Editing logically complex discourse meanings

Rodger Kibble, Richard Power, Kees van Deemter

Information Technology Research Institute
University of Brighton
U. K.
FirstName.LastName@itri.brighton.ac.uk

Abstract

WYSIWYM (What You See Is What You Meant) is a new methodology for editing knowledge representations by interacting with a generated NL *feedback text* which presents both the knowledge already defined and the options for extending and modifying it. This paper concerns novel problems that arise when this methodology is extended to deal with certain familiar and supposedly well-understood semantic phenomena. The basics of WYSIWYM editing are briefly explained, followed by discussion of the problems to do with coreference and quantification and some partial solutions, including an experimental DRT-based prototype. Then we turn to more difficult problems in the area of quantification and sketch some directions in which solutions may be found.

Keywords: knowledge representation, natural language generation, underspecification

1 Introduction

Many applications of Artificial Intelligence require editing of information expressed in a knowledge representation formalism of some sort. Expert systems are an obvious example; others are systems for generating documents, and for encoding design specifications. With most currently available support tools, knowledge editing has to be performed by knowledge engineers who are familiar with the representation formalism. Elsewhere (Power & Scott 1998, Power et al 1998), a new knowledge-editing method called 'WYSIWYM editing' has been introduced and motivated. WYSIWYM editing allows a domain expert to edit a knowledge base reliably by interacting with a generated NL *feedback text* which presents both the knowledge already defined and the options for extending and modifying it.

WYSIWYM is a potentially powerful tool in many domains including some that are not commonly thought of as involving the editing of a knowledge representation formalism. Examples are the design of question-answering systems and interface design for complex technical equipment. Applications of various kinds are currently being investigated in a

number of projects at ITRI in the University of Brighton. Among the domains studied are software documentation, pharmaceutical information (as expressed in Patient Information Leaflets, for example) and maritime law.

Until recently, however, existing versions of WYSIWYM systems were very limited in the expressive power of the knowledge representation formalism used to express the information edited. The DRAFTER-II system (Power et al, 1998), in particular, did not allow the construction of quantifiers, nor did it allow a proper treatment of coreferential information (i.e., recurring variables or constants). The present paper discusses some of the problems that arise when these shortcomings are remedied. The key issue is that of transparency: can we allow the author of the system to understand the content of the knowledge base, and the effect of each of the available editing operations? Problems of this kind go to the heart of what WYSIWYM stands for, namely: an effort to make knowledge editing easier. On the theoretical side, the treatment of logically complex information involves interesting questions in the domain of logic (*What are possible ways in which complex formulas can be built up from meaningful parts?*) as well as computational linguistics (*How can each of the meaningful parts be 'translated' into natural language?*). The present paper will touch upon questions of both kinds.

The plan of the paper is as follows. First, the basics of WYSIWYM editing will be briefly recapitulated (Section 2). Then we will explain some of the problems that result for WYSIWYM when its knowledge representations deal with coreference and quantification and we will sketch some partial solutions (Section 3). After an explanation of the limitations of WYSIWYM (Section 3.1) we will discuss the issue of coreference, and sketch how the problems in this area have been tentatively tackled in the DRAFTER-III system (Section 3.2). Then we will turn to the more difficult problems in the area of quantification and sketch some directions in which solutions may be found (Sections 3.3, 3.4). We conclude by highlighting some unresolved problems and drawing some general conclusions (Section 4).

2 Editing simple material by WYSIWYM

WYSIWYM is a method of editing knowledge bases (KBs) in which the user interacts with a *feedback text* generated by a natural language generation (NLG) system (e.g. Power and Scott 1998). The idea of WYSIWYM editing is to provide an interface to a KB that can be used easily by an author who is a domain expert but not necessarily an expert in knowledge representation. The feedback text serves two purposes: it shows the knowledge that has already been specified, and it shows the options for adding new knowledge. These options are marked by *anchors*. By clicking on an anchor, the user obtains a pop-up menu from which a semantic option can be selected. The acronym WYSIWYM stands for 'What You See Is What You Meant': a natural language text ('what you see') presents a KB that the author has built by purely semantic decisions ('what you meant').

This method was first implemented in DRAFTER-II (Power et al, 1998), a re-engineered version of DRAFTER (Paris et al, 1995) which is designed for use by the technical authors who produce software documentation. By interacting with the feedback text, the author

defines a procedure for performing a task, e.g. the task of saving a document in a word processor.

When a new KB is created, DRAFTER-II assumes that its root will be some kind of procedure. A procedure instance is created, and assigned an identifier (for internal use only), e.g. `u1`. The definition of the concept `procedure` specifies that every procedure has two attributes: a goal, and a method. The goal must be some kind of action, and the method must be a list of actions. This information is conveyed to the author through a feedback text

> `Achieve` **this goal** `by applying` *this method.*

with several special features.

- Undefined attributes are shown through anchors marked by the use of boldface or italics.
- A **boldface** anchor indicates that the attribute is obligatory: its value must be specified. An *italicized* anchor indicates that the attribute is optional.
- All anchors are mouse-sensitive. By clicking on an anchor, the author obtains a pop-up menu listing the permissible values of the attribute; by selecting one of these options, the author updates the KB.

Although the anchors may be tackled in any order, we will assume that the author proceeds from left to right. Clicking on **this goal** yields a pop-up menu that lists all the types of actions that the system knows about:

choose
click
.....
save
schedule

(to save space, some options are omitted), from which the author selects 'save'. Although apparently selecting a word, the author is really selecting an option for editing the KB. The program responds by creating a new instance, of type `save`, and adding it to the KB (as a Prolog assertion) as the value of the `goal` attribute on `u1`:

```
procedure(u1).
goal(u1, u2).
  save(u2).
```

From the updated KB, the generator produces a new feedback text

> `Save` **this data** `by applying` *this method.*

including an anchor representing the undefined `actee` attribute on the `u2` instance. Note that this text has been completely regenerated. It was not produced from the previous text merely by replacing the anchor **this goal** by a longer string. By continuing to make choices at anchors, the author might expand the KB in the following sequence:

- `Save the document by applying` *this method.*
- `Save the document by performing` **this action** (*further actions*).
- `Save the document by clicking on` **this object** (*further actions*).
- `Save the document by clicking on the button with` **this label** (*further actions*).
- `Save the document by clicking on the Save button` (*further actions*).

At this point the KB is potentially complete (no boldface anchors remain), so an *output text* can be generated and incorporated into the software manual.

> `To save the document, click on the Save button.`

To delete information, the author opens a pop-up menu on any span of the feedback text that presents a defined attribute. For instance, the span 'the document' presents the `actee` attribute on the instance `u2`. Clicking on this span in the feedback text, the author obtains the menu

Cut
Copy

If 'Cut' is selected, the instance that is currently the value of the `actee` attribute is removed to a buffer, leaving the attribute undefined. The resulting feedback text introduces an anchor in place of 'the document'.

> `Save` **this data** `by clicking on the Save button` (*further actions*).

When the buffer is full, the pop-up menus that open on anchors contain a 'Paste' option if the instance in the buffer is a suitable value for the relevant attribute.

3 Extending WYSIWYM to logically complex material

3.1 Limitations of DRAFTER-II

In our first experiment with WYSIWYM we made two simplifying assumptions: first, that corresponding descriptions always refer to the same entity; secondly, that no entities are hypothetical. In effect, the first assumption allows the author no control over coreference. If document instances are introduced at two locations in the KB, perhaps because a document is both saved and printed, the author cannot specify whether the saved and printed documents are identical or different. The second assumption also imposes a major limitation. It allows no way of constructing representations that include logical operators such as negation, quantification, and implication.

To understand the source of these limitations, we need to look more closely at the semantic formalism and associated editing operations used in the original WYSIWYM model. Figure 1 shows a semantic network of the kind used in DRAFTER-II to represent a procedure for saving a document. The procedure might yield the feedback text

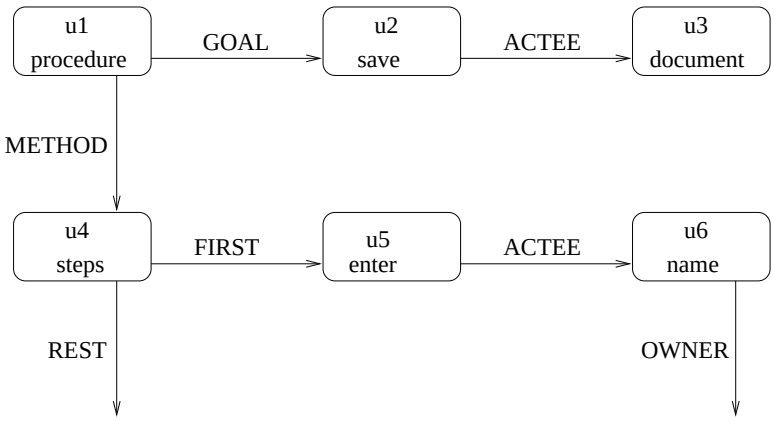


Figure 1: Representing a procedure in DRAFTER-II

`Save the document by entering the name of` **this object** (*further actions*).

The network diagram includes arcs showing where further information may be added: both the `rest` attribute on u4 and the `owner` attribute on u6 are currently unspecified.

In DRAFTER-II, the only way of adding information to the graph was by inserting a *new* node, of a selected type, as the value of a currently unspecified attribute. For instance, if the author chose 'document' on the anchor **this object**, a new node u7 was added as the value of the `owner` attribute on u6. The generated text gave the impression that u7 was coreferential with u3 (the document to be saved), but this was merely an artifact of the rules applied by the generator, which always expressed such entities by definite descriptions. Obviously this is unsatisfactory: if there is in fact only one document, the value of the `owner` attribute on u6 should be set to the existing document entity u3. The creation of a new entity u7 should imply that there are two separate documents.

Note that this limitation in DRAFTER-II did not arise from the semantic formalism itself. A network diagram like figure 1 can show whether the documents are identical or separate: in the former case a single document entity u3 has two incoming arcs; in the latter case, there are two document entities u3 and u7. The limitation arose rather from the editing operations that the program supported. In section 3.2 we describe a proposal for extending the set of editing operations so as to allow control of coreference.

The problem of logical complexity is altogether more difficult: it cannot be overcome by defining more editing operations on the same formalism. It has been known for a long time that semantic networks like figure 1 cannot encode generalizations about hypothetical entities (i.e. statements with quantifiers) unless they are supplemented by some kind of scoping device. Hendrix (1979) achieved this objective by partitioned networks in which the nodes and arcs are assigned to 'spaces'. We are currently exploring a similar idea based on Discourse Representation Theory (Kamp and Reyle, 1993; Asher, 1993).

If we consider the semantic network in figure 1 in the light of DRT, its limitations become obvious. It consists essentially of a set of discourse referents together with a

set of simple conditions: there are no complex conditions. Such a network would suffice only for the special case of a DRS composed of a single box, K0. This is not a suitable representation for an instructional text, which ought to include both conditional and modal operators. The meaning we want to attach to figure 1 is that *if* the reader wants to save a document (any document), various actions *should* be performed. The document referent and all the action referents should properly be bound to subordinated boxes, not to K0. We were able to get away with this simplification in DRAFTER-II only because the text did not require a distinction between real past actions and hypothetical future ones. In section 3.4 we discuss some technical difficulties in extending WYSIWYM editing to a representation based on DRT.

3.2 Coreference

Let us turn to the first addition to the DRAFTER-II knowledge representation language that gives rise to problems of transparency. We have seen in the previous section that DRAFTER-II did not allow the author to control coreferentiality. In an earlier paper (van Deemter & Power, 1998), we have sketched how the DRAFTER-III program comes to terms with this shortcoming. Whenever the author wants to expand an anchor whose type is compatible with that of one or more other expressions in the feedback text, the system asks whether the newly created instance is coreferential with a previously-created instance. For example, imagine the author wants to expand an anchor of the form 'this object', specifying that it is a document. Imagine, furthermore, that two other documents have already appeared in the feedback text, namely $(\text{document})_1$ and $(\text{document})_2$. Then the newly introduced object may corefer with $(\text{document})_1$ or with $(\text{document})_2$ or with neither. In order to avoid asking potentially superfluous questions, the system first presents the user with a menu containing the two options

Existing object?
New object?

Only in case of the first choice will the system follow up with the menu

$(\text{document})_1$?
$(\text{document})_2$?

In case of the second choice, the system will replace the anchor by some expression of the form $(\text{document})_i$, where i is a new index. Note that indices are added to the feedback language to avoid ambiguities in the feedback text but also, perhaps more crucially, in the menu. The menu has to contain simple expressions which are easy to understand and which do not have the full benefit of a potentially disambiguating linguistic context. (Each of the expressions in the menu may appear several times in the feedback text.) Hence it is not easy to see how indices (or other artificial devices, such as color coding, for that matter) can be avoided under all circumstances.

The issue of distinguishing between different possible antecedents in the menus as well as in the generated texts — which already invites deviations from natural language, as we have seen — by no means exhausts the potential problems posed to knowledge editing

by coreference. The most difficult problems that we encountered in this area have to do with the knowledge editing itself: Now that a given instance in the KB can have several occurrences in the KB, operations like cutting and copying no longer have one single possible interpretation. To illustrate, let us look briefly at the *cut* operation.
In DRAFTER-II, cutting is a conceptually simple operation that does not allow variations: an attribute value is replaced by its anchor. When coreference enters the picture, this is no longer the case. There are two questions that have to be addressed when the information associated with an indexed NP is cut. In what follows, we will simplify and say, by a slight abuse of language, that *an NP* is cut, rather than the information associated with it. (See van Deemter & Power 1998 for explanation on what information this is, in the different cases that can arise.) The question are, firstly, *Does the author intend to cut this NP alone, or does she intend to cut all NPs with the same index as this NP?* Secondly, *if* the author intends to cut all expressions with the same index, then *Does the author intend the system to respect the indices?* In other words, does she assume that the anchors that will result when the NP is de-selected must all be filled by NPs that have the same index? Depending on how these questions are answered, three different variants of the Cut operation arise. Let α_i be the NP on which the author has clicked. Then

- *Cut-one* only affects this occurrence of α_i.
- *Cut-all* affects all occurrences of α_i, also severing all coreference links between their anchors.
- *Cut-all*^{c} affects all occurrences of α_i, respecting all coreference links between their anchors.

Consider our earlier example, adapted to the situation where `u3` is the value of two different attributes:

The semantic network is implemented by a set of Prolog assertions:

```
procedure(u1).
goal(u1, u2).
  save(u2).
  actee(u2, u3).
    document(u3).
method(u1, u4).
  steps(u4).
  first(u4, u5).
    enter(u5).
    actee(u5, u6).
      name(u6).
      owner(u6, u3).
```

To spell out the effect of the new operations, it will be convenient to use variables. If *Cut-one* is applied to the occurrence of `u3` in the `owner` attribute then the only effect on the content of the KB is that the `owner` of `u6` is undefined, which may be represented by

a variable, say x. If, instead, *Cut-all* is applied to the same instance of `u3`, then both the `owner` of `u6` and the `actee` of `u2` are undefined, which may be represented by using two different variables, say x and y. If, finally, *Cut-all*^{c} is applied then, once more, both are undefined, but their values must be equal. This may be represented by using the same variable, say x, in both cases: `actee(u2, x)`, `owner(u6, x)`.

The DRAFTER-III program has implemented the ideas outlined here, as well as a natural language generator that takes the new representations as input and transforms these into texts. Thinking through all the implications of this revised system has been surprisingly difficult, however, and it remains to be seen whether the concept of coreference (and the way we have tried to make it transparent, using indices) is transparent to the domain authors for whom WYSIWYM is intended. The experience, in the MUC 6 and 7 projects (Hirschman et al, 1997), of *annotating* for coreference has shown that the concept is not always clear, even for experienced annotators. Thus, it is less than obvious what coreference means in relation to intensional contexts. For example, it seems that the bracketed NPs in the following feedback text should receive different indices, even though their referent is equal:

`Make sure that (the date of the program) equals (the current date).`

In these cases, it is difficult to see how indices can be used. The same is true whenever a reference marker fails to correspond with a single constituent in the text, as in the case of abstracted or summated reference markers (See e.g. Kamp & Reyle 1993: 306-314). Note, however, that the phenomena mentioned here may not pose a serious problem for the construction of *feedback* texts, which can be restricted to make a simplified fragment of language where the notion of coreference may not be problematic.

3.3 Logical complexity

In DRAFTER-II and DRAFTER-III, editing operations are defined on a semantic network representation. Our aim in this section is to explore ways in which these systems could be extended so as to allow the construction of logically complex meanings. We would like to do this by extending the semantic network formalism rather than by introducing a completely new knowledge representation, so that the editing operations already mentioned (Insert, Cut, Copy and Paste) can be preserved.

A fairly simple extension of the semantic network formalism has been implemented in an experimental program in the domain of patient information leaflets (PILs). Three innovations are proposed over the semantic model employed in DRAFTER-III:

- The nodes of the network are taken to represent discourse referents, in the DRT sense. Since nodes may correspond to actions as well as individuals, a result of this is that the meanings of verbs are represented as one-place predicates on eventualities, for example *consult(u6)*.

- We allow some nodes of the network to represent complex conditions such as implications, obligations and negations. Such nodes will be called *abstract referents*.

During editing, a node representing a negation will be introduced in the usual way as a token of this abstract type.

- Every node of the network is bound to a DRT box. The root of the network must be bound to K0; other nodes may be bound to subordinated boxes introduced by abstract referents.

As an example, figure 2 shows a DRT representation of a typical sentence from a patient information leaflet (1); figure 3 shows the equivalent graph representation.

(1) If you are pregnant, consult your doctor

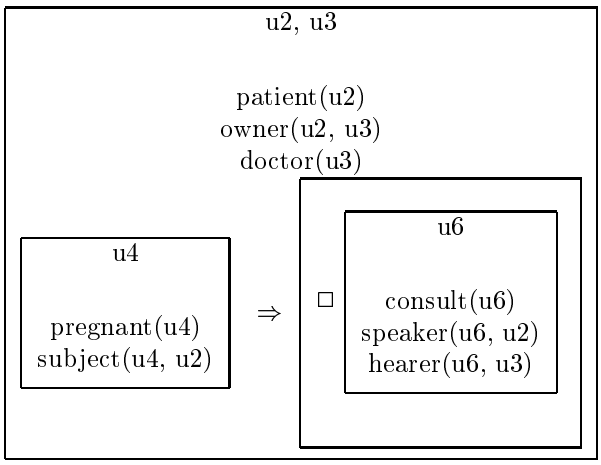


Figure 2: DRT representation for a logically complex sentence

3.4 Handling Quantifiers

From the point of view of knowledge editing, the DRT-based formalism just outlined still has some drawbacks.

1. To edit the KB via the feedback text it's important that KB elements correspond to an identifiable span of text. This isn't always the case where the KB is encoded as a DRS. For example, suppose the current feedback text is 'every packet contains 30 tablets' and the author wants to cut 'every packet' to replace it with some other quantity; the system can't straightforwardly excise this part of the content leaving a proper DRS.

2. Related to this is the lack of a consistent editing sequence for quantified and non-quantified statements. A natural procedure might be to allow the author to first

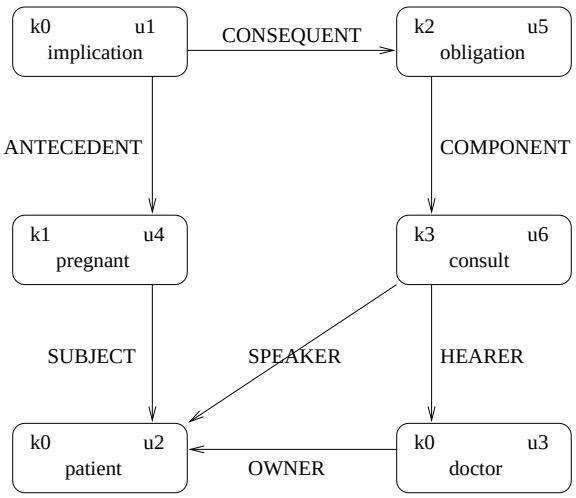


Figure 3: Extended semantic network including complex conditions

select a predicate and then specify the arguments, as in the DRAFTER examples presented in section 2. However, if any argument is a quantified expression it is necessary to construct the quantificational schema before selecting the main predication. This is not a natural way to compose a text and authors are likely to find it confusing.

3. The fact that DRSs have to be constructed 'top-down' makes the author responsible for determining quantifier scope in statements with multiple quantifiers. Again, this is likely to confuse authors who have no training in logic.

In this section we consider whether some of these difficulties can be resolved by adopting a 'flatter' formalism such as Quasi-Logical Form (Alshawi & Crouch 1992), U(nderspecified)DRT (Reyle 1996) or Minimal Recursion Semantics (Copestake et al MS). These systems are motivated in part by the goal of 'underspecifying' certain semantic distinctions such as quantifier scope, though the attraction from our point of view is that they appear to offer greater flexibility in the task of knowledge editing. This will be illustrated with the aid of a simple example in the framework of UDRT. (Note that what follows is programmatic, and does not report on implemented work.) UDRT differs from standard DRT principally in that U(nderspecified)DRS components are not depicted as nested inside each other, but as discrete representations partially ordered by a subordination relation written as $\leq$. In Reyle's presentation the upper and lower bounds of the subordination relation are the clause boundary and the condition corresponding to the main verb. In the simple examples below we only show one clause, so the top box K0 is treated as the upper bound. Some potential advantages of UDRT are:

1. The DRS can be constructed 'inside out', i.e. the author can start by selecting a predicate regardless of whether the arguments are quantified. A DRS-condition

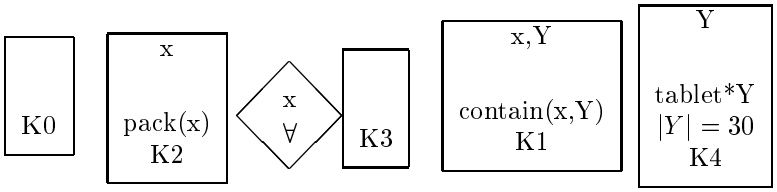
created in one editing step may turn out to be subordinated to the 'scope' portion of a quantificational schema created in a subsequent step.

2. There is a better likelihood of each KB element corresponding to a separate span of feedback text; for instance a quantificational schema is represented separately from the content of its nuclear scope (this should become clearer in the example below).

3. More flexibility for scoping, including leaving some scope relations unspecified. In fact it's not clear whether we need or want to exploit this, despite the fact that this was the prime motivation for the development of UDRT.

A UDRS can be constructed monotonically by adding components and/or subordination constraints, so in principle an author should be free to select elements in any order.The following shows a possible sequence of feedback texts at selected stages of the composition of a target text "Every pack contains 30 tablets", with the resulting UDRS shown in Figure 4. (Note that in these texts **something** indicates an unfilled argument slot and does not imply existential quantification; the subscripts identify these slots with discourse referents in the UDRS.)

- **Eventuality**
- **Something**$_x$ `contain/s` **something**$_y$
- `Every` **thing**$_x$ `contains` **something**$_y$
- `Every pack contains` **something**$_y$
- `Every pack contains 30` **things**$_y$
- `Every pack contains 30 tablets`



$K1 \leq K3,\ K2 \leq K0,\ K4 \leq K3$

Figure 4: Underspecified DRS for *Every pack contains 30 tablets*

At this stage we have a complete, fully scoped UDRS which is equivalent to the more familiar looking DRS in Figure 5. The constraints indicate that sub-DRSes K1 and K4 are subordinated to K3 and so are within the nuclear scope of the quantifier. Note that if $K4 \leq K3$ is replaced by $K2 \leq K4$ then referent Y is effectively introduced in the top box, taking wide scope.

An advantage of the formalism is that UDRS components remain separately represented and can be excised cleanly without perturbing the rest of the structure. Figure

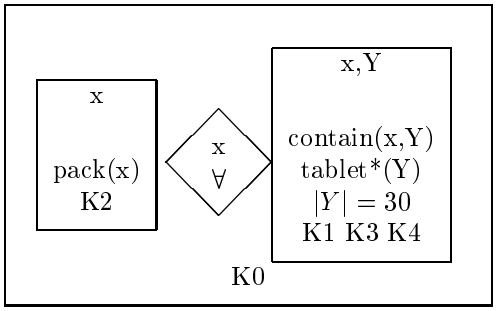
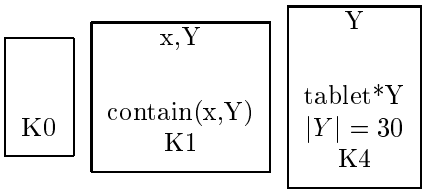


Figure 5: Standard DRS for *Every pack contains 30 tablets*

6 shows the result of 'cutting' "every pack": the duplex condition is removed as well as constraints mentioning K2 or K3[1], leaving K1 and K4 in place. This would generate the feedback text "**Something** contain/s 30 tablets".



$K1 \leq K0,\ K4 \leq K0$

Figure 6: UDRS for "**Something** contain/s 30 tablets".

Scope and underspecification

Ideally we wish to avoid asking an author to make decisions about quantifier scope where these can be made by the system. We envisage various strategies for resolving scope; one possibility is for scope to be implicitly determined by the order of argument selection, another is to include axioms in the database for certain stereotypical situations. For instance in the domain of patient information leaflets we frequently find instructions to take a certain amount of medication at a specified frequency, or information that a package contains a certain quantity of medication. So we could encode rules such as the following, where `A > B` is read as "A outscopes B".

```
take(Patient, Medication-Units, Frequency) -->
    PATIENT > FREQUENCY > MEDICATION-UNITS

contain(Package, Medication-Units) -->
    PACKAGE > MEDICATION-UNITS .
```

[1]Note that $K4 \leq K0$ was not shown in the previous diagram as it was redundant.

A third possibility is to leave scope relations undetermined in the KB and rely on linguistic rules of preference to favour a particular reading. There has been a great deal of work on underspecification in interpretation of natural language (see e.g. papers cited above and contents of van Deemter & Peters (eds) 1996). A question which now concerns us is whether there's a place for underspecified input in NLG. which is one of the applications for which WYSIWIM has been developed. Output text will inevitably be ambiguous in some respect; the question is whether it is necessary or worthwhile to enforce distinctions in the KB which will be lost in the generated text. We don't take a stand on this issue but note some of the issues:

- Often the scope makes no difference to the import of a message:

  (2) Take 2 tablets every day with a glass of water.

  It is obviously essential that 'every day' outscopes 2 tablets, but it probably makes no difference whether the patient takes both pills at once with one glass of water, or takes them separately refilling his glass for the second pill. Authors would be understandably annoyed if the system forced them to make a decision.

- Expressibility: The theoretical upper bound of scope ambiguity is the factorial of the number of quantifiers (and other operators) in a sentence, so (3) has 24 different readings. It's questionable whether all these options can be expressed unambiguously and comprehensibly in a NL feedback text.

  (3) Most patients should take 2 tablets every day with a glass of water.

- Supposing we do allow underspecified input, there is a potential danger that the generator could produce an output text where the preferred interpretation is an unintended one. More needs to be known about linguistic rules of preference; relevant work includes studies by researchers at SRI (Moran & Pereira 1992) on quantifier scope and van der Sandt (1992) on anaphora resolution.

4 Conclusion

We have basically done three things in this paper. *Firstly*, we have recapitulated some of the main ideas behind WYSIWYM knowledge editing. Doing this, we have stressed WYSIWYM's use of generated natural language to clarify the content of the KB and we have explained some limitations of previous WYSIWYM-based systems, pointing out how they were unable to come to grips with certain kinds of information. *Secondly*, we have sketched two extensions of WYSIWYM which are intended to remedy these shortcomings. One of these allows the author of a WYSIWYM system to control whether or not two instances in the knowledge base are coreferential or not. The other extension, which causes the semantic nets underlying WYSIWYM to resemble Discourse Representation Structures, allows the expression of quantified information. *Thirdly*, we have discussed a number of shortcomings that this 'DRT-based' extension is likely to have from the point of view of an

author who is not an expert in logic or knowledge representation. Among the unresolved questions raised in the paper are the following:

1 Is natural language always an appropriate medium for expressing logically complex information? (Cf. e.g. Reiter 1997 for a related discussion.) Our experience with the treatment of coreference suggests that there may be a trade-off between naturalness and clarity, and that it might be advisable to adopt, in the feedback expressions, a limited number of 'unnatural' devices such as indices, brackets, etc.

2 What is the most convenient regime for allowing an author to build up a logically structured text? Our discussion in section 3.4 suggests that ideally, the author should be completely free as to the order in which constituent are added to (or removed from) the feedback texts. As we have pointed out, underspecified representation languages such as UDRT might be used to make this possible.

3 What types of ambiguity are allowed in feedback texts? One possibility would be to disallow all expressions that are ambiguous (when world knowledge is taken into account). Another possibility is to rely on the rules of preference that have been proposed in the literature (e.g., papers cited in section 3.4) and to require that the *most preferred* interpretation of a feedback text) that is consistent with world knowledge) should be the intended interpretation. Both strategies, however, may be refined by disregarding ambiguities that lead to valid instructions on each of their interpretations.

An unexpected aspect of points (2) and (3) is the role of underspecification. Usually, underspecification is thought to be relevant for natural language *interpretation* only, except in machine translation, where underspecified representations stemming from interpretation of the source language can be the input to the generator for the target language (e.g. Alshawi 1992). Some of the more speculative parts of this paper, however, suggest yet another possible role for underspecification in relation to natural language *generation*.

Acknowledgements

Thanks to Paul Piwek for helpful comments on an earlier draft of this paper. This work is supported in part by the UK Engineering and Physical Sciences Research Council (EPSRC) under grant references GR/L51126 (Kibble) and GR/M36960 (Power).

References

Alshawi, H. (Ed.) 1992, *The Core Language Engine*, MIT Press, Cambridge Mass.

Alshawi, H & R Crouch, 1992, Monotonic Semantic Interpretation, in *Proc ACL 1992*, pp 32-39.

Asher, N, 1993, *Reference to Abstract Entities in Discourse*, Kluwer, Dordrecht.

Copestake, A, D Flickinger & I Sag (MS), Minimal Recursion Semantics: An Introduction, CLSI, Stanford.

Hendrix, G, 1979, Encoding Knowledge in Partitioned Networks, in N Findler (ed), *Associative Networks: Representation and Use of Knowledge by Computers*, pp 51-92..

Hirschman, L, P.Robinson, J.Burger, and M.Vilain, 1997, Automating Coreference: The Role of Annotated Training Data, in *Proceedings of the AAAI Spring Symposium on Applying Machine Learning to Discourse Processing.*

Kamp & Reyle 1993, *From Discourse to Logic*, Kluwer, Dordrecht.

Moran, D, and F.C.N.Pereira. 1992, Quantifier Scoping. In Hiyan Alshawi (Ed.) *The Core Language Engine*, MIT Press, Cambridge Mass..

Paris, C, K Vander Linden, M Fischer, A Hartley, L Pemberton R Power & D Scott, 1995, A Support Tool for Writing Multilingual Instructions, in *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, Montreal, pp 1398-1404.

Power, R, & D. Scott, 1998, Multilingual authoring using feedback texts, in *Proceedings of the 17th International Conference on Computational Linguistics and 36th Annual Meeting of the Association for Computational Linguistics*, Montreal, pp 1053-1059.

Power, R, D. Scott & R. Evans, 1998, What You See Is What You Meant: direct knowledge editing with natural language feedback, in *Proceedings of the 13th Biennial European Conference on Artificial Intelligence*, Brighton, pp 675-681.

Reiter, E, 1997, Choosing a Media for Presenting Information, *News Journal on Intelligent User Interfaces*
(http://www.dfki.de/ andre/etai/nj/backissues/inbodyN1.html#gdebate)

Reyle, U, 1996, Co-indexing Labeled DRSs to Represent and Reason with Ambiguities, in van Deemter & Peters (eds), pp 239 - 268.

van Deemter & Peters (eds) 1996, *Semantic Ambiguity and Underspecification*, CLSI Publications, Stanford.

van Deemter & Power 1998, Coreference in Knowledge Editing, in Proceedings of the COLING-ACL workshop on the Computational Treatment of Nominals, Montreal, pp 56-60

Van der Sandt, R, 1992, Presupposition Projection as Anaphora Resolution, *Journal of Semantics*, **9**, 333-377.