . . . .	63
5.2	Summary . . . . .	67
<b>6</b>	<b>Conclusions</b>	<b>68</b>
	<b>Bibliography</b>	<b>70</b>

# Preface

The first stage in the development of a system, be it computer software or a large engineering project, begins with the writing of a *requirements specification* in English, or some other natural language. From this description, a first high-level formal specification is then developed, and successively refined until a final description of the system is used to build it ([42]).

But natural languages do not have the properties that we associate with those languages used for formal specification tasks: that they are unambiguous, precise, and have well-defined semantics. In other words, many formal specifications can be consistently developed from one natural-language specification. Thus, using a specification written in English without a method to alleviate its deficiencies puts into question the entire process of developing a system from a specification.

Another area where formal and natural-language specifications interact is in the derivation of natural-language information from formal specifications. For example, if a formal specification eventually results in a system, that system will need to be documented for use and maintenance. Using the formal specification can make the task of producing such documents much easier (especially when there is more than one target language), as well as introducing some method in their production (useful in safety-sensitive applications).

This document reports on our participation in the MORSE (“A Method for Object Re-use in Safety-critical Environments”) project. Our specific brief was to investigate the role that natural-language processing techniques can play in improving any of the aspects linking natural-language requirements specifications and formal specifications. This work reflects the MORSE project interest in investigating domain-independent techniques, and in approaches that can realistically be used given to-day technology.

We want to thank our partners for their collaboration and the many fruitful exchanges during the project: Mark Christian, Hamid Lesan, Derek Mannering, and Steven Hughes (Lloyd’s Register of Shipping), Paul Grace and Chris Hall (Ultra Electronics, née Dowty Controls), Roger Banks, Alan Cuff and Simon Soper (BICC Transmittion, BICC plc), Paul Collinson, Richard Fink and Susan Oppert (West Middlesex Hospital), Benita Hall (British Aerospace Airbus Ltd), and Stan Price (Price Project Services Ltd), the DTI monitoring officer.

We thank Roger Banks and Alan Cuff for making available to us case studies of natural-language specifications.

Most of the material in this report originates in external deliverables of the MORSE project:

Macias, Benjamin and Stephen G. Pulman. 1993. Natural Language Processing and Formal Specifications (D12). Doc. Id. MORSE/CU/BM/1/v2.

Macias, Benjamin. 1993. Use of CLARE to analyse specifications in English-like languages (D24). Doc. Id. MORSE/CU/BM/4/V2.

Macias, Benjamin. 1994. Use of CLE/CLARE in semi-automatic translation of English and English-like specifications (D30). Doc. Id. MORSE/CU/BM/6/V2.

Macias, Benjamin. 1994. A Method to Control the Drafting of Requirements Specifications in Natural Language. Doc. Id. MORSE/CU/BM/7/V1.

Macias, Benjamin. 1995. Natural-Language Processing and Requirements Specification (D40). Doc. Id. MORSE/CU/BM/8/V1.

and the following publications:

Macias, Benjamin and Stephen G. Pulman. 1995. A Method for Controlling the Production of Specifications in Natural Language. To appear (fall 1995), Computer Journal.

Macias, Benjamin and Stephen G. Pulman. 1993. Natural Language Processing for Requirements Specifications. In Redmill F. and T. Anderson (eds), Safety-Critical Systems. Chapman and Hall, London.

Macias, Benjamin. 1994. Natural-Language Interfaces to Formal Specifications. Submitted.



# Chapter 1

## A Brief Introduction to Natural-Language Processing

This chapter summarises the basic ideas behind some general techniques of natural-language processing (NLP henceforth), and present examples of their application to actual sentences contained in some example specifications provided by the MORSE participants. The introductory part can only cover the bare essentials behind NLP techniques; interested readers should consult the relevant chapters of Charniak and McDermott [10], or for a more specialised account, Gazdar and Mellish [19].

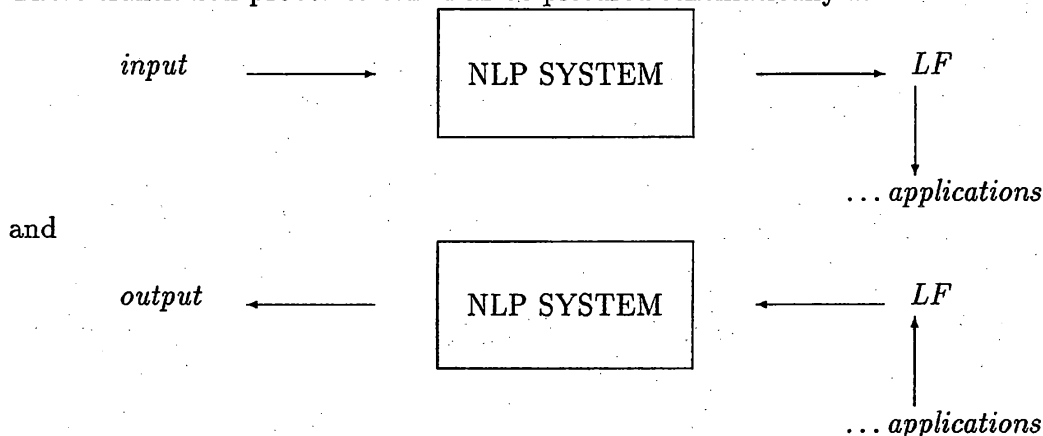
CLARE, the specific natural-language system we will be employing, is a natural-language processing system designed to provide a computer interface capable of domain modelling reasoning. At its core, CLARE contains a domain-independent system known as the Core Language Engine (CLE) to carry out general linguistic processing of English sentences. Both CLARE and the CLE have been applied to a variety of tasks, such as building an experimental translation system for English-Swedish, and creating an interface to a database. We do not have space here to give a description of CLARE to any degree of detail; readers may refer to [3] and [4] for more information and further references.

### 1.1 Natural-language Processing Systems

Natural-language processing (NLP) systems are, described very generally, computer programs that use as data sentences in a natural language such as English. Typical application examples include a translator of sentences from one language into another, a generator of messages in English, or a program that allows a user to query a knowledge-based system such as a database, in English, instead of using a specialised query language.

NLP is conventionally divided into two main areas: *understanding*, or the problem of how to extract the meaning of natural-language sentences, and *generation*, the study of how to produce sentences conveying a specific message. In other words, the core task of a NLP system is thus either to translate an input English sentence into its “meaning”, or to take a “message” and derive an English expression from

it. In this review, we will identify such “meanings” or “messages” associated with the content of an English expression with the uniform notion of a *logical form* (LF). These translation processes can then be pictured schematically as:



CLARE is designed to perform these tasks, and uses for both of them the same sets of rules. To constrain our review of the basic NLP techniques to a manageable size, we will limit ourselves to describing the process of extracting the meaning of English sentences. Although the task of generation involves some problems of its own, the reader can think of it in very general terms as being similar to understanding, but applying the rules involved in extracting a logical form in inverse order.

## 1.2 Structure of a NLP System

A common approach to the design of a NLP system is to separate the task of understanding into a series of steps, each one associated with a different aspect of the problem. This division corresponds in the main to the distinction found in linguistic theory to distinguish between different levels of language, such as morphology, syntax, and semantics.

This methodological division naturally results in computer systems that consist of a number of semi-independent modules, each containing the knowledge that corresponds to a different aspect of language. The input sentence is processed by each of these modules sequentially, so that the output of one module corresponds to the input of the next one, with a LF as the final result. A NLP system must also contain a lexicon with a list of English words and their meanings.

A system such as CLARE has the following modules, listed in order of application:<sup>1</sup>

1. morphology: containing rules of word formation.
2. syntax: rules for combining parts of speech into constituents.
3. semantics: the meaning of the parts of speech and their combination.

<sup>1</sup>We are simplifying considerably in the interest of legibility; readers are encouraged to consult the source documents.

4. contextual reasoning: context dependent aspects of interpretation like reference of pronouns, resolution of ellipsis, and choosing between ambiguous interpretations.

There are various motivations for this approach. Sequential processing is —from a computational point of view— the easiest way to understand the behaviour of what are often complex systems; with this organisation, the user can sometimes help to guide the system to process a sentence. A modular organisation has also well-known advantages from a software-engineering point of view: modular systems are easier to program and test, and the encapsulation of application-specific components encourages portability. As we will try to convey below, a core NLP system capable only of strictly linguistic analysis must encode nevertheless a surprisingly large amount of knowledge. By making this part of the overall task separable, we can use the same modules in quite different domains. After all, the rules of English apply to speech on gas installations as much as it does to utterances on the properties of landing gears. Finally, by maintaining separate modules for the each linguistic-specific aspect of the process, we can incorporate new results from linguistic research in a more direct manner.

We now give a brief descriptions of the main modules of a NLP system, indicating their task and some of the computational problems at each step.

### 1.2.1 Morphology

In order to extract the meaning of a sentence, we first need to know how to categorise each of the words of the input sentence, i.e. assign to each word a label from the set { nouns, verbs, adjectives, particles, punctuation marks, ... } For example, if we have a sentence like:<sup>2</sup>

Authorisation is required before the transfer of propane from the trucks  
can commence

we will have to use the information that *transfer* is a noun to conclude that (and eventually comprehend) that *the transfer of propane* is a noun phrase. Clearly, the first element required is a lexicon containing information about English words and their meanings. However, more is needed than just a very long list of words, namely the ability to match derived words with their base forms, deducing at the same time the information conveyed; in the sentence above, for example:

- The word *trucks* is a noun derived from the noun *truck*, and the ending *+s* is used in English to indicate plurality. Hence the phrase *the trucks* is about a set of more than one truck.
- The word *required* is the participle verbal form of the verb *to require*. Participles in English are formed by adding the ending *+ed* to the base verb form, and this word has the simplified spelling *required* (and not *requireed*).

---

<sup>2</sup>Unless otherwise stated, our examples will either be direct quotes from [5], or possible variations of them in the same context.

- The word *authorisation* is a noun derived from the verb *to authorise*, composed with the nominal suffix *+ation*

Any system designed for general application must contain many such rules. CLARE has dozens of them;<sup>3</sup> an example rule is the one for third-person singular verbs, present tense:

```
morph(v_v_es_Thirdps,
  [v: [agr=(sing/\3), vform=fin, mhdfl=H,
      gaps=Gaps, lexform=L, modifiable=Mo,
      mainv=M, vpellipsis=V,
      subcat=S, subjform=Su, paradigm=Par],
    v: [agr=_, vform=base, mainv=M, mhdfl=H, vpellipsis=V,
      gaps=Gaps, lexform=L, modifiable=Mo,
      subcat=S, subjform=Su, paradigm=Par],
    es]).
```

This rule forms the third person singular form (*agr=(sing/\3)*) of a verb (e.g. 'authorises') by combining its base form (*vform=base*) with a suffix '-es'. The resulting verb inherits most of its other properties from the base form. 'Segmentation' rules cope with any spelling changes that may be necessary: 'authorise+es'='authorises', although 'finish+es'='finishes'. Describing these morphological combinations by rule reduces the size of the lexicon, which would otherwise have to contain all the variant forms of a word. This is clumsy for English, and impossible for languages like German, which have a much richer system of morphology.

The task of categorisation is complicated by the fact that, in the absence of a suitable context, each word considered independently can have several possible interpretations:

- *transfer* can be either a verb (infinitive or imperative) or a noun (singular)
- *can*: might be a noun (singular) or a verb (either singular or plural)

Thus, the output of this process is not usually unique, but equal to the product of the number of labels of each word. In the sentence above, assuming that "transfer" has three possible labellings, and "can" two, the morphological analyser will produce six different hypotheses of how to label each word in the sentence. This information will be taken by the syntactic component as input.

### 1.2.2 Syntax

Unfortunately, we cannot derive the meaning of a sentence by considering only the meaning of each of its words. We must also take into account the relation that words have to each other in a sentence; that is, we must find its *structure*. For example, we can say that a sentence such as:

<sup>3</sup>These are actually divided in CLARE into segmentation and morpho-syntactic rules. See [3], p.91-92.

the operator must authorise every transfer

has schematically a structure:

[*S* [*NP*the operator] [*VP*must [*VP*authorise [*NP*every transfer]]]]

Syntactic rules code exactly the kind of structure internal to English sentences; the set of all rules for a language is called a *grammar*. Alternatively, we will say that *the grammar generates a language*.<sup>4</sup> Let us give an example. A very small grammar capable of generating —among others— the sentence given above will contain at least the following rules:

- [1] sentence = noun\_phrase + verb\_phrase
- [2] noun\_phrase = det + noun
- [3] verb\_phrase = verb + noun\_phrase
- [4] verb\_phrase = 'must' + verb\_phrase
- [5] det = 'the'
- [6] det = 'every'
- [7] noun = 'operator'
- [8] noun = 'transfer'
- [9] verb = 'authorise'
- ...

where the following conventions apply: each line corresponds to “an equation” defining one valid structure of the language.<sup>5</sup> Each equation begins with a number in square brackets; the only purpose of this number is to give a name to the rule. The left side of the equation (what is to the left of the ‘=’ sign) contains the name of what we are defining; the right side being its definition. The ‘+’ sign is used as the “immediately followed by” relation. Finally, those elements in single quotation marks stand for the *terminals* of the language (e.g. English words); those without quotes correspond to *non-terminals*, representing the structures of English.

An example can perhaps clarify the notation. The rule 4 above can be paraphrased as: “a structure consisting of the word ‘must’ immediately followed by a *verb\_phrase* structure is a valid *verb\_phrase* structure of English.

Continuing with the example, we can apply these rules to show that the sentence above belongs to the language defined by the grammar, or in other words, that it is grammatical. We will do so through a process of *parsing*, which consists of showing that the definition of *sentence* can be used to produce the target sentence by repeatedly choosing a non-terminal and replacing it by one of its definitions. For the sentence above, this can be shown as follows:

---

<sup>4</sup>The grammars we will talk about in this section are called *context-free* because of the form of the rules; those used by CLARE are somewhat more powerful.

<sup>5</sup>Strictly speaking, these rules are not equations in that a non-terminal often has more than one definition, e.g. there are several rules with the same non-terminal and different right-hand sides. For example, rules 5 and 6 define different ways of rewriting a determiner. A true definition of a determiner would be:

det = 'the' ∨ det = 'every' ∨ det = 'a' ∨ ...

sentence  
 (rule 1)  
 noun\_phrase + verb\_phrase  
 (rule 2)  
 det + noun + verb\_phrase  
 (rule 5)  
 'the' + noun + verb\_phrase  
 (rule 7)<sup>6</sup>  
 'the operator' + verb\_phrase  
 (rule 4)  
 'the operator must' + verb\_phrase  
 (rule 3)  
 'the operator must' + verb + noun\_phrase  
 (rule 9)  
 'the operator must authorise' + noun\_phrase  
 (rule 2)  
 'the operator must authorise' + det + noun  
 (rule 6)  
 'the operator must authorise every' + noun  
 (rule 8)  
 'the operator must authorise every transfer'

As a side effect, we have also obtained the structure of the sentence. In essence, this is done by building a representation that mirrors the application of each rule in the derivation process. In this case CLARE will produce, in a different but equivalent manner, the following analysis (considerably simplified):

```

[s,
  [np, [det,the],
        [noun,operator]],
  [vp,
    [verb,must],
    [vp,
      [verb,authorise],
      [np,
        [det,every],
        [noun,transfer]]]]]
  
```

Although this might not be immediately apparent, the structure underlying sentences approximately corresponds to the one that we need to find the meaning of a sentence. This process is thus a necessary step to derive the LF of a sentence; the output of the syntactic component is the set of all the possible analyses of the input sentence.

---

<sup>6</sup>We will slightly simplify the notation: when two terminal symbols are found together, we will consider them part of the same string, and omit the intervening + sign. Otherwise, we would have written the following line as:

'the' + 'operator' + verb\_phrase

CLARE contains a very large syntactic component, with a coverage that includes many of the most common constructions of English. In addition, CLARE contains some useful specialised capabilities, such as an extensive treatment of dates, and the possibility to handle idioms. Nevertheless, although linguists have produced grammars with hundreds of rules like the ones above, exhaustive coverage has proved elusive, limiting current applications to the most ordinary constructions of a language. These limitations must be borne in mind when devising applications of NLP techniques.

The parsing process allows the elimination of some of the combinations generated by the morphological analysis (e.g. by identifying *every transfer* as a noun phrase, the possibility of *transfer* being a verb in the imperative will be ruled out), but it is also the case that new alternatives are often generated at this stage. This is because many constructions have more than one parse, usually corresponding to different possible readings. For instance, a phrase such as *the detection of a gas leak in the reception and storing area* can either mean detection in:

{the reception} and {storing area},  
or  
the {reception and storing} area

Similarly to the process of assigning lexical categories to words, it often happens that many possibilities usually not envisaged by a human reader are generated by a machine analysis, revealing ambiguity and vagueness in seemingly clear sentences. Unfortunately, languages are structured in such a way that relatively small sentences can produce dozens or even hundreds of analyses. This creates the problem for a NLP system of how to choose the most appropriate analysis (something which people usually do without conscious effort). This is one of the most difficult problems in NLP research, and we return to some of its implications below.

### 1.2.3 Semantics

Although this might not be immediately apparent, the syntactic structure underlying sentences is necessary in order to work out the compositional semantics of a sentence. So the next stage of processing takes as input the syntactic analysis or analyses produced by the parser, and together with the lexical information contributed by each word, derives an initial logical form (LF) of the sentence's meaning. For instance, if the input is a simple affirmative English sentence, this process will commonly extract a main actor from the subject noun phrase, a main predicate from the verb or verbs, and a list of affected objects from the complement phrases of the main verb. This initial step will obtain the entities involved in the sentence, but leaving the precise nature of their relation underspecified.

Using again the sentence *the operator must authorise every transfer*, CLARE will derive an initial LF saying approximately that "there is relation of authorisation holding (in some fashion) between one or more events, an operator, and every transfer." This corresponds to (in a somewhat simplified form):

```

authorise(term(exists,C,event(C,pres,must)),
          term(the,A,operator(A)),
          term(every,B,transfer(B)))

```

At this stage (which we call “quasi-logical form” or QLF) several pieces of information necessary to interpret the sentence appropriately have not yet been determined. Usually, to spell out the exact relationship implied by the sentence, one must consult the context and various types of reasoning. Such processes will result in more and more detailed logical forms.

One example of this refinement process is the introduction of quantifiers (section 1.2.5). A full first-order formula containing the quantifier information lacking in the previous formula would read (again, leaving many details aside): “there exists an operator  $A$ , such that for every transfer  $B$ , there exists an authorisation event  $C$  of  $B$  by  $A$ ”. A formula reflecting this would have the general form:

```

exists([A],
       and(forall([B],
                  impl(transfer(B),
                       exists([C],
                              and(authorisation(C,A,B),
                                   event(C))))),
          operator(A)))

```

The semantic component must contain rules to extract a QLF for each of the many variations allowed by the syntax of English to convey meanings: passives, negatives, interrogatives, their combinations, and so on. Most of these QLFs will contain constructs that need further resolution. The “reference resolution” phase of interpretation tries to produce candidates to “fill in” these constructs so as to arrive at a complete proposition: the message conveyed by that utterance of that sentence in that particular context.

We will now describe various aspects of this process.

## 1.2.4 Reference Resolution

As we explained in the previous section, there are a number of pieces of information that will still need to be fully specified to obtain a fully-resolved logical form of a sentence. In this section we will first describe a number of problems to be solved in order to determine the LF, and a basic strategy to do so afterwards.

The reader must bear in mind that, in contrast to earlier stages of sentence interpretation, processes of contextual reasoning rarely have well-defined solutions, as they entail the search for an interpretation that is *relatively better* than the others in the context of the intended application. They will often involve complex reasoning — at best computationally expensive, and at worst impossible to automate — and require contextually-dependent information.<sup>7</sup>

<sup>7</sup>In the next paragraphs, we will introduce some of the main aspects of pragmatic process, but by not means all. Interested readers can consult the general references, or chapter 12 of [2].



## 1.2.5 Reference Resolution Problems

### Quantifier Scope

When a sentence contains determiners such as *a*, *the*, and *every*, its logical translation will contain quantifier phrases, which must specify their relative scopes. For example, the sentence:

every truck is controlled by the operator

can be interpreted in two different ways; in other words, can correspond to two different logical forms. One can read the sentence as saying that, for a given operator, all the trucks are under his control. Alternatively, it can also mean that for each truck there is one operator, which might or might not be the same one. In terms of the produced LF, this corresponds to choosing between:

```
exists([A],
      and(forall([B],
                impl(truck(B),
                    exists([C], and(control(C,A,B), event(C))))),
          operator(A)))
```

meaning: “there is an operator A such that for every truck B, it implies that there exists an event C of control, where A is the controller of B”, and:

```
forall([A],
      impl(truck(A),
          exists([B,C],
                and(and(control(C,B,A), event(C)),
                    operator(B))))))
```

or: “for every truck A, it implies that there exist an operator B and an event C of control, such that B is controlling A.”

Spelling out these alternatives and choosing between them requires complex reasoning. There are many factors involved in preferring one reading over the alternatives, such as the specific lexical items contained in the sentence, knowledge of the ongoing discourse, and general plausibility. The following examples of similar sentences have different preferences:

- *Every truck is controlled by an operator*  
(either the same operator, or a different one for each truck)
- *Every truck is driven by an operator*  
(more likely to be a different one for each truck)
- *Every truck is controlled by a central operator*  
(more likely to be a single operator controlling all the trucks)

## Reference Determination

Another problem is the one of *reference determination*, which consists of fully identifying persons and objects being mentioned indirectly in a piece of discourse.

A common strategy for referring to entities in a discourse is through the use of pronouns. Although in some cases syntactic constraints can be used to interpret them unequivocally,<sup>8</sup> it is often necessary to consider the situation in which the pronoun is used. Take the following sentence:

*the operator* will be then notified locally on indicator YL101 that *he* can proceed with the transfer ...

In order to obtain a full LF of the sentence, the initial task is to link *the operator* to the person specifically addressed (say, John Smith, or "whoever is manning the position at the time"). Then, the personal pronoun *he* must be paired up with whoever is ultimately being referred to. There are several proposals of how to do this, but basically they all consist of two processes: first, collect all the possible referents in the discourse (in this case, all male animate subjects), and then rank them according to some measure of plausibility.

In some cases, this can be a very difficult problem to automate, as it requires some understanding of what is being said. Consider another example:

this last event will automatically stop the transfer whilst *the others* require specific action.

Here, context of the sentence itself reveals some possible referents for "the others": other events, or perhaps other operators. It could even refer to some other entity in the surrounding context: a transfer, a computer, a button, or an alarm. A correct answer needs both a large amount of knowledge about the domain and the capacity to reason over it.

A variation on this problem arises when processing definite noun phrases. In a sentence such as:

The outstation will open *the input valves* of the selected reservoir for the chosen liquid

the noun phrase *the input valves* is referring to some specific valves that have presumably been mentioned before. Here, the extra information contributed by the restriction of *the selected reservoir* must be used to filter down the candidates to those who fulfill this characteristic.

---

<sup>8</sup>In sentences such as *the operator will start the procedure himself*, the reflexive pronoun *himself* can only refer to the operator.

## Anaphora and Ellipsis

Common discourse avoids constant repetition of entities by leaving them implicit. There are grammatical constructions that achieve this effect, such as the passive:

the outstation will open the input valves of the selected reservoir and  
the valve operation will be confirmed

does not mention who or what will confirm the valve operation. A similar phenomenon is the use of *ellipsis*, also leaving implicit some parts of speech. Imagine the following fragment in a sequence of instructions:

The outstation will open the input valves.  
And also check their status.

In this case, the subject of the second sentence has been omitted.

## Vague Relations

Another source of complexity is the use of *vague relations*. A relation is vague when a phrase such as *the truck pump connection* is used: a relation between a truck, a pump, and a connection is established, but it is not explicit which one. Without context or specific knowledge of the domain, we cannot tell whether the phrase might mean “the connection between the truck and the pump,” or “the connection on the pump on the truck”, or even “the connection on some other entity for the truck’s pump.”

Similar problems occur when prepositionally modified noun phrases are used in a vague fashion. Examples of these could be the phrases:

pump for the truck  
plug for the pump

Again, the preposition *for* could be equivalent to a number of specific relations in the semantics definitions.

### 1.2.6 Processes of Reference Resolution

At this point, we hope that it is clear to the reader what the main problem for a computer program is when dealing with English expressions: every level of language, from words to sentence meanings, has a considerable degree of ambiguity and vagueness. A NLP system that considers all the possible interpretations of a sentence will end up producing tens or hundreds of them, and most will not be right (in the sense of not corresponding to what the writer of the sentence had in mind).

This is a problem without a simple solution. Ultimately, it presupposes a complete knowledge and understanding of a domain, the comprehension of the participants’ intentions, and so on. Representing the kind of information that is needed for this is extremely complicated and presupposes a solution to many of the classical

problems of Artificial Intelligence. For this reason, systems that carry out this kind of reasoning currently only do so in the context of very limited domains, domains which are restricted enough for it to be realistic to build a complete axiomatisation of them. It is, at the current state of the art, not possible to build general purpose reference resolution mechanisms.

Nevertheless, heuristic techniques can be used to control the combinatorial explosion and narrow down the search for an interpretation. In general terms, the idea is to define a mechanism that rates the alternatives at each stage according to their plausibility, and instructing the system to pursue only those analyses whose rank is above a certain threshold.<sup>9</sup> By adopting these heuristics, we can indicate which readings to prefer.

These heuristics can define general preferences, or others of a more limited nature according to the domain of application. The first kind uses general procedures that apply to all applications, and do not presuppose any special knowledge. For example, a useful syntactic heuristic is always to assume that phrases are complements rather than adjuncts; in the example:

The transfer can be stopped by the operator

we prefer a reading in which *the operator* is interpreted as the actor of the action of stopping, and not as some sort of modifier (as in *the transfer can be stopped if the switch by the operator is activated*). Another heuristic that frequently helps is to produce syntactic analyses with phrases attached to the nearest constituent. In:

Power will be applied to the local plug for the truck's pump.

the phrase *for the truck's pump* will be considered a modifier of *the local plug*, and not a global modifier (as in *power will be applied to the local plug for the rest of the day*).

More domain-specific heuristics use general methods, but might depend on some knowledge that has been specifically crafted for the application in mind. One instance of them is the use of sortal restrictions. Informally, *sorts* can be pictured as a division of the elements in the discourse into classes characterising their properties. For example, an operator is (in order of increasing generality) a male, human, animate, physically-real entity, whereas computers and trucks can be regarded as inanimate, physically-real entities. In certain applications, it could make sense to discriminate among trucks and computers by assigning to the latter the capacity of acting and taking decisions. With this, we could analyse *by the computer* as an agent-bearing phrase, and *by the truck* as some spatial modifier in:

The reservoir will be closed by the computer

The pump can be activated by the truck

---

<sup>9</sup>CLARE contains such a mechanism to handle preferential readings, and implements a number of preferences. See [4], ch 5.

Finally, we can use domain-specific criteria to indicate *ad hoc* preferences that always apply in the application language. If for instance we use an artificial language with a closed vocabulary, we can make constructions with multiple readings unambiguous by fixing a specific analysis. For example, specifications commonly use noun noun compounds such as:<sup>10</sup>

reservoir output valve  
valve open operation  
truck earth connection XS101  
local start button HS107A  
truck transfer pump plug

In most such situations, the knowledge that a *truck transfer pump plug* probably means a *truck*  $\{\{transfer\ pump\} plug\}$  and not a  $\{truck\ transfer\} \{pump\ plug\}$  (or any of the other many possibilities) can be of considerable help. In other words, by pre-defining the meaning of specific noun-noun constructions for specific domains, the problem can be considerably alleviated.

The use of specific knowledge to handle ambiguity must always be considered carefully. Although a certain amount of tailoring of a general system can always be expected for each application, one must proceed carefully to maintain the generality of a system. Put briefly: without general methods, the successive specialisation of a NLP system will make it less and less likely it will ever be used in other domains. As is the case in other disciplines, the mark of a good technique is that it is applicable to a variety of situations.

---

<sup>10</sup>To see their multiple ambiguity, compare a  $\{white\} \{bread\ container\}$  with a  $\{white\ bread\} \{container\}$ .

## 1.3 Conclusion

This chapter has introduced the basic techniques used in NLP to analyse sentences in a natural language. These techniques have the complementary purposes of extracting the intended meaning from English sentences, and of producing English sentences corresponding to some message. The main problem for analysis is that natural languages have a pervasive degree of ambiguity and vagueness at all levels.

Because people are superbly efficient at choosing the intended meaning among the many possible, this basic feature of language is not always noticed. Nevertheless, misunderstandings are not uncommon, even when both reader and writer share the same intuitions about the domain of discourse. In other words, if two readings of a given statement are almost equally plausible, then the reader may choose a different interpretation than was intended by the writer. Indeed, there is some experimental evidence showing that, although the use of formal methods can be very effective in the production of software, the misinterpretation of the natural language input leaves residual errors in the production of a system ([7]).

The implications for domains in which security is a fundamental consideration are obvious: an initial requirements specification written in English may result, if differently understood by the specifier and the implementer, in a system whose performance deviates from the original intentions of the specifier.

In terms of the inherent difficulty of the various applications we have sketched, in general we started with the simplest and worked up to the most complex. There are two types of complexity involved: the degree of linguistic processing required by an application, and the degree of domain modelling required. Linguistic processing gets more complex and non-deterministic as we go from morphology through to semantics. The precise resolution of linguistic properties like reference or disambiguation demands more complex and labour-intensive applications, as it depends on non-linguistic factors such as those that would be captured in a domain model. This means that, at least in the short term, it is most worth exploring those applications that involve general purpose, domain independent, linguistic processing.

## Chapter 2

# Controlling the Form of Requirement Specifications

### 2.1 Introduction

In the previous chapter we stressed that ambiguity is a pervasive, if often undetected property of natural languages. The purpose of this chapter is to review some basic techniques advanced to control the ambiguity, vagueness, and lack of clarity of specifications in natural language. These techniques are not usually considered to fall within NLP proper, but they could plausibly be used to produce better specifications.<sup>1</sup> The review will also help to pinpoint some of the deficiencies of so-called “claims languages” for specification processes ([12]).

The reader must keep in mind that we are restricting ourselves to tasks that we believe are realistically achievable with current technology, and that we are, with the exception of one or two of the later suggestions, primarily emphasising the role of domain-independent NLP techniques. As we remarked in the previous chapter, more ambitious applications of NLP in this area may be possible, but they would require sophisticated domain modelling which might be too labour-intensive to be profitable.

Before we turn to the review, we start with a brief summary of the characteristics of the source natural-language specifications at our disposal.

### 2.2 Some Features of Natural-Language Specifications

We begin by describing some basic features of the two case studies. As is to be expected from the discussion in the first chapter, we found many examples of possible ambiguities in the source documents due to syntactic, semantic, and pragmatic factors. In this respect, the documents studied do not differ from many others

---

<sup>1</sup>We say “plausibly” because, to our knowledge, none of the approaches reviewed in this chapter have ever been tried in a real setting.

studied previously in the computational-linguistic literature devoted to properties of manuals and other instructional texts.

A number of papers have been devoted specifically to the linguistic issues that manuals and other instructional texts present. ([20]) carried out a pioneering study of discourse phenomena in the context of task-oriented dialogues. The collection of papers by ([27]) contain studies of the linguistic features of languages used in specific domains. ([45], [46], [40], and [37]) have studied the use of rhetorical relations in manuals, with the goal of generating natural-sounding paraphrases. ([16]) have studied some semantic issues of linguistic expressions used in manuals.

We concentrate here on those basic features of the documentation bound to raise questions about the proper use of natural-language processing techniques in the context of requirements specifications, and to identify some possible realistic solutions.

Although the texts varied widely in terms of contents, style, and length, we found that they share a number of characteristics in this regard.

### 2.2.1 Sentence Length

The first case study ([5]) is a small example of some 800 words. We found that the average sentence length is 27.42 words. We looked at another second case study ([14]), of some 60,000 words. A sample of this text found an average length of 31.24 words.

This figures of average word length suggest that current natural language techniques will have to be supplemented with other methods before they can be applied directly.

### 2.2.2 Punctuation

Punctuation is used (or not) in an extremely irregular manner, either because one writer does not use the same marks consistently, or because different writers have different intuitions of how to use them.

Lack of discipline in punctuation, allied with arbitrarily large sentences, was found to lead to extremely poor specifications. We found many examples similar to ([14], p 33):

“If selected to automatic at MCC1 for keyboard control at the supervisory computer, air compressors can be individually stopped/started manually and the outlet valves, opened/closed manually from the supervisory computer keyboard, provided (in the case of the air compressor) the differential air pressure switch is calling for that compressor to run.”

The first consequence of this is that punctuation marks should be incorporated into the grammar of the system sparingly. The second consequence is that free use of punctuation seems to encourage writers to type obscure paragraph-long sentences. Writers should therefore be discouraged from using punctuation marks whenever possible.



### 2.2.3 Presentational Units

One important discovery of the case studies' examination was that writers of requirements specifications resort very often to well-established forms to organise and present information. We will refer to these supra-sentential units as *presentational units*. These results were obtained by reviewing one case study ([5]), and a sample of some 40 pages of another one ([14]).

Aside from direct declarative statements and lists (to which we return below), the most preferred presentational units were:

- *Conditional (if-then-else) statements:*  
"if the open signal from xv13 or xv14 is not received the duty pump will be inhibited from starting". ([14], p 58)
- *When statements:*  
"when the level controller is switched to local operation, the set point will be [as] manually set at the controller". ([14], p 63)
- *Before and after statements:*  
"before starting the transfer procedure, the operator answers a menu in order to define the loading process"  
  
"after the operator has chosen which reservoir will be loaded, the outstation will check the input valves of all the reservoirs." (both from [5])

The distribution of these presentational units is variable, but some of them are very common. For example, the 'when/if'-sentence type is employed very often; a sample of 15 pages (p. 63-77) of [14] found 26 variations of this form.

Lists are one of the most common presentational devices used in the case studies. For example, a sample of [14], found that up to 40% was organised in list form. The following example is typical ([14], sec. 15.18, p 81):

- "The wash sequence initiation procedure will:
- a. Co-ordinate the filter wash requests.
  - b. Formulate a wash queue if necessary.
  - c. Start the wash sequence."

Lists are therefore a very productive way of presenting texts. It would thus be desirable to preserve some form of mechanism to allow writers to use them. The other main observation to be made here is that, unlike the presentational units listed above, lists can easily lead to under-defined specifications.

Consider the example above. The specification is not clear enough in a number of respects. We do not know whether the result of the sequence is one of the actions on the list (i.e. an *or-list*), or all three actions (i.e. an *and-list*). In the second case, we would also like to know whether we are dealing with an unordered sequence of actions (an *unordered-and-list*), or one where the actions are listed in the order in which they are to be executed (a *sequential-and-list*).

We turn now to a review of some techniques to introduce some discipline in the syntactic form of specifications.

## 2.3 Use of a controlled vocabulary

The first aspect of writing specifications that can be enhanced by a NLP system relates to the set of words and expressions that users employ when drafting specifications. For a given application, it is to be expected that both the vocabulary and the associated meanings will be, if not entirely fixed, reasonably limited. Furthermore, it is to be expected that all the users agree on their meaning. A simple NL system, using just some morphological and syntactic analysis techniques, can be written to define a controlled vocabulary, and ensure that the users employ only this vocabulary to define specifications. By ensuring that every definition consists of words in the authorised set, we can encourage all users to use words and expressions consistently while writing specifications. A more complete interface can also be used to restrict the task of adding words and expressions to a "super-user"; this would help the administration of the project and add some control to the process.

The words in the controlled vocabulary presumably reflect the most important concepts used in the specifications, and could also be used to automatically generate indices and lists of cross-references. This could be very useful for the administration of the project, especially when developing large specifications (which can sometimes run into the hundreds of pages). Cross-referencing in particular can aid consistency: whenever some concept is referred to, if other references to it are immediately made available it is much easier to see that all these references are mutually consistent. However, accurate cross referencing requires at least a modest degree of linguistic analysis. For example, one would want references to the 'pump operator' to be cross referenced to 'operator of the pump' but probably not to things like 'inoperative pump'.

This could of course take place within a more general management information system (MIS) to administer and document the entire development process: definitions, changes, amendments, and so on. When more than one person is involved in the specification, or it takes place over a long period of time, these techniques can also be useful to enforce a consistent style throughout a project, and support its administration.

The idea of using a limited vocabulary to restrict the intended meaning of a specification has been adopted in practice. The AECMA Simplified English standard comprises some 1500 words, each of which can only be used, with few exceptions, in one way only ([1]). Hoard et al ([23]) have built a system that, among other things, checks that input sentences comply with this standard.

## 2.4 Enhancing a simple style of writing

A more sophisticated use of NLP techniques involves the development of programs to analyse and criticise the *style* of a given text. In these applications, the goal is not only to check that every input sentence is grammatically correct, but also that it agrees with a set of pre-determined criteria. These criteria will typically try to produce documents that are clear and simple to understand.

For example, an organisation writing user manuals will try to enforce a style of writing consisting of simple, direct sentences, and could contain style rules of the following kind:

- favour affirmative sentences over passive ones
- mark noun phrases with several noun-noun compounds as inadequate
- do not use reduced relative clauses
- avoid the use of many qualifying phrases
- penalise sentences that are too large or contain more than a small number of coordinations

These criteria can then be used by a general purpose syntactic analysis system to score each sentence, and assign a rating that reflects its complexity. Hopefully, these constraints will enhance the author's strategy for organising information, and will result in documents that are easier to read and comprehend. These rules will often refer to syntactic characteristics that can only be provided by some sort of syntactic analysis, and will presumably be developed on top of a NLP system that performs full syntactic analysis (one such system has been produced by [36]).

Heidorn et al ([22]) report on such an early system designed to control the style of input texts. More recently — and using a formalism similar to the one in CLARE — Douglas and Dale([17]) have built another system for this purpose. In the context of specifications, the system mentioned above to help the writing of specifications according to the AECMA standard ([23]) performs a few of the syntactic checks recommended by the standard.

## 2.5 Keeping track of the information in the specification

So far, we have reviewed possible contributions of NLP techniques to the production of syntactically clear specifications. In this section, we sketch how NLP techniques might help the user see the information conveyed by an input specification written in English.<sup>2</sup>

Ideally, this information can be employed to organise a statement in such a way that it contains precisely the information one needs to derive a formal specification. At a very abstract level one can think of this as a translation problem: a translation from conversational English into a completely explicit sublanguage of English.

One of the challenges of using NLP in the context of formal specifications is the study of general techniques to produce good analyses without a detailed model of the universe of discourse. The application presented here does so by requiring the

---

<sup>2</sup>The idea was outlined in the project infancy as a possible technique to try, but was never carried out.

user to provide all the information necessary to carry out the contextual reasoning required to fully understand and disambiguate the sentence. Suppose that the user proposes the following sentence as part of a possible specification:

a reservoir will be chosen to be loaded, and the outstation will check all the input valves

The system can begin by pointing out that, before a successful analysis is produced, the user must completely determine the entities referred to by the phrase *the input valves*. The user then tries a more specific description:

a reservoir will be chosen to be loaded, and the outstation will check the input valves of every reservoir

A complete LF can be extracted, but the program might have to make assumptions that are marked as having a high cost. For example, the verb *to check* might be allowed to have a NP as complement, but it would prefer a *that sentence*:<sup>3</sup>

*“check the input valves of every reservoir” is possibly underspecified  
try “check THAT ...”*

The user finds that in fact an important piece of information is missing, one indicating what to check for. The next attempt is:

a reservoir will be chosen to be loaded, and the outstation will check that the input valves of every reservoir are closed

A further analysis of this LF might encounter two unconnected statements, suggesting that either a relation is being missed, or that there are actually two unconnected sentences that would be better if written separately. The user realises then that there is indeed a causal connection missing between the two sentences, so he suggests:

after a reservoir has been chosen to be loaded, the outstation will check that the input valves of every reservoir are closed

The consideration of this LF can point to the fact that there is a main event related to the choosing of the reservoir, but the potentially crucial information specifying who or what chooses the reservoir is missing. The user then corrects the sentence and types:

after a reservoir has been chosen by the operator, the outstation will check that the input valves of every reservoir are closed

This sentence fully describes the action of choosing a reservoir, but inadvertently, the purpose of the action has been deleted. If the system keeps track of the information contained at each step, it can detect this problem and prompt the user:

<sup>3</sup>We will write the system's response in italics.

*the event associated to "to choose" is less specific than before; previous qualifying "to be loaded" is missing*

The user re-reads the next-to-last sentence and suggests this time:

after a reservoir has been chosen to be loaded by the operator, the outstation will check that the input valves of every reservoir are closed

The system has no further suggestion, and the process finishes.<sup>4</sup>

## 2.6 Designing an English-like specification language

Another possible approach to control the form of specifications is to design artificial languages to write sentences that look like English ones, although their form and intended meaning are completely pre-determined. The idea is that these languages would be useful because they would completely constrain what can be said in them (like a technical language), but could be read by a non-specialist.

One can envision a NLP application to supporting the designer of such a language for specifications. To explain in detail how to do this, we will use an example application to frame the discussion. Assume that we are engaged in the production of an artificial language to draft the specification of a series of events — possibly subject to conditions— in the domain of gas storage applications.

To achieve this goal, we would like to define a grammar to produce sentences such as:

the outstation will check and open the reservoir input valves

the reservoir output valves will be closed depending on the operator's criteria

We first write some simple rules to generate a sentence of the first type:<sup>5</sup>

[1.1] event\_spec = entity\_spec + 'will' + list\_actions\_spec + entity\_spec

[2.1] entity\_spec = 'the outstation'

[2.2] entity\_spec = 'the reservoir input valves'

[2.3] entity\_spec = 'the reservoir output valves'

[2.4] entity\_spec = 'the operator's criteria'

...

---

<sup>4</sup>This approach could benefit from taking into account the scores with which CLARE rates the plausibility of each analysis found. An initial effort to directly use these scores as a measure of adequateness was not successful. See [32].

<sup>5</sup>We use the conventions introduced in chapter 1.

```

[3.1] list_actions_spec = action_spec
[3.2] list_actions_spec = action_spec 'and' list_actions_spec
[3.3] list_actions_spec = action_spec 'or' list_actions_spec
...
[4.1] action_spec = 'check'
[4.2] action_spec = 'open'
[4.3] action_spec = 'close'
[4.4] action_spec = 'be closed'
...

```

With them, we can produce the first sentence by applying the rules above to transform an `event_spec` into a sentence. At each step, we will take the leftmost `spec` and replace it according to some rule:

```

event_spec
(rule 1.1)
entity_spec + 'will' + list_actions_spec + entity_spec
(rule 2.1)
'the outstation will' + list_actions_spec + entity_spec
(rule 3.2)
'the outstation will' + action_spec + 'and' + list_actions_spec + entity_spec
(rule 4.1)
'the outstation will check and' + list_actions_spec + entity_spec
(rule 3.1)
'the outstation will check and' + action_spec + entity_spec
(rule 3.2)
'the outstation will check and open' + entity_spec
(rule 2.2)
'the outstation will check and open the reservoir input valves'

```

If we want to extend the grammar to include sentences of the second kind, we need to add only the following rule:

```

[1.2] restricted_event_spec = event_spec + 'depending on' + entity_spec

```

With this, and following essentially the same steps (rules 1.2, 1.1, 2.3, 4.4, 2.4), we obtain the second sentence. There are some problems, nonetheless. Inadvertently, we have opened the possibility of generating sentences such as:

the reservoir output valves will be opened and closed depending on the operator's criteria

which is ambiguous, that is, can be interpreted as meaning:

the reservoir output valves will be opened depending on the operator's criteria, and the reservoir output valves will be closed depending on the operator's criteria

or

the reservoir output valves will be opened, and depending on the operator's criteria, closed.

The second interpretation leaves it open that the valves may be opened without reference to the operator's criteria.

A NLP system could identify this kind of problem: first, code the grammar using rules such as the ones above, and generate a representative set of sentences from the grammar. Then, input this set of sentences to CLARE, which will produce as many analyses as possible for each example. If the grammar generates potentially ambiguous specifications, we will be able to detect this by finding that CLARE has identified valid sentences (i.e. sentences that were obtained by following correctly the rules of the artificial language) with more than one analysis, and are therefore ambiguous.

## 2.7 Evaluation of these proposals

The use of *controlled languages* aimed at limiting the vocabulary, syntactic forms, or sentence length in order to simplify sentence comprehension seems prima-facie to be reasonable. Unfortunately, there has been no single experiment to our knowledge to evaluate any such proposal. A proper evaluation must wait until such evaluation has been carried out. There is however some psycholinguistic evidence supporting at least some of these techniques (see [35] for an extensive survey of the literature in this domain). For example, experimental evidence suggests that the use of a limited vocabulary does not impede the communication of subjects collaboratively trying to solve a set problem.

There are other potential problems with controlled languages. First, they do not by themselves achieve the intended goal. As we have argued, many sentences in English have more than one interpretation, even if they are syntactically simple, short, and have been written using only authorised words. This phenomenon is so pervasive that general injunctions such as: "instructions should be as specific as possible," or "data should not be presented too quickly or in an unclear manner" are unlikely to be completely effective.

Controlled statements might be less clear than uncontrolled ones. For example, many controlled language checkers prefer the active rather than the passive so as to avoid the lack of clarity associated with a missing subject. But sometimes, the active version of a sentence may contain an ambiguity that is eliminated by the use of the passive:

You should connect the truck with the pump

The truck with the pump should be connected

The truck should be connected with the pump

There is a trade-off between striving for clarity and conciseness, while trying simultaneously to use a limited number of words. Without experimental evidence, it is not obvious whether controlled techniques deliver better documentation.

Controlled languages might also be incomplete; i.e. how do we know that everything that needs to be said can be said in a controlled language? An excessive amount of limitations can force the writer into producing statements that, although valid from the definition point of view, might not mean what the user wanted to write in the first place. For example (taken from [23]), a writer might want to type the statement "do not touch any cable," but due to the limitations of the standard, end up by writing "do not touch the cable" or "do not touch all the cables." These are valid alternatives, but have different semantics from the original one.

Finally, the use of "claims" languages must, as argued in the previous section, at least be complemented with some technique to ensure that the meta-grammar does not introduce unintended ambiguities.

## 2.8 Conclusions

We first reviewed the most salient characteristics of the case studies. We found that freely written specifications suffer from sentences that are too long, and use punctuation erratically, and are generally unclear. Both from the point of view of realistic use of current NLP techniques, and overall clarity of presentation, we suggest that we must aim at limiting the length of sentences, as well as eliminating when possible the use of punctuation. Importantly, the presentational units identified play a significant role in organising the overall structure of the texts. If we can control their use to eliminate under-specification, we can expect much better specifications, as such units constitute a significant proportion of specifications.

We also surveyed a number of proposals aimed at reducing the scope of ambiguity and vagueness in natural-language specifications. These proposals are based on the assumption that by limiting the form of the statements allowed, we will produce better documentation. Nevertheless, a constant theme in natural-language processing is that a large amount of ambiguity and vagueness obtains in all but the simplest of sentences. This is likely to happen no matter how restricted the vocabulary, syntax, or sentence length. In the context of specification tasks, this observation implies that merely controlling these factors will not produce unambiguous and precise specifications (although they might well contribute to making the specification statements more readable). A better assessment of their precise utility will have to wait until proper studies have been carried out in realistic specification settings.



## Chapter 3

# A Window-Based Interface for Specifications in Natural-Language

### 3.1 Introduction

The conclusions of the study so far are the following. We began by arguing that a fundamental problem for specifications in natural-language is that the main characteristic of natural languages is the massive amount of vagueness and ambiguity at all levels. We continued by examining a number of proposals to solve this problem. We found them wanting, mostly because they concentrate on the form of the language, without controlling the potential ambiguity of their meaning. Finally, we looked at the characteristics of some real specification documents. These documents show that, in order to apply current NLP techniques and improve their clarity, some control must be put not so much on the language being used, but in the actual process of *writing* the specification.

We present in this chapter an interface to a natural-language system that takes into account these observations. The interface works like a top-down, syntax-directed editor, using some (user-defined) statement types and ordinary English sentences. This approach reduces the length of the English sentences considerably, minimises the use of punctuation, and generally seems to encourage clear writing.

The user begins by selecting the kind of statement to be written, and recursively selects other statement types, or simply writes an ordinary English sentence. This sentence is passed on to CLARE (the NLP system we use) and analysed. If the sentence has more than one logical analysis, the user is requested to select the one they intended, disambiguating the input. The source specification in English is stored together with its logical analysis, and can be subsequently be modified, extended, or used to generate documentation. Assuming that the NLP system returns every conceivable analysis, this approach captures the intended meaning among those possibly intended.

Like the "claims language" or the "simplified English" standards, we take it as our starting point that the process of writing documents needs to be controlled to

produce better documentation. But rather than rely on a 'style manual', we use a natural language processing system to interact with the user. In this way we hope that the user and the natural language system will converge on an English statement of the requirements that represents an appropriate compromise between expressivity and clarity, as represented by the fact that the natural language system can arrive at a complete, disambiguated, analysis of the input.

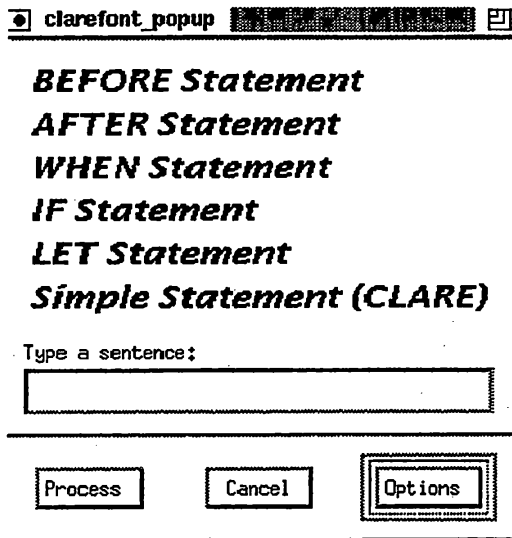
### 3.2 An Example

We present now the architecture of the interface. It has been designed to present an easy-to-use, controlled interface to the natural-language processing system at our disposal, CLARE ([3], [4]). In the configuration used here, CLARE carries out morphological, syntactic, semantic and some contextual analysis on an input sentence and then presents one or more possible logical forms representing the interpretation(s) of the sentence.

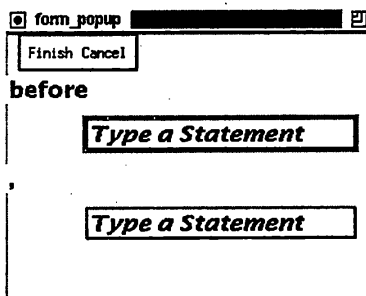
The user begins by selecting the kind of statement to be written, and recursively selects other statement types, or simply writes ordinary English sentences. Each sentence is passed on to CLARE and analysed. If the sentence has more than one logical analysis, the user is requested to select the one they intended, disambiguating the input. The source specification in English is stored together with its logical analysis, and can be subsequently be modified, extended, or used to generate documentation. In principle, the logical analysis could be subject to further forms of processing: for example, it could be linked to appropriate parts of a formal specification, or cross-indexed to a diagram.

We introduce the basic operation of the interface through an example from one of the specifications ([5]). Let us assume that the user wants to type in the following specification —based on an actual example: "Before starting the transfer procedure, the operator answers a menu to define the loading process (valves that will be closed or opened, the propane reservoir and the phases that will be used)."

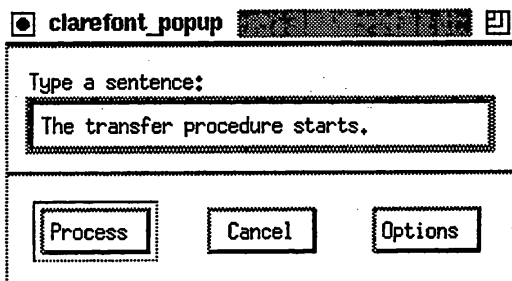
The system begins by displaying the options available to the user. After selecting the option to process a sentence, the system opens a window that displays the presentational units available, plus the option to type a sentence directly:



Let us begin with the first statement in the example: “before starting the transfer procedure, the operator answers a menu to define the loading process.” This sentence type corresponds to one of the presentational units available, the ‘Before statement.’ We can directly select the option from the menu, or type ‘Before’ and hit the return key. Because this is a predetermined statement, the system will retrieve its definition and structure, and selecting it will produce a new window:

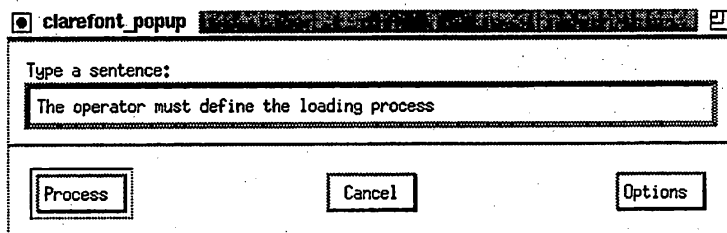


We choose now ‘Type a Statement’ to input the first part of the ‘Before statement.’ In the current implementation this will result in a new window where we can type the first English sentence:



If this sentence is in turn a complex one, then a new window is necessary: in this instance, the sentence is a simple one and the extra window a little redundant.

However, the sentence is then processed by CLARE. The system finds that the sentence is grammatically correct, and that it has one logical analysis. It is then considered correct, and stored for future reference. We go back to the 'before' window, and choose again to type an English statement with 'Type a Statement'. As before, another window pops up, and we type:



The screenshot shows a window titled "clarefont\_popup". Inside the window, there is a label "Type a sentence:" followed by a text input field containing the text "The operator must define the loading process". Below the input field, there are three buttons: "Process", "Cancel", and "Options".

The sentence is analysed and validated by the natural-language system; its logical analysis is saved. With this we finish; we have obtained a valid specification of the statement:

**BEFORE**

the transfer procedure starts,  
the operator must define the loading process.

### 3.3 Lists

Lists (conjunctions, disjunctions, or sequences) are treated by an extension of the mechanism shown above. Assume now that the user wants to draft the second part of the specification, and that the list of actions in parentheses ("valves that will be ...") is to be interpreted as a sequential coordination. The process of specification is carried out through the same mechanisms as before: select a 'when' statement from the menu, capture an English sentence, and then type in the conjunction. To do this, the user can either choose an option from the menu, or simply type:

The operator has defined which valves to open or close and then

The sequence 'and then' acts as a keyword to tell the interface that we have started a sequential conjunction; the system will then loop until an input statement lacks a trailing 'and.' We continue with the following sentences in the same fashion, until we obtain, instead of the plain English description, the controlled specification:

The operator has defined the loading process

WHEN

the operator has defined which valves to  
open or close

AND THEN

the operator has defined the target  
propane reservoir

AND THEN

the operator has defined the phases  
that will be used.

To conclude, the process of choosing high-level presentational units from the menu, and typing English sentences for the natural-language system, has taken us from the freely written version:

Before starting the transfer procedure, the operator answers a menu to define the loading process (valves that will be closed or opened, the propane reservoir and the phases that will be used)

to a controlled one:

BEFORE

The transfer procedure starts,  
the operator must define the loading process.

The operator has defined the loading process

WHEN

the operator has defined which valves  
to open or close

AND THEN

the operator has defined the target  
propane reservoir

AND THEN

the operator has defined the phases  
that will be used.

This specific rendering of the free version is only one among many that we could have typed. Our purpose here is not to suggest this specific style of drafting, but merely to illustrate how it can be done using the interface. Other writers, with different styles, will produce alternative specifications. A range of variant forms is permitted by the system, within the limits imposed by grammatical coverage. As we hope this example suggests, the approach chosen allows for better specifications in a relatively unconstrained manner.

### 3.4 Defining local identifiers

One factor that contributes to the naturalness of a text is the use of pronouns and definite noun phrases ('the ...'). Unfortunately, it can also create uncertainties on

how to understand it. Definite noun phrases can denote a specific object or objects in the discourse, can refer to some unspecified (and possibly non-existent) object fulfilling a description, or can be used generically to talk about the properties of a set of entities. The problem of establishing the correct reference for referring noun phrases is a well known one. Consider as an example the following fragment:

An operator executes the starting procedure when he opens the input valve, and then he activates the pump attached to the input valve.

...

An operator executes the closing procedure when he turns the pump attached to the input valve off, and then he closes it.

Both sentences contain references to "an operator" and "the input valve." In the absence of other information, it is not clear whether they refer to same entities or not. The pronouns "he" and "it" need also to be resolved to obtain a complete interpretation of the specification.

To ameliorate this problem, we have introduced a *LET facility*. This feature makes it possible for the user to introduce global names, and associate them to entities that will be repeatedly used in the specification. In the example above, the same sentences captured using the system and the LET facility will look like:

LET V be the input valve.  
LET P be the pump attached to V.  
LET OP be the operator.

OP executes the starting procedure WHEN  
OP opens V  
AND THEN  
OP activates P.

OP executes the closing procedure WHEN  
OP turns P off  
AND THEN  
OP closes V.

There is an alternative —and more natural— way of achieving the same effect. This can be done by using a construct like this:

An operator, OP, executes the starting procedure WHEN  
OP opens the input valve, V,  
AND THEN  
OP activates the pump, P.

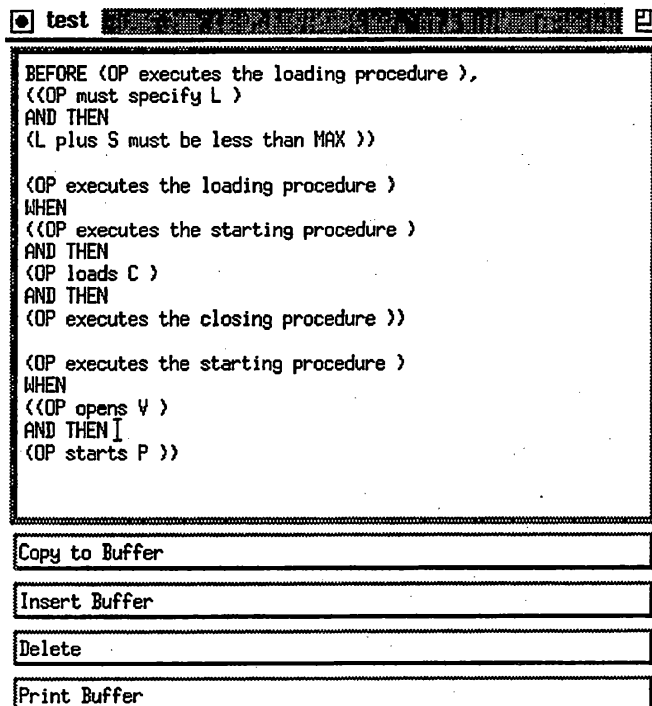
OP executes the closing procedure WHEN  
OP turns P off  
AND THEN  
OP closes V.

The first mention of an entity also introduces an identifier parenthetically, which can be used later on to establish reference to that entity. This is an extension of a mechanism already available in English. The result may read more naturally while still eliminating the possibility of this type of ambiguity.

Although the first technique works, it is not a particularly natural construct, linguistically speaking, and the resulting specifications look a little like expressions of a programming language. The tension between these two alternative ways of introducing identifiers points to the conflict that arises when simultaneously trying to achieve precision, conciseness, and intelligibility. Further work is required before we find how far we can stray from pure natural languages. Still, we believe that a certain amount of artificiality is justified by the mileage we get in terms of better specifications.

### 3.5 Storing, modifying, and reusing a specification

The process of capturing a specification statement is backed by a separate window that displays a text of the input at each moment of the process. The first use of this text window is to help users orientate themselves through the implicit tree created by selecting options on the main input window. The text window for one such specification is:



```
test
```

```
BEFORE (OP executes the loading procedure ),  
((OP must specify L )  
AND THEN  
(L plus S must be less than MAX ))  
  
(OP executes the loading procedure )  
WHEN  
((OP executes the starting procedure )  
AND THEN  
(OP loads C )  
AND THEN  
(OP executes the closing procedure ))  
  
(OP executes the starting procedure )  
WHEN  
((OP opens V )  
AND THEN  
(OP starts P ))
```

Copy to Buffer

Insert Buffer

Delete

Print Buffer

This output is produced from an internal representation, which contains a detailed description of each statement, as well as its corresponding logical analysis. It

can be used to produce various kinds of paraphrases according to the user's needs, or kept as an independent file for other kinds of processing. An example of an internal representation is:

```
spec(1,
    1,
    before(
        clare([id(op), executes,
            the, loading, procedure]),
        andthen( [clare([id(op), must,
            specify, id(1)]),
            clare([id(1), plus, id(s),
            must, be, less,
            than, id(max)])] ) ) ).
```

which corresponds to the specification:

```
BEFORE OP executes the loading procedure,
    OP must specify L
AND THEN
    L plus S must be less than MAX
```

The second purpose of the text window is to let the user copy, update, or modify specific parts of a statement without having to rewrite it from scratch. This is done by first selecting from the text window a statement or sub-statement with the mouse:

● test □

```
BEFORE (OP executes the loading procedure ),
((OP must specify L )
AND THEN
(L plus S must be less than MAX ))

(OP executes the loading procedure )
WHEN
((OP executes the starting procedure )
AND THEN
(OP loads C )
AND THEN
(OP executes the closing procedure ))

(OP executes the starting procedure )
WHEN
((OP opens V )
AND THEN
(OP starts P ))
```

Copy to Buffer

Insert Buffer

Delete

Print Buffer



Once a text has been selected, it can be deleted or copied to a special buffer. If it is deleted, this action creates a copy of the statement which is put into an auxiliary buffer for further manipulation. When a text is copied to the buffer, it can be added to the specification at any valid site. One such case occurs when a text has been deleted, creating a "hole" in the specification. Another possible insertion site is a sequences of statements linked with 'AND\_THEN' conjunctions. When there is such a site, and the buffer is not empty, the user can position the cursor there, and select 'insert.' The program will insert there whatever is found in the buffer.

Both the text window and the auxiliary buffer are generated anew every time the specification is modified or added to. It would be possible for edits of this kind to introduce ambiguities into previously checked specifications, of course, and so reprocessing is necessary to confirm the final result.

### 3.6 Ambiguous statements and paraphrasing

We have indicated above how to control for ambiguity at the level of presentational units. But to ensure that a complete statement is unambiguous, we must also check that each English sentence is unambiguous. We do this by adding a feature to the system that accepts a natural statement together with a specific interpretation. When CLARE processes a sentence and more than one logical analysis is found, the user is asked to choose among the various logical forms. The system then stores the original statement and its disambiguated logical form. This can be used for future reference, or in order to generate various paraphrases.

Interactive disambiguation of this type is a large research topic in its own right. Linguistically naive users may not always notice that a sentence is ambiguous, since contextual knowledge usually eliminates all but one linguistically possible interpretation, the others being unconsciously rejected as implausible. Furthermore, one cannot assume that the language used to represent the meanings of sentences (in our case, first order logic enriched with a few higher order constructs) will be familiar to users.

We have chosen a simple temporary solution to this problem, presenting users with a "logicians' English" paraphrase of the logical forms representing the meanings of the sentences. For example, if the user types in the sentence:

The operator has stopped the process on the menu

the system will generate the following two (slightly simplified) analyses:

```
quant(exists, A,  
      [operator,A],  
      quant(exists, B,  
            [and, [process,B],  
                  quant(exists,C,[menu,C],[on,B,C]])],  
      quant(exists, D,  
            [and, [event,D],  
                  quant(exists, E,
```

```

[current_time,E],
[precedes_in_time,D,E]],
[stop,D,A,B]))

```

and

```

quant(exists, A,
      [menu,A],
      quant(exists, G,
            [operator,G],
            quant(exists, M,
                  [process,M],
                  quant(exists, N,
                        [and, [event,N],
                              quant(exists, O,
                                    [current_time,O],
                                    [precedes_in_time,N,O]]],
                        [and, [stop,N,G,M],
                              [on,N,A]]))))))

```

These analyses correspond to the interpretations on which the phrase 'on the menu' modifies 'process' or 'stop' respectively. (compare 'the operator has read the message on the menu' and 'the operator has positioned the mouse on the menu'). The system generates paraphrases of each, and asks the user to choose between them:

1. There is some operator OP, some process PRO (such that there is a menu ME, and PRO is on ME), and a past event E, such that E is an event of OP stopping PRO.
2. There is some menu ME, some operator OP, some process PRO, and some past event E, such that E is an event of OP stopping PRO, and E is on ME.

From these paraphrases, the user chooses the one that corresponds to the intended interpretation. The system then stores the desired interpretation with the input sentence, which can later be used to annotate the English source, or substitute it directly.

The CLARE system is bidirectional, and can generate sentences from a logical form. It would thus be possible to generate full English sentences from the logical form back to the user. However, we would need careful checking to ensure that we did not simply re-generate the original sentence, and we would also need to ensure that the paraphrase did not itself introduce some other unintended ambiguity.

### 3.7 Evaluation of the system

In order to evaluate the basic design, we asked a few subjects to try a pencil-and-paper version of the system.<sup>1</sup> Although the current system contains some variants

<sup>1</sup>Thanks to Victor Carreño, David Carter, Paul Curzon, Roger Hale, and Ian Lewin.

compared to the one used in the example, it is basically the same. The experiment thus allowed us to gauge the user's reaction to such a system, and increase its functionality.

To evaluate the system, we asked the users to carry out the specification of operations on a simple system. We gave them examples of the style of English that the system is intended to handle, but encouraged users to go beyond this if they felt hindered by it.

The example specification involved a container of some sort, with one input and one output. The input to the container was through a pump, and the container had a sensor to measure how full the container was. We attached three valves to the container as well, one for the input, one for the output, and one emergency valve. The users were asked to specify some simple operations, such as loading the container, based on the following informal model intended to be representative of a simple and already quite clear specification:

"The loading operation will be done as follows:

1. The operator will specify the amount of material to be loaded into the container.
2. The operator will open the input valve V1.
3. The operator will then start the pump attached to the input valve.

After the process is over, the operator will turn the pump off, and he will close the input valve V1.

The amount to be loaded will not exceed the capacity of the container. If this happens, the loading operation will stop, and the emergency signal will be activated.

The results of the experiment were the following. First, we asked the users whether the overall design made the specification easy to follow. Most thought that the resulting style was indeed easy to follow, although it was noted that its style was perhaps too similar to a programming language, making it too unlike ordinary English. We also queried the users about the extent to which the system contributed to reach the goals that we had set. Their response was that it achieved those goals, but it was again noted that the resulting text was more difficult to follow than ordinary English.<sup>2</sup> It was the opinion of the users, especially those with specification experience, that it could be used in a realistic setting.

One comment that we took into account concerns the rigidity of the system. The version tried in the experiment did not include a facility to edit or change an ongoing specification. Several users pointed to the need of features to cut-and-paste, delete, or modify a draft as one goes along. As a result, we added the facility described above to correct this problem.

---

<sup>2</sup>This might of course reflect the shortcomings of an example we wrote, and not the system in general. As several users commented, some of our own drafts could have been done in a simpler and clearer way.

It was noted by the participants in the evaluation that the system needed to be supplemented by some kind of indication of the meaning as well as the form of the presentational units. For example, it was pointed out that the semantics of individual sentences such as "OP executes the X procedure" is influenced by the presentational unit in which it occurs. If we have "OP executes the X procedure WHEN..." then this statement usually defines what procedure X is, whereas "OP executes the X procedure" in "BEFORE OP executes the X procedure....." functions quite differently. In practice, the system should therefore be used in conjunction with a set of stylistic conventions (and presumably some explicit training) to give guidance to the writers on the intended meaning for each construct.

### 3.8 Further and Related Work

The current prototype has been developed with as little customisation of the NLP system as possible. We wanted to see how far we could go on the basis of already existing tools. Customisation is restricted to the addition of necessary vocabulary items. While the system accurately processes all the examples used in this paper, its coverage is still not adequate for real applications. However, the CLARE system provides a wealth of tools for this kind of customisation, given an adequate corpus of examples, and this extension of coverage is not a problem in principle.

Part of the process of customisation requires the addition of domain knowledge in a form suitable for inference, for the purposes of disambiguation. The addition of such knowledge makes it possible to use the logical forms which are the output of the natural language system for further types of processing.

The idea of building natural-language interfaces through menus was introduced by ([44]). However, in the system described there the composition of basic sentences also had to be achieved by menu selection, one word at a time. It may be useful to adopt this technique, or some variant of it, if it is impossible to guarantee that the NL system will accurately analyse the basic sentences in any other way. The use of natural-language techniques to help the process of specification writing has been explored by a few authors. ([6]) sketch a system that, among other capabilities, is able to build formal descriptions through natural-language dialogues. ([34]) describe a system to derive formal descriptions from natural language specifications. ([15]) contains a description of a knowledge-based system that maintains a database of a software development project. This system uses a natural-language system interface. ([18]) have built a system for the specification of automatic teller machines. These systems all have the common characteristic that they are application specific, with the natural language processing being hard-wired to the application in question. Our work has concentrated on using general purpose systems, on the assumption that customising will eventually be less effort than starting over again for each new application.

### **3.9 Conclusion**

We have presented a system designed to help in the process of writing specifications in natural language. The system controls the writing of statements in English, reducing the length and syntactic complexity of sentences, and introducing some structure in the specification. By forcing the user to choose between alternative analyses of a sentence, the system also ensures that the final statements are disambiguated. The supra-sentential presentational units that drive the system have been obtained from real-life specifications. They seem to be very general, but if desired, the system can be easily tailored to presentational units found in other domains.

## Chapter 4

# Natural-Language Interfaces to Formal Specifications

### 4.1 Introduction

The work reviewed so far is aimed at controlling the process of typing specifications to improve the source of a formal specification. Because of our emphasis on general techniques, we have so far approached the problem without consideration of the underlying domain of application. But many English requirements specifications correspond ultimately to formal specifications in a language such as RSL ([41]). The aim of this chapter is to complement those techniques with some other strategies to connect natural-language statements with their underlying representation.

Specifically, we study how to connect questions in English about a system, with its formal specification, and how to produce cooperative answers in an English-like language describing aspects of the specification. We do this by examining how to process several kinds of queries the users might pose in the domain of valve specification.

We expect to advance our understanding of the link between the English specification and the formal specification, which will eventually allow us to provide a natural-language system with modules that can be used to ask questions about the representation, and to produce paraphrases in English of the properties contained either explicitly or implicitly in the formal specification. It would thus be possible in principle for a non-expert to interrogate the specification to determine whether certain properties obtain or not.

This change of emphasis requires that we take into account the semantics of the system being described. Although it is not possible to provide an entirely domain-free characterisation of this link, we believe that by focussing on the *formal* properties of the representation we can generalise the connection between the English specification and its formal counterpart.

Our approach will show how, by treating a given specification as a data base, and by using the features particular to the underlying specification language, we

can produce a “commonsense axiomatisation”<sup>1</sup> that will allow us to process queries about the properties of the objects in the specification, as well as to generate precise descriptions in natural language of the specification. By further pursuing this line, we reduce the role of the natural-language requirements specification, and restrict its use to constrained translations to and from the formal specification.

This approach is not domain-independent. In other words, each domain requires a new effort to axiomatise its semantics, and needs to be done again when applying the same techniques to a different domain. As we hope will become clearer through the examples, this exercise demands a considerable amount of effort, and it is perhaps justified only in cases where many different specifications fall into a similar range of objects (e.g. pumps, valves, and so on). Otherwise, the quantity of work involved in the axiomatisation of a specific domain might not be justified.

The formalism of Horn clauses and programming in Prolog used in this chapter is introduced in the textbook of [43]. The work described here closely resembles work done to build natural-language interfaces to data bases. [13] extensively review the work done in this area.

## 4.2 Processing natural-language queries

The first issue that we want to examine is: what aspects do we have to consider if we want to build a natural-language interface to a formal specification? We can divide the problem into the following aspects:

- What kinds of natural-language statements can we practically process about a given formal specification?
- How do we represent a formal specification for the purposes of natural-language processing?
- What kinds of axiomatisations, and deductive operations on them, do we need to translate natural-language statements about properties of the representation into queries about the specification?

To facilitate the discussion, we will use an example taken from a concrete specification from the project about the formal properties of valves ([30], and [29]).

We are not concerned here with any particular language for formal specifications but rather with the information contained in them, either explicitly or implicitly. Modern specification languages have similar expressive power, so that the conclusions obtained here should hopefully be equally valid for all such languages.

We use in this paper the analyses produced by CLARE, but the general form of such analyses guarantees that our techniques will apply to any source analyses in first-order logic terms.

---

<sup>1</sup>In the spirit of [24].

### 4.3 Formal specification of a valve

As an example, we will take one from the domain of valves.<sup>2</sup> The purpose of the specification is to monitor and control the valve, using the sensors and the command as input, and producing as output a status signal and a command to open or close the valve.

Thus, we introduce a valve as an entity with three inputs, corresponding to:

1. A command to open or close the valve.
2. A boolean sensor which is true if the valve is open.
3. A boolean sensor which is true if the valve is closed.

and two outputs:

1. A command to the actuator of the valve.
2. A status signal to the operator.

A fragment of a specification is:<sup>3</sup>

```
scheme VALVE(T:TIME) = class
type
    Time = T.Time,
    Com_Val = open_c | close_c,
    Com = Time -> Com_Val,
    Act = Time -> Com_Val,
    Status = is_open | is_closed | is_opening | is_closing |
             sen_err | act_err,
    Sen = Time -> Bool,

value
    c,o: Time,
    valve: Sen x Sen x Com -> Act x Status,
    is_opening: Com x Time -> Bool,
    is_opening(c,t) <=> let x = com_age(t,c,close_c) in
                        x < o and x > 0 end,
    is_closing: Com x Time -> Bool,
    is_closing(c,t) <=> let x = com_age(t,c,open_c) in
                        x < c and x > 0 end,

    status: Sen x Sen x Com -> Status

    status(op,cl,t) <=>
        if is_opening(c,t) then is_opening
```

<sup>2</sup>This section is based on specifications provided by Hamid Lesan (personal comm, May 1994).

<sup>3</sup>Altering slightly the concrete syntax of RSL in an obvious way.



```

elseif is_closing(c,t) then is_closing
elseif op(t) = cl(t) then sen_err
elseif c(t) = close_c then
    if op(t) then act_err else is_closed end
else if cl(t) then act_err else is_open end
end
axiom
forall op, cl: Sen, c: Com, t: Time .
    status(op,cl,c) <=> let (_,s) = valve(op,cl,c) in
        s end,
(status(op,cl,c) <> sen_err and
status(op,cl,c) <> act_err ) =>
    let (act,_) = valve(op,cl,c) in
        act(t) = c(t) end
end

```

### 4.3.1 Processing of queries in English

The point we want to examine here is: assuming that we have written a formal specification of a system, and that this specification has been stored as a data base, what sort of statements can we expect a user to make about the system? In particular, we are interested here in questions, the kind of statements most likely to be useful in this setting. For example, a user interested in finding out information about the specification might ask questions like:

- what is the definition of a valve?
- what is the output of 'status'?
- how do I find whether the valve is opening?

It is of course true that, given that the descriptions contain arbitrary expressions using constructs of RSL and first-order logic, no algorithm exists that will answer every question one can ask about the specification. Still, as we will see, there remain several kinds of questions like the ones above that can be usefully answered.

To do so, we will take the question's analysis produced by CLARE, and use it to interrogate a data base corresponding to the formal specification. The link between both will be given by defining the semantics of the domain, and the *intended semantics* of the formal specification language. Because we use Prolog here, both the data base and the axiomatisations will be done in terms of Horn clauses.

The processing of a query is based on the evaluation of its analysis against the data base using a Prolog meta-interpreter, which uses Horn clause axioms to bridge the differences between the semantics of the logical forms and the data base. As a side effect, the query will produce the values that satisfy the logical form.<sup>4</sup> A generation module then takes the answer obtained by the meta-interpreter and produces a final answer.

---

<sup>4</sup>The interested reader is referred to the appendix, where this process is explained in more detail.

This approach avoids the kinds of problems normally attacked by theorem-provers in general because we decide beforehand which properties we are going to axiomatise, and how they relate to the kinds of questions we want to answer.

### 4.3.2 Representing the formal specification

We begin by representing in terms of Horn clauses the specification above. The transcription has been done with an eye to the processing needs we have, but basically all we are doing is representing the specification using a different syntax.<sup>5</sup> The following fragment of VALVE should illustrate the underlying idea:

```

type( sen,
      function_type,
      [time], [bool] ).

type( status1,
      enumerated_type,
      [is_open, is_closed, is_opening, is_closing, sen_err, act_err ] ).

type( com_val,
      enumerated_type,
      [ open_c, close_c ] ).

value( status,
       function,
       [sen, sen, com], [status1] ).

axiom( status,
       2,
       forall( [ type( op, sen ),
                 type( cl, sen ),
                 type( t, com ) ],
               equals( status(op,cl,t), second( valve(op,cl,c) ) ) ) ).

```

We look now at how we process different kinds of queries, according to their kind. The results described here have all been implemented and allow us, as we will see, to go all the way from a question in English to the data base, and back to a restricted form of English.

### 4.3.3 Direct questions about the specification

The first category of questions which we can handle directly include those referring to features that can be directly connected to some formal element in the definition:

---

<sup>5</sup>As opposed to a semantically-based translation, where we would translate each construct of RSL into a semantically-equivalent one in terms of Horn clauses.

what is the definition of a valve?  
 what is the output of function 'is\_closing'?  
 are there any axioms for 'status'?

Let us take the first sentence. CLARE will produce:

```
[whq,
  quant(wh,
    A,
    [impersonal,A],
    quant(exists,
      B,
      [and,
        [definition,B],
        quant(exists,C,[valve,C],[genit,B,C])],
      quant(exists,D,[event,D],[be,D,A,E^[eq,E,B]])))]
```

We look now at how this form is processed. Unless otherwise stated, we will assume throughout that the restriction of a quantified form will always be successfully evaluated, and will return the predicated variable "wrapped" in a functor bearing its type. For example, the evaluation of:

```
[impersonal,A]
[definition,B]
[valve,C]
```

will succeed in each case, returning the bindings:

```
A = imp(_)
B = def(_)
C = valve(_)
```

In terms of the axiomatisation, this means that there will be a number of Horn clauses such as:

```
impersonal(Imp):- Imp = imp(_).
definition(D):- D = def(_).
valve(V):- V = valve(_).
```

This approach allow us to concentrate in the definition of the main predicates. For the query above, this is the predicate be. Assuming that each variable has been tagged with its type, we want to define the conditions under which the goal

```
be( ev(D), imp(A), E^[eq,E,def(valve(B))] ).
```

succeeds. We provide the following axiom, saying that be will obtain in case Property makes reference to a certain Entity (valve in this case), and that entity has a corresponding Definition in RSL:

```

be( ev(Event), imp(What), Property ):-
    definition(Property,Entity),
    rcl_definition(Entity,Definition),
    What = Definition.

```

We use the following auxiliary definitions to connect those predicates to the date base:

```

definition(E^[eq,E,def(Of)],What):-
    Of=..[What|_].

```

```

rcl_definition(Entity,Definition):-
    type( Entity, function_type, From, To ),
    Definition = function_type( Entity, From, To ).

```

```

rcl_definition(Entity,Definition):-
    type( Entity, enumerated_type, List ),
    Definition = enumerated_type( Entity, List ).

```

```

rcl_definition(Entity,Definition):-
    value( Entity, function, From, To ),
    Definition = function( Entity, From, To ).

```

```

rcl_definition(Entity,Definition):-
    axiom( Entity, Num, Axiom ),
    Definition = axiom( Entity, Num, Axiom ).

```

When evaluated by the meta-interpreter, the query will produce the answer:

```

imp(function(valve, [sen, sen, com], [act, status1]))

```

This answer is passed on to a separate module whose task is to create responses that are more friendly to the user. This module will in this case produce the final answer to the original question. The output of the generator is produced by basically unwrapping the functional information around the answer and using it to build an English-like expression. The reader should nevertheless keep in mind that the motivation here is to build tools to reduce the amount of uncertainty surrounding full natural-language expressions. Therefore, the focus is not on generating completely natural-sounding English, but rather on producing paraphrases that faithfully reflect the contents of the database, even at the cost of some artificiality in the reply.<sup>6</sup>

In this case, the answer that we obtain is:

---

<sup>6</sup>We use this example only to show the kind of answer the system can produce. Depending on a more careful analysis of what the user needs, this answer could be tailored to produce more or less information, or it could be displayed using another layout.

valve is a function

Let SEN be a function type  
Let COM be a function type  
Let ACT be a function type  
Let STATUS be an enumerated type

Domain:  
SEN x SEN x COM

Range:  
ACT x STATUS

#### 4.3.4 Question about properties of the domain

A slightly more complicated case arises when a query refers to some feature of the object being defined. Consider for example the question:

How do I find the output of 'valve'?

Whereas we do not have to know anything about the domain to answer the questions exemplified in the previous section, questions about functions and their properties require an axiomatisation of the domain in question. In other terms, we need to examine first the *naive semantics* the users are working with when they pose them. In this case, it can be questions about the various states of the valve, or how to obtain information about them, or the conditions that produce a result.

The semantics of this example is very simple: each of the possible states of the valve one might be interested in correspond to one of the possible values of the type *Status*, the output of the function *status*. We then associate each of the possible queries to one of the possible values. The question above will produce the analysis:

```
[whq,  
  quant(wh,  
    A,  
    [manner,A],  
    quant(exists,  
      B,  
      [valve,B],  
      quant(exists,  
        C,  
        [and,  
          [output,C],[output_of,C,B]],  
        quant(exists,  
          D,  
          [event,D],  
          [and,  
            [find,D,user,C],  
            [manner_of,D,A]])))]).
```

Again, we concentrate on the main predicate. We define find as:

```
find( ev(Event), _Who, Property ):-
    property(Property,output,Entity),
    function_of_entity(Entity,Function),
    axiom(Some,Axiom,Definition),
    in_rhs(Function,Axiom),
    Event = Some.
```

with the following meaning: an Event of find will obtain for a certain Property if the property refers to a function related to the Entity in question (output and valve, here), there is a Function in the specification connected to the property, and there is an axiom somewhere in the specification whose right handside uses the function. This will lead to the following solution:

```
manner(ev(status))
```

which will, after some further manipulations to provide appropriate input to the generator, produce the reply:

By evaluating the status function

#### 4.3.5 Questions about functions and their properties

The method described above can be extended to deal with some more complicated questions, as in:

When does a valve open?

To answer this question, we need to make explicit the fact that each state of the valve is associated with a certain value in the type Status. First, the analysis of the sentence is:

```
[whq,
 quant(wh,
   A,
   [entity,A],
   quant(exists,
     B,
     [valve,B],
     quant(exists,
       C,
       [event,C],
       [and,[open,C,B],
         [when,C,A]])))]
```

We define open as follows:

```

open( ev(Event), What ):-
    valve(What),
    property_function(open,What,Function),
    property_value(open,What,Value),
    axiom(Some,Axiom,Definition),
    in_lhs(Function,Axiom),
    in_rhs(Value,Axiom),
    Event = Axiom.

```

which means that Event will correspond to the axiom that has on its left hand side the function that returns the information sought, and has on its right hand side the specific value (is\_open in this case) we look for. We complement this definition with a simple-minded rendering of when:<sup>7</sup>

```

when( ev(Axiom), ent(When) ):-
    axiom(Some,Axiom,Definition),
    When = Definition.

```

which retrieves the entire definition of the relevant axiom or axioms. The generated answer in this case is:

```

Let OP be of type SEN.
Let CL be of type SEN.
Let T be of type COM.

```

```

for every OP, CL, T:

```

```

    status of OP, CL, T is equivalent to:

```

```

    if
        is_opening(c,t), then is_opening
        elseif is_closing(c,t), then is_closing
        elseif op(t) = cl(t), then sen_err
        elseif c(t) = close_c, then
            if op(t), then act_err
            else is_closed,
        elseif cl(t), then act_err
        else is_open

```

## 4.4 Conclusion

We have presented an approach to treat natural-language questions about a formal specification as queries to a data base. The method that we use is based on the

<sup>7</sup>Simple-minded because we return the entire axiom. A more sophisticated system could try to isolate its *conditions*, which is the only information required. There is nonetheless a security issue here. It might in fact be a better strategy not to try to reduce the information produced to its simplest expression, but simply produce the entire paraphrase as we do in this example. The role of bringing in the intelligence required to make complete sense of the information remains thus with the human user, as it should.

axiomatisation of the meaning of the queries in terms of the representation of the specification as a data base, as well as an axiomatisation of the formal properties of the specification language constructs required by the query. This method allows the user to interrogate the contents of the specification in English, and to obtain answers in a quasi-English language.

We believe that this method can be extended in a general (if not too exciting) manner: begin by capturing the kinds of predicates bound to be of use, axiomatise their meaning in terms of properties of the data base and the underlying specification language, and finally write as axioms the properties of the functions needed to complete the axiomatisation. A specification language contains a large but not too large set of predefined functions (dozens rather than hundreds), all with well-defined semantics. It should then be possible, at least in principle, to collect a handful of useful predicates, and write a set of axioms for them.

Nevertheless, the writing of a full-blown axiomatisation for some concrete domain is a time-consuming task, and should be embarked on only after considering the cost-effectiveness of the solution.



## 4.5 Appendix 1: Interpretation of Queries

Given a logical form, we will apply the Prolog resolution mechanism to the predicates composing the query, possibly returning a set of *substitution* values that satisfy the query.<sup>8</sup>

The clauses giving the axiomatisation of the main predicates is described in the next section. With respect to the quantifiers and logical connectives contained in the formulae, we will treat them as follows:<sup>9</sup>

1. **[and,X,Y]** will be satisfied if there are substitutions S1 and S2 such that S1  $\circ$  X is satisfied,<sup>10</sup> and S2  $\circ$  ( S1  $\circ$  Y ) is satisfied.
2. **quant(forall,X,R,B)** will be satisfied if for every substitution S1 such that S1  $\circ$  R is satisfied, there exists a substitution S2 such that S2  $\circ$  ( S1  $\circ$  B ) is satisfied.
3. **quant(exists,X,R,B)** will be satisfied if there is at least one substitution S1 such that S1  $\circ$  R is satisfied, and such that there exists a substitution S2 such that S2  $\circ$  ( S1  $\circ$  B ) is satisfied.
4. **quant(wh,X,R,B)** will be interpreted procedurally as a command to find and display every X such that R and B.

One further detail that needs to be settled is the treatment of quantifier restrictions. Usually, the restriction of a quantifier contains the information usually associated with the *type* of the variable (file, event, etc). If we evaluate the logical form top-down, the meta-interpreter will attempt to satisfy the restriction before the body. Furthermore, this information is relevant for the evaluation of the body. Consider:

```
quant(exists,  
      B,  
      [file,B],  
      quant(exists,  
            C,  
            [event,C],  
            [and,  
             [read,C,user,B]]))
```

Essentially, this is equivalent to trying to prove that there exist B and C such that:

---

<sup>8</sup>A *substitution* for a term  $T$  is a list  $L$  of pairs of the form  $X = Y$ , where  $X$  is a variable in  $T$ ,  $Y$  is a term,  $X$  appears only once on the left side of a pair in  $L$ , and  $X$  does not occur on the right side of any pair in  $L$ .

<sup>9</sup>[19], ch 9.

<sup>10</sup>Given a term  $T$ , and a substitution  $S$ , we will denote by  $S \circ T$  the term that results from replacing in  $X$  by  $Y$  in  $T$ , for each pair  $X = Y$  in  $S$

```
file(B) and event(C) and read(C,user,B)
```

The variables here have no bindings. The first two conjuncts can be seen purely as constraining B and C to take values in the sets of files and events, respectively (as one would do it a typed system). Also, this information is relevant for the resolution of the predicate read, because the answer that we want depends on the type of B: the operation that we look for will vary depending on whether B is a file, a directory, etc. We will then assume that for each such predicate containing typing information there will be a Horn clause of the form:

```
file(B):- B = file(B1).
```

```
manner_of(C,manner(M)):- M = C.
```

to 'mark' each variable with its type according to the restriction. By doing so, when the meta-interpreter attempts to evaluate the main predicate, the restrictions will have been processed, and the main predicates will have the form:

```
read(ev(C1),user,file(B1))
```

The definition of read can then use this information to obtain the type of the variable. We will use the same approach to return in the argument variables any results of evaluating a goal.

## Chapter 5

# Generating English-Like Paraphrases of Formal Expressions

### 5.1 Introduction

In the last two chapters, we have introduced various mechanisms to produce paraphrases of underlying formal representations. Our main interest in this chapter is to further explore the question of how to produce paraphrases of formal expressions in a way that, on the one hand, are as unambiguous as possible, and on the other, close enough to ordinary English to be understood by those without knowledge of logic or set theory. We concentrate on the issue of how to generate paraphrases from logical expressions involving quantifiers, although we also briefly touch on the appropriate generation of paraphrases from set-theoretical expressions. These paraphrases are produced by very simple programs that use “canned” expressions that directly expand the representations.

As we have repeatedly remarked, natural languages contain a wealth of sources of potential ambiguity and vagueness. If the information to be produced is originally coded in some formal language, it follows that the closer we get to actual natural language, the higher the potential for misunderstandings. The dangers of such occurrences in specification tasks are obvious. The main emphasis of our work is thus not on natural-sounding paraphrases, but rather on paraphrases that are close enough to ordinary English to be understood by those without knowledge of logic or set theory.

#### 5.1.1 Logical Expressions

The first class of formal expressions that we are interested in paraphrasing is the one of logical expressions.<sup>1</sup> The most useful setting for a paraphrasing mechanism is

---

<sup>1</sup>We centre here on those specific logical forms produced by CLARE. The method should be applicable to any other formalism based on first-order logic.

when a user needs to decide on the correct analysis among a number of possibilities proposed by a natural-language processing system. For example, when the sentence:

an operator starts every transfer of propane

is analysed, it results in two different logical forms, one for each possible reading:

```
quant(exists,  
      A,  
      [operator,A],  
      quant(forall,  
            B,  
            [and,[transfer,B],  
              quant(exists,C,[propane,C],[genit,B,C]]),  
            quant(exists,D,[event,D],[start,D,A,B])))
```

and

```
quant(forall,  
      A,  
      [and,[transfer,A],  
        quant(exists,B,[propane,B],[genit,A,B]]),  
      quant(exists,C,[operator,C],quant(exists,D,[event,D],[start,D,C,A])))
```

The simplest way to paraphrase these sentences is to produce the kind of English expressions one finds in logical books:

There exist an A, such that A is an operator, and such that for every B, if B is a transfer, and there exists a C such that C is some propane, and B is 'of' the C, then there exists a D, such that D is an event, and D is an event of A starting B.

and

For every A, if A is a transfer, and there exists a B such that B is some propane, and the A is 'of' the B, then there exists a C, such that C is an operator, and there exists a D, such that D is an event, and D is an event of A starting B.

The method to do this is not too complicated; the target sentences have exactly the same structure as the input logical forms, and most of the work can be done by substituting canned expressions like 'for every', and 'there exists' instead of the logical connectives. These expressions are nonetheless unsatisfactory for two reasons. First, they are so convoluted that it is doubtful that they help us to achieve the goal of facilitating the understanding of a logical statement by somebody who does not read logic. Second, their complication suggests that it is quite possible that a user will misunderstand them. Thus, from a security point of view, it can be argued that this solution is in fact worse than not having a paraphrase at all. Those who can

read logic will still be able to understand the first, unambiguous expression, while those who cannot at least will not get the impression that they are understanding what they are reading.

The two analyses in question differ only in the order of the quantifiers. One possibility would be to build a program to identify the differences in the analyses and present the user with options such as:

an operator starts every transfer of propane, where there is a possibly different operator for each transfer,

or

an operator starts every transfer of propane, where it is the same operator for each transfer

One approach would be to use this idea and extend it to other kinds of situation where more than one analysis is available. There are nonetheless several reasons why the logical forms might differ beyond quantifier ordering. The only general approach would be to try to identify the differences among each branch of every analysis (a partial solution for the representation used by CLARE could take advantage of the incremental construction of the logical form to do this at each step of the form's refinement), and then generate a natural-language expression to point to their differences. Unfortunately, the problem of looking for similarities and differences in each branch of every formula is exponential in nature. It may be that the formulae we are dealing with are small enough that this complexity does not matter. But we have in any case decided to settle for trying to produce a paraphrase directly from the logical form, but using natural language when possible to simplify the output, as above.

We first notice that logical expressions involving quantifiers arise systematically from certain syntactic forms:

"every operator starts the process"

corresponds to

`forall Op. operator(Op) => starts(Op,the-process)`

and has the paraphrase: "for each Op, such that Op is an operator, Op starts the process", whereas

"some operator starts the process"

`exists Op. operator(Op) and starts(Op,the-process)`

with the paraphrase: "there is some Op, Op is an operator, and Op starts the process."

We assume for now that the restriction in the formulae contains a single predicate as above. By using these forms instead of a more literal paraphrase of the logical form, we obtain:

There is some A, A is an operator, and  
for each B, such that B is a transfer (...),  
there is some D, D is an event, and  
D is an event of A starting B.

and

For each A, such that A is a transfer (...),  
there is some C, such that C is an operator, and  
there is some D, such that D is an event, and  
D is an event of A starting B.

We indicate how to complete the rest of the restriction. In general, expressions with complex restrictions such as

forall X. ( p(X) and q(X) ) => r(X)

arise from English expressions containing modified noun phrases, as *transfer of propane*. We will apply the same rules to paraphrase them as the ones used in the main body, with the addition of the words "and is such that." With this, we obtain the final versions (with the rest of the restrictions in parenthesis):

There is some A, A is an operator, and  
for each B, such that B is a transfer (and is such that  
there is some C, C is some propane, and  
B is 'of C'),  
there is some D, D is an event, and  
D is an event of A starting B.

and

For each A, such that A is a transfer (and is such that  
there is some B, B is some propane, and  
A is 'of B'),  
there is some C, such that C is an operator, and  
there is some D, such that D is an event, and  
D is an event of A starting B.

This treatment works as well for more complex examples. Consider

some operator of each pump in some plants snores

Given the analyses:

```
quant(forall,  
  A,  
  [and,  
    [pump,A],  
    quant(exists,B,[plant,B],[in,A,B])],
```

```

quant(exists,
      C,
      [and,
       [operator,C],
       [and,[operator,C],[genit,C,A]]],
      quant(exists,D,[event,D],[snore,D,C]))

```

and

```

quant(exists,
      A,
      [and,
       [operator,A],
       quant(forall,
             B,
             [and,
              [pump,B],
              quant(exists,
                    C,
                    [plant,C],
                    [in,B,C]]],
             [genit,A,B]))],
      quant(exists,D,[event,D],[snore,D,A]))

```

We obtain for the first analysis, according to the rules:

For each A, such that A is a pump (and such that there is some B,  
 B is a plant, and  
 A is in B),  
 there is some C, such that C is an operator (and such that C is 'of' A),  
 and  
 there is some D, D is an event, and  
 D is an event of C snoring.

The second one corresponds to:

there is some A, A is an operator (and such that for each B,  
 such that B is a pump,  
 there is some C, C is a plant, and  
 B is in C), and  
 A is 'of' B, and  
 there is some D, D is an event, and  
 D is an event of A snoring.

Another way to simplify further these expressions is by refining the use of the restrictors. In general, instead of writing:

for each X, such that X is P, ...  
 there is some X, and X is P

we can simply write:

for each P X,...  
there is some P X,...

so that for

"every operator starts some process"

or

forall Op. operator(Op) => exists Pr. process(Pr) and starts(Op,Pr).  
exists Pr. process(Pr) and forall Op. operator(Op) and starts(Op,Pr).

we obtain

paraphrase = for each operator Op,  
                  there is some process Pr,  
                  and Op starts Pr  
paraphrase = there is some process Pr, and  
                  for each man Op,  
                  Op starts Pr.

This move reduces the last two analyses to:

for each pump A,  
          and such that there is some plant B,  
                  A is in B, and  
there is some operator C,  
          such that C is 'of' A, and  
there is some event D, and D is an event of C snoring

The second one produces:

there is some operator A,  
          and such that for each pump B,  
                  and such that there is some plant C, and  
                          B is in C, and  
          A is 'of' B and  
there is some event D, and  
D is an event of A snoring.

The corresponding paragraph versions are (modifying the punctuation to get rid of nested parenthesis):

For each pump A (such that there is some plant B, and A is in B),  
there is some operator C (such that C is 'of' A), and  
there is some event D of C snoring.

and



There is some operator A (such that for each pump B --- where there is some plant C, and B is in C---, A is 'of' B), there is some event D, and D is an event of A snoring.

Note that the second expression is not very good. A better alternative is to transpose the second qualifying phrase to the end of the clause:

There is some operator A (such that for each pump B, A is 'of' B --- where there is some plant C, and B is in C), there is some event D, and D is an event of A snoring.

We could also say:

There is some operator A. For each pump B (such that there is some plant C, and B is in C), A is 'of' B, and there is some event D of A snoring.)

The use of the original English expressions from which quantifiers originate seems to provide us with reasonably robust interpretations. Nevertheless, it is desirable in security contexts to ensure as far as it is possible that the user is getting the intended reading of a sentence. Our final modification is to add explicit information about the relative quantifier scopings with this purpose. This is done by identifying every predicate which contains some variables existentially quantified, and some universally quantified:

For each pump A (such that there is some plant B, and A is in B), there is some operator C (such that C is 'of' A), and there is some event D of C snoring, where:

There is a possibly different plant B for each pump A, and there is a possibly different operator C for each pump A.

The second one corresponds to:

There is some operator A (such that for each pump B --- where there is some plant C, and B is in C---, A is 'of' B), there is some event D, and D is an event of A snoring, where:

There is a possibly different plant C for each pump B, and it is the same operator A for each pump B.

Eliminating the 'of' expressions would also be desirable, but requires some linguistic knowledge. For many cases, given an expression like 'A is of B' one can use the predicate associated with A: 'such that A is the operator of B'. But this will not always be correct, for there may not be a unique A associated with B: consider 'A is the wheel of B' (where B is a car). However, 'A is a wheel of B' sounds clumsy. Perhaps it would be best to pass the decision back to the user by generating 'A is a/the wheel of B'.

## 5.1.2 Set-theoretic expressions

Formal specification languages often allow a combination of logical and set-theoretic expressions. A useful feature of a paraphrasing mechanism would be to be able to paraphrase these expressions as well.

### Set expressions

We assume throughout that we have, for each set expression that we want to paraphrase, a suitable internal representation and a "canned" expression describing each internal predicate (add note).

The simplest expressions involve extensional definitions of sets. We paraphrase the symbol "==" as "is defined as", in expressions like:

```
Benelux == {Belgium,Netherlands,Luxembourg}
```

A possible representation would be something like:

```
define( setid(Benelux),
        enumerate([Belgium,Netherlands,Luxembourg]) ).
```

and the expressions:

```
p( define,      "is defined as" ).
p( setid,      "the set containing" ).
p( enumerate, "the member(s)" ).
```

By writing some simple-minded code to substitute the formal expression by its corresponding natural-language string, we can directly produce:

```
"the set Benelux is defined as the set containing the members
Belgium, Netherlands, and Luxembourg."
```

Sets can also be defined in terms of properties. We use the paraphrasing method of the previous section. For example,

```
Sixes == {N:integer | exists M:integer(M). N = 6 x M}
```

which corresponds to:

```
"the set Sixes is defined as the set of integers N, such that
there is an integer M, and N equals 6 times M."
```

or:

```
First == { M: set_of_players | and(has_two_members(M),subset(M,competitors))}
```

which is:

The set First is defined as the set of players M, such that:  
M has two members,  
and  
M is a subset of competitors

or in paragraph form,

"The set First is defined as the set of players M, such that M has two members, and M is a subset of competitors."

Operations on sets also directly correspond to their representation; identifiers are produced directly (operations prefixed by "the"):

```
in(Belgium,EEC)
union(EEC,Nato)
intersection(Central_America,Scandinavia)
```

which produces:

```
"Belgium is a member of EEC"
"the union of the set EEC and the set Nato"
"the intersection of the set Central America and the set Scandinavia"
```

Equations involving sets can be produced using

```
equal(union(a,b),union(b,a))
equal(intersection(central_america,scandinavia),
      empty_set)
```

or:

```
"the union of the set A and the set B equals the union of the set B
and the set A."
```

```
"the intersection of the set Central America and the set Scandinavia
equals the (set) empty set."
```

We have expanded here the type information on the left side, but not on the right side. We could also do it, by introducing a relative clause, and obtain:

```
"the union of the set A and the set B equals the set which is the
union of the set B and the set A."
```

```
"the intersection of the set Central America and the set Scandinavia
equals the set which is the empty set.")
```

## Relations

Relations are particular kinds of sets, in which each member is a pair. The method of paraphrasing is therefore alike. Using the same approach we used for sets, we obtain from:

```
Rides == {(Alice,bicycle),(Huw,bicycle),(Ben,car),(Ben,bicycle)}
```

the sentence:

```
"the relation Rides is defined as the set containing the pairs  
(Alice,bicycle), (Huw,bicycle), (Ben,car), (Ben,bicycle)."
```

The relational definition:

```
wrides == domain_restr( {alice,kate}, rides )
```

would produce:

```
The relation Wrides is defined as the domain restriction of the set  
containing the members Alice and Kate, and the relation Rides.
```

We can also use the definition itself to produce a more explicit paraphrase:

```
"The relation Wrides is defined as the set of x in X, and y in Y, such  
that x is in the set containing the members Alice and Kate, and the  
pair (x,y) is in the relation Rides."
```

## Functions

Functions are restricted relations. For example:

```
Stock == partial_function(ITEM,N)
```

We paraphrase the definition as:

```
"The partial function Stock maps an item I into a non-negative number N"
```

If partial functions are defined as well,

```
partial_function(X,Y) == {R:X <-> Y | forall x:X, y,z:Y .  
                        (x,y) in R and (x,z) in R => y = z}
```

we could give a 'deeper' paraphrase, possibly as a response to a request for further clarification:

```
"The partial function Stock is defined as a relation R between the set  
ITEM and the set N, such that for every item I, every non-negative number  
NO, and every non-negative number N1, if (I,NO) is in R, and (I,N1) is  
in R, then NO equals N1."
```

## Equations

Consider compound expressions such as:

$$A - B^2$$
$$(A \cup B) \cup C$$

which we represent as:

$$\text{minus}(A, \text{squared}(B))$$
$$\text{union}(\text{union}(A, B), C)$$

Using the method so far, we would get

$$A \text{ minus } B \text{ squared} \dots$$
$$\text{The union of the union of } A \text{ and } B \text{ and } C \dots$$

which are ambiguous. Using parentheses solves the problem, but the paraphrases remain unclear:

$$A \text{ minus } (B \text{ squared}) \dots$$
$$\text{The union of } ( \text{the union of } A \text{ and } B ) \text{ and } C \dots$$

This method becomes worse for more complicated expressions. We suggest the introduction of local identifiers (we mention the type of the identifier the first time it is mentioned):

$$\text{The union of the set } D \text{ and the set } C, \text{ where:}$$
$$D \text{ is the union of the set } A \text{ and the set } B \dots$$

or

$$\text{The union of the set } D \text{ and the set } C \text{ (where } D \text{ is the union of the set } A \text{ and the set } B), \dots$$

or

$$\text{Let the set } D \text{ be the union of the set } A \text{ and the set } B. \text{ The union of } D \text{ and the set } C \dots$$

This method can be applied generally, and is especially useful in paraphrasing equations (omitting type information this time):

$$(A \cup B) \cup C = A \cup (B \cup C)$$
$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

$$\text{The union of } D \text{ and } C \text{ equals the union of } A \text{ and } E \text{ (where } D \text{ is the union of } A \text{ and } B, \text{ and } E \text{ is the union of } B \text{ and } C).$$

or

Let D be the union of B and C. Let E be the intersection of A and B.  
Let F be the intersection of A and C. The intersection of A and D  
equals the union of E and F.

This approach can also be used in functional equations:

$$\text{override}(F,G) = ((\text{dom } G) \text{ domain\_subtract } F) \cup G$$

corresponds to:

The override of the function F and the function G equals U, where:  
U is the union of S and G,  
S is the domain subtraction of D and F, and  
D is the domain of G.

Another example is:

$$(S \text{ domain\_restr } R) \cup (S \text{ domain\_subst } R) = R$$

which results in:

The union of the set D1 and the set D2 equals the relation R, where:  
D1 is the domain restriction of the set S and R, and  
D2 is the domain subtraction of S and R.

## 5.2 Summary

We have presented a simple approach to generate paraphrases of formal expressions using "canned" expressions, specially first-order logic expressions involving quantifiers.

Surprisingly, there seems to have been very little work on this question. [11] looks at the translation of proofs into English; [31] suggest how one might generate paraphrases of relational database queries. More generally, the question of how to generate natural-sounding paraphrases of quantitative data has been approached by a number of researchers. [45], [46], [40], and [37] have studied the use of rhetorical relations to generate natural-sounding paraphrases in manuals. [28], [39], [25], [26], [8], [38], and [33] have also worked on the paraphrasing of quantitative data. The last work is exceptional in that it considers the consequences of generating ambiguous expressions.

# Chapter 6

## Conclusions

We set out to investigate the possible use of natural-language techniques in the area of requirements specifications. We recapitulate now on the main lessons of our participation in the project.

The first important conclusion was to point out that the well-known ambiguity and vagueness of natural-language has important consequences for the process of deriving a formal specification from a source one written in a natural language. Unless there is some control over the source specification, one must not accept unquestioningly the validity of the formal specification (in the sense of corresponding to the intentions of the source specification writer).

We then examined the form of the source documents and reviewed a number of proposals to impose some discipline in the writing of specification statements. We found that, in the main, sentence length and faulty punctuation conspire to produce poor specifications, as well as making the application of current natural-language processing techniques problematic.

After reviewing some ideas aimed at restricting specifications along various lexical and syntactic dimensions, we concluded that these approaches have some shortcomings. First, because restrictions over the form of some specification statement do not by themselves produce clearer or less ambiguous statements. Second, because these ideas are based on plausible criteria to improve the readability of texts, but have not been experimentally tested in realistic settings, and must therefore wait until this has been done to be properly assessed.

One important finding of the sample specifications was that writers naturally organise their text in terms of a small number of presentational units. This observation was the starting point for the design of a window-based interface to a full natural-language system. This system is our main contribution to the project, and is fully described in chapter 3. The system is domain-independent; in comparison to syntax-based approaches, it controls the amount of ambiguity; and due to its design, it encourages short-sentence writing with minimal or no punctuation.

We also explored in some detail the issue of how to use a formal specification to drive a natural-language query system. The results are encouraging, and suggest that further work along these lines could help both to reduce the need for different kinds of documentation, and complement other techniques (e.g. animation) designed

to help in the understanding of the properties of formal specifications. On the other hand, such applications lack the generality of other domain-independent techniques, and will probably be of use only where the investment is justified.

We concluded by addressing the issue of how to generate simple paraphrases of quantified first-order logic expressions and set-theoretical expressions. As before, our main interest is to avoid ambiguity, even at the cost of artificiality in the paraphrasing. This chapter complements the use of paraphrases in previous chapters.

At the end of the project, our conclusion is that there remains a wide scope for the application of NLP to specification tasks. Unlike many other areas where natural-language is an optional extra, natural language already has a well-established role to play in the process of capturing specifications in English, and linking them to formal specifications in a formal language. This is so for two reasons. One, because a natural language is the only one understood by every participant in the project, both on the side of clients and developers, and will continue to be so for the foreseeable future. The other, because formal specifications can help in the production of a system documentation even after a system has been delivered. In safety-sensitive domains, a strengthening in the link from formal specifications to natural-language can only be helpful.

More generally, we found through our interaction with the industrial partners in the project that they consider formal specification methods essentially unintelligible. Before formal methods are adopted more widely in industry, their use should be made more accessible ([9], [21]). This is specially true of safety-sensitive applications, where its use by uncertain clients might constitute an added risk.

We believe that the present study shows some promising applications where natural-language processing techniques can contribute to the writing of better specifications, and help in the adoption of formal methods in extra-academic environments.



# Bibliography

- [1] AECMA. 1989. Aerospace Industries Association Simplified English Manual. Derby. BDC Publishing Services.
- [2] Allen James. 1995. Natural language understanding. Redwood City, Ca. Benjamin/Cummings. Second ed.
- [3] Alshawi, Hiyan, David Carter, Richard Crouch, Steve Pulman, Manny Rayner and Arnold Smith. 1992. CLARE: a contextual reasoning and cooperative response framework for the Core Language Engine. Final report. Cambridge. SRI International.
- [4] Alshawi, Hiyan (ed). 1992. The Core Language Engine. Cambridge, Mass. MIT Press.
- [5] Banks Roger. 1991. Transmittion case study. Requirement specifications. MORSE Project, Doc. Id. MORSE/TRANSMITTON/RJB/1/V1.
- [6] Bergland, G David, Geoffrey H Krader, D Paul Smith, and Paul M Zislis. 1990. Improving the Front End of the Software-Development Process for Large-Scale Systems. AT&T Technical Journal, March-April. 7 - 21.
- [7] Bloomfield Robin E and Peter K D Froome. 1986. The Application of Formal Methods to the Assessment of High Integrity Software. IEEE Transactions on Software Engineering. Vol SE-12, no. 9, September. 988 - 993.
- [8] Bourbeau, L, D Carcagno, E Goldberg, R Kittredge, and A Polguere. 1990. Bilingual generation of weather forecasts in an operations environments. Proc COLING. 90 -92.
- [9] Bowen,Jonathan and Victoria Stavridou. 1993. The Industrial Take-up of Formal Methods in Safety-Critical and Other Areas: A Perspective. In James C. P. Woodcock and Peter G. Larsen (eds). FME'93: Industrial-Strength Formal Methods. Berlin. Springer.
- [10] Charniak Eugene and Drew McDermott. 1985. Introduction to Artificial Intelligence. Reading, Mass. Addison-Wesley.
- [11] Chester, Daniel. 1976. The Translation of Formal Proofs into English. Artificial Intelligence. 7. 261-278.

- [12] The Claims Language. Annex B, version 1, May 1990. ITSEC. 116-125.
- [13] Copestake Ann and Karen Spärck Jones. 1990. Natural language interfaces to databases. *Knowledge Engineering Review*. 5 225-249.
- [14] Cuff, Alan. 1992. Transmittion Case Study No. 2. The Sutton Bingham Water Treatment Works. MORSE Project. Doc. Id. MORSE/TRANSMITTON/ARC/4/V1.
- [15] Devanbu, Premkumar, Ronald J Brachman, Peter G Selfridge, and Bruce W Ballard. 1991. LaSSIE: A Knowledge-Based Software Information System. *Comm ACM*, vol 34, no 5, 35-49.
- [16] Di Eugenio, Barbara and Michael White. 1992. On the Interpretation of Natural Language Instructions. *Proc Int. Conf. Comp. Ling. (COLING)*. 1147-1151.
- [17] Douglas, Shona and Robert Dale. 1992. Towards robust PATR. *Proc. Int. Conf. Comp. Ling. (COLING)*. 468-74.
- [18] Fuchs, Norbert E, Hubert Hofmann, and Rolf Schwitter. 1994. Specifying Logic Programs in Controlled Natural Language. Zurich, Department of Computer Science, University of Zurich, Technical Report 94.17, November.
- [19] Gazdar, Gerald and Chris Mellish. 1989. *Natural language processing in Prolog: an introduction to computational linguistics*. Wokingham. Addison-Wesley.
- [20] Grosz, Barbara. 1978. Discourse Knowledge. In Donald E. Walker (ed). *Understanding spoken language*. New York. Elsevier North-Holland. 229 - 344.
- [21] Hall, Anthony. 1990. Seven Myths of Formal Methods. *IEEE Software*. Vol 7, no 5. p 11 - 19.
- [22] Heidorn, G. E., K. Jensen, L. A. Miller, R. J. Byrd, and M. S. Chodorow. 1982. The Epistle text-critiquing system. *IBM Systems Journal*. 21. 305-26.
- [23] Hoard, James E., Richard Wojcik, and Katherina Holzhauser. 1992. An automated grammar and style checker for writers of simplified English. In Patrick O'Brian Holt and Noel Williams (eds). *Computers and Writing. State of the Art*. Oxford, Intellect, p 278-296.
- [24] Hobbs, Jerry R and Robert C Moore (eds). 1985. *Formal Theories of the Commonsense World*. Norwood, Ablex.
- [25] Iordanskaja, Lidija, Richard Kittredge, and Alain Polguere. 1991. Lexical selection and paraphrase in a meaning text generation model. In Cecile L. Paris, William R. Swartout, and William C. Mann (eds). *Natural Language Generation in Artificial Intelligence and Computational Linguistics*. Dordrecht, Kluwer, 293-312.

- [26] Iordanskaja, Lidija, M Kim, Richard Kittredge, B Lavoie, and Alain Polguere. 1992. Generation of extended bilingual statistical reports. Proc COLING, 1019-1023.
- [27] Kittredge, Richard and John Lehrberger (eds). 1982. Sublanguage: studies of language in restricted semantic domains. Berlin, de Gruyter.
- [28] Kukich, Karen. 1983. The design of a knowledge-based text generator. Proc Assoc. Comp. Ling. (ACL), 145-150.
- [29] Lesan, Hamid. Personal communication. May 1994.
- [30] Lesan, Hamid and Benita Hall. 1994. Safety Propertiers. MORSE external deliverable D22, Doc. Id. MORSE/LLOYD'S/HAL/44/V1.
- [31] Lowden, Barry G T and Anne De Roeck. 1985. Generating English paraphrases from relational query expressions. Behaviour and Information Technology. 4(4). 337-348.
- [32] Macias, Benjamin. 1993. Use of CLARE to analyse specifications in English-like languages. MORSE External Deliverable D24, Doc. Id. MORSE/CU/BM/4/V2.
- [33] McKeown, Kathleen, Karen Kukich, and James Shaw. 1994. Practical Issues in Automatic Documentation Generation. Proc Assoc. Comp. Ling (ACL), 7-14.
- [34] Miriyala, Kanth and Mehdi T Harandi. 1991. Automatic Derivation of Formal Software Specifications from Informal Descriptions. IEEE Trans Soft Eng, vol 17, no 10, 1126-1142.
- [35] Ogden, William C. 1988. Using Natural Language Interfaces. In Martin Heider, (ed). Handbook of Human-Computer Interaction. Amsterdam. North-Holland. 281 - 299.
- [36] Payette, Julie and Graeme Hirst. 1992. An Intelligent Computer-Assistant for Stylistic Instruction. Computers and the Humanities 26 (2). 87 - 102.
- [37] Peter, Gerhard and Dietmar Rösner. User-Model-Driven Generation of Instructions. 1994. User Modeling and User-Adapted Interaction. vol 3, 289-319.
- [38] Robin, Jacques. 1993. A revision-based generation architecture for reporting facts in their historical context. In Helmut Horacek and Michael Zock (eds). New Concepts in Natural Language Generation. London, Frances Pinter. 238 - 268
- [39] Rösner, Dietmar. 1987. SEMTEX: A text generator for German. In Gerard Kampen (ed). Natural Language Generation: New Results in Artificial Intelligence, Psychology and Linguistics. Dordrecht, Martinus Nijhoff, 133 - 148.

- [40] Rösner, Dietmar and Manfred Stede. 1992. Customizing RST for the Automatic Production of Technical Manuals. In Robert Dale, Eduard Hovy, Dietmar Rösner, and Oliviero Stock (eds). Aspects of Automated Natural Language Generation. Berlin, Springer. 199 - 214.
- [41] RAISE Language Group. 1992. The RAISE specification language. Prentice Hall.
- [42] Sommerville, Ian. 1985. Software Engineering. Wokingham: Addison-Wesley.
- [43] Sterling, Leon S and Ehud Shapiro. 1986. The art of Prolog. Cambridge, Mass. MIT Press.
- [44] Tennant, Harry R, Kenneth M Ross, Richard M Saenz, Craig W Thompson, and James R Miller. 1983. Menu-Based Natural Language Understanding. Proc Assoc. Comp. Ling (ACL), 151 - 158.
- [45] Vander Linden, Keith. 1993. Generating Effective Instructions. Proc 15th Ann Conf Cog Sci Soc, 1023 - 1028.
- [46] Vander Linden, Keith. 1994. Generating Precondition Expressions in Instructional Text. Proc Assoc. Comp. Ling (ACL). 42 - 49.



