Towards Incremental Semantic Annotation*

Diego Mollá
Macquarie University
Marsfield, Sydney NSW 2109, Australia
Diego.Molla-Aliod@mq.edu.au

Abstract

In this paper it is argued for the use of a type of notation that allows the incremental expression of semantic information. A sound incremental semantic annotation must be able to be extended without deleting any of the previous annotations in the same text. This way one can specify the semantic contents of textual information according to the resources and time available. To achieve this sort of incremental extensibility, the use of a flat notation is proposed by reifying all the predicates and operators in the (possibly nested) original expression. One such notation is introduced, and it is shown how the notation can express semantic information with the detail that matches our needs. Some interesting properties of this notation are also indicated. Finally, it is shown how this notation helps in the task of pin-point information retrieval by answer extraction, based on ExtrAns and WebExtrAns, two existing answer extraction systems.

1 Introduction

Incremental extensibility is necessary in any type of semantic annotation for several reasons. First, in many cases we will not know how to syntactically analyse certain strings because of structural ambiguity (example: What is the intended bracketing of *computer language analysis*?). Second, we will often not know how to semantically analyse it (in what semantic relationship do "language" and "analysis" stand in "language analysis"?). And even if we can specify the full semantics without ambiguities we may find it unnecessary (or too expensive) to actually do it. In all these cases we must somehow represent underspecified relationships. And in all cases we may later want to add constraints, either because we acquired information later that allows us to disambiguate a structure or assign it a meaning, or because we decided we needed a certain distinction after all.

In this paper we discuss a notation for semantic information that is inherently incrementally extensible, thus allowing the annotator to focus on the specific types of semantic information that are required for the task at hand. Section 2 introduces the notation. The main idea in this notation is the flattening out of the (possibly nested) original expression by reifying all the predicates and operators in the expression. Section 3 shows the ability of these logical forms to express partial semantic information and how more information can be added without having to destroy old information. Finally, Section 4 shows how an NLP application, answer extraction, can take advantage of this notation, based on our experience with a real answer extraction system.

2 Flat Logical Forms

2.1 The basic idea

The basic idea is to do like computer programmers do when the programming language only allows a function to return the result in one of the arguments: use one of the arguments to collect the result. For example, the instruction (1a) can be substituted by the sequence of instructions (1b), where the functions `factorial` and

*This paper is an adaptation of the paper "Ontologically Promiscuous Flat Logical Forms for NLP" presented at IWCS-4, Tilburg, The Netherlands, on January 2000.

`exp` are modified to make them return the value in their first argument:

(1) a.
```
A := (factorial(25)-exp(12)) * 2;
```

b.
```
factorial(X,25);
exp(Y,12);
A := (X-Y) * 2;
```

If the programmer wants to redefine subtraction and multiplication, then (1b) could even be translated into:

(2)
```
factorial(X,25);
exp(Y,12);
subtract(P,X,Y);
multiply(A,P,2);
```

The resulting expression (2) is a completely flat form that produces the same result as the nested form (1a).

2.2 Ontological promiscuity

A key technique to flatten out a (nested) logical form is the use of reification. To reify is:

> To convert into or regard as a concrete thing: *to reify an abstract concept*
> The Macquarie Concise Dictionary

We can reify partial expressions and logical operators that appear in the logical form. A consequence of the reification of an expression is the introduction of a new entity, the *reified entity*. In our analogy with computing programs, these reified entities are the "results" of the partial expressions.

The reified entities can be seen as "handles" to refer back to the reified expression. Let us start with a preliminary example:

(3) a. *John ate an apple quickly*

b. $\exists a(\texttt{quick}(\texttt{eat}(j,a)) \wedge \texttt{apple}(a))$

c. $\exists a, e_1(\texttt{eat}(e_1,j,a) \wedge \texttt{quick}(e_1) \wedge \texttt{apple}(a))$

The logical form (3c) is a flat form of (3b). To obtain it, we have reified the eating event and by so doing the predicate `quick` introduced by the adverbial *quickly* does not need to take the predicate `eat` as its argument. Instead, the argument is the reified entity e_1 that represents the eating event. Reifying events is a common practice in Davidsonian approaches (Davidson 1967; Parsons 1985),[1] but we go further and reify all the predicates. This is Hobbs' ontological promiscuity (Hobbs 1985b). The flat form of (3b) becomes:

(5) $\exists a, e_1, e_2, e_3(\texttt{eat}(e_1,j,a) \wedge \texttt{quick}(e_2,e_1) \wedge \texttt{apple}(e_3,a))$

By using ontological promiscuity, one can easily express sentences with adverb modifiers (or any other type of modifiers):

(6) a. *John ate a pale green apple very quickly*

b. $\exists a(\texttt{very}(\texttt{quick}(\texttt{eat}(j,a))) \wedge \texttt{apple}(a) \wedge \texttt{pale}(\texttt{green}(a)))$

c. $\exists a, e_1, e_2, e_3, e_4, e_5, e_6(\texttt{eat}(e_1,j,a) \wedge \texttt{quick}(e_2,e_1) \wedge \texttt{very}(e_6,e_2) \wedge \texttt{apple}(e_3,a) \wedge \texttt{green}(e_4,a) \wedge \texttt{pale}(e_5,e_4))$

The relation between ontological promiscuity and the programming technique discussed in the introduction can be seen quite clearly by comparing the original nested form (6b) and the flat form (6c) with (1a) and (2).

2.3 Existence

A potential complication of the proposed notation is the scope of existential quantification. Since now all the variables have existential quantification with the widest possible scope, what happens with entities that should appear in a narrower scope, as is the case with opaque verbs, negation, or implication?

[1] It may be argued that actually Davidson's events and our reification system belong to different domains. In fact, we may need to use *both* the event as a new independent entity (s below) and the predicate reification (e_1 below):

(4) a. *John sleeps deeply*
$\exists e_1, s(\texttt{sleep}(e_1, \boxed{s}, j), \texttt{deep}(\boxed{s}), \texttt{john}(j))$

b. *John probably sleeps*
$\exists e_1, s(\texttt{sleep}(\boxed{e_1}, s, j), \texttt{probable}(\boxed{e_1}), \texttt{john}(j))$

This possibility, however, is not considered further in this paper.

Hobbs himself addresses this complication by making the universe of quantification an imaginary universe that contains everything that one can think of. This is what Hobbs calls the "Platonic universe". Those entities that also exist in the universe of discourse are specifically marked by a new predicate, say, `Rexists`. Thus, if an entity is the argument of `Rexists`, then it exists in the universe of discourse. If an entity is not the argument of `Rexists`, then we do not know if it exists in such universe or not. This can be seen in the example:

(7) *John wants to fly*
$\exists e_1, e_2(\texttt{want}(e_1, j, e_2) \wedge \texttt{fly}(e_2, j) \wedge \texttt{Rexists}(e_1))$

The existence of e_1 (`want`) is asserted, but the existence of e_2 (`fly`) is not asserted. Therefore, it is not known whether e_2 exists in the universe of discourse. One needs to use extra axioms (based on our general knowledge of wanting and flying) that allow us to infer that, say, e_2 does not exist in the universe of discourse. The logical form (7) is therefore underspecified with respect to the existence of e_2 in the universe of discourse.

Hobbs relates a non-reified predicate with its corresponding reified predicate (marked with $'$ in his notation) by means of the axiom:

Axiom 1

$\forall x_1, \cdots, x_n[p(x_1, \cdots, x_n) \equiv \exists e(\texttt{Rexists}(e) \wedge p'(e, x_1, \cdots, x_n))]$

This axiom states that an unprimed (non-reified) predicate is equivalent to its primed (reified) predicate plus the assertion that the reified entity exists in the universe of discourse. Thus, any unprimed predicate can always be expressed by means of a primed predicate plus `Rexists`. However, a primed predicate cannot necessarily be expressed by means of an unprimed predicate, since a primed predicate by itself does not ensure that the reified event exists in the universe of discourse.

Logical operators would be liable to reification just in the same way as predicates. A possible (simplified) expression in Hobbs' notation of a sentence with negation and implication would be:

(8) *if John is a bachelor, then he is not married*
$\exists e_1, e_2, e_3, e_4(\texttt{bachelor}(e_1, j) \wedge \texttt{married}(e_2, j) \wedge \texttt{not}(e_3, e_2) \wedge \texttt{if}(e_4, e_1, e_3) \wedge \texttt{Rexists}(e_4))$

This notation is clearly different from a traditional non-flat expression that uses logical operators:

(9) *if John is a bachelor, then he is not married*
$\texttt{bachelor}(j) \longrightarrow \neg\texttt{married}(j)$

The question is, if we assume (as Hobbs does) that the logical operators $\longrightarrow$ and $\neg$ are equivalent to `if` and `not`, can Axiom 1 make (8) and (9) equivalent? Note that Axiom 1 would introduce existential quantification inside the components of the implication:

(10) *if John is a bachelor, then he is not married*
$(\exists e_1(\texttt{bachelor}(e_1, j) \wedge \texttt{Rexists}(e_1))) \longrightarrow \neg(\exists e_2(\texttt{married}(e_2, j) \wedge \texttt{Rexists}(e_2)))$

Hobbs (Hobbs 1985a) proposes the use of specific axioms that allow inferences with logical operators. However, these axioms become very difficult to use when we have a combination of operators. The case of embedded existential quantifiers becomes especially difficult, and therefore the equivalence between (8) and (9) is not clear.

One may argue (as Hobbs might) that (9) is not an adequate logical form of the example sentence, and therefore (8) and (9) ought to be different. However, we would like to keep the possibility of (9) or similar being adequate. Since Hobbs' combination of `Rexists` and Axiom 1 does not allow us to establish a logical equivalence relation between a nested form and its flat form, we do not use Axiom 1 in our notation. To avoid confusion and distinguish our notation from Hobbs', to express existence in the universe of discourse we will use `True` instead of `Rexists`.

What we propose is this: Reify all the predicates and logical operators in the logical expression, and add `True` for the entity that results of reifying the top level operator. To ease readability, the expressions in the rest of this paper will not show the existential quantification over the reified entities (it is always going to have the

widest possible scope), and the predicates are conjoined by a comma, instead of $\wedge$. Thus, our proposed representation of (9) becomes:

(11) *if John is a bachelor, then he is not married*
$\texttt{bachelor}(e_1, j)$, $\texttt{married}(e_2, j)$, '$\neg$'(e_3, e_2), '$\longrightarrow$'(e_4, e_1, e_3), $\texttt{True}(e_4)$

Note that even existential quantifiers over non-reified variables are subject to reification.[2] The final version of (3a) *John ate an apple quickly* becomes:

(12) $\texttt{eat}(e_1, j, a)$, $\texttt{quick}(e_2, e_1)$, $\texttt{apple}(e_3, a)$, '$\wedge$'(e_4, e_2, e_3), '$\exists$'(e_5, a, e_4)

3 Incrementality

An important feature of our flat forms is that it is possible to use them to express partial information. Consider, for example:

(13) *John may have wanted to eat all the apples in the basket*

This sentence contains, among others, a modal verb, a complex tense, an embedded clause, a quantifier, and a plural. It may be difficult to provide a correct logical form for the sentence. Let us assume that there is an agreement about the logical form of (13), say:

(14) $\texttt{all}(x, \ \texttt{apple}(x) \wedge \texttt{in}(x, b) \longrightarrow \texttt{may}(\texttt{perfect}(\texttt{want}(j, \texttt{eat}(j, x)))))$

It is fairly easy to find the corresponding flat form:

(15) $\texttt{True}(e_1)$, $\texttt{all}(e_1, x, e_2)$, '$\longrightarrow$'$(e_2, e_3, e_4)$, '$\wedge$'$(e_3, e_5, e_6)$, $\texttt{apple}(e_5, x)$, $\texttt{in}(e_6, x, b)$, $\texttt{may}(e_4, e_7)$, $\texttt{perfect}(e_7, e_8)$, $\texttt{want}(e_8, j, e_9)$, $\texttt{eat}(e_9, j, x)$

[2]The reification of existential quantifiers over reified entities is also possible, even the reification of the True predicate. But this would introduce new reified entities without any apparent advantages.

Given (15), it is trivial to produce a partial representation of it, one need just remove some of the predicates. For example, the following partial logical form could be used to annotate the semantics of the sentence:

(16) $\texttt{eat}(e_9, j, x)$, $\texttt{apple}(e_5, x)$, $\texttt{in}(e_6, a_1, b)$

The new variable a_1 is introduced to express that the attachment of the PP is not resolved (as it could be either x or e_9). It is trivial to see that (16) provides partial information with respect to the complete logical form (15). We would say that (16) is a partial logical form of (15).

An example of the use of incrementality is the case of scope underspecification. Thus, the sentence *every man dates some woman* has two readings. The most natural reading is the one where every man dates a different woman:

(17) a. *every man dates some woman*

b. $\texttt{every}(x, \texttt{man}(x) \longrightarrow \texttt{some}(y, \texttt{woman}(y) \wedge \texttt{date}(x, y)))$

c. $\texttt{man}(e_1, x)$, $\texttt{woman}(e_2, y)$, $\texttt{date}(e_3, x, y)$, $\texttt{every}(e_4, x, e_9)$, '$\longrightarrow$'$(e_9, e_1, \boxed{e_6})$, $\texttt{some}(e_6, y, e_{10})$, '$\wedge$'$(e_{10}, e_2, \boxed{e_3})$, $\texttt{True}(\boxed{e_4})$

There is, however, another reading where there is one woman such that every man dates her:

(18) a. *every man dates some woman*

b. $\texttt{some}(y, \texttt{woman}(y) \wedge \texttt{every}(x, \texttt{man}(x) \longrightarrow \texttt{date}(x, y)))$

c. $\texttt{man}(e_1, x)$, $\texttt{woman}(e_2, y)$, $\texttt{date}(e_3, x, y)$, $\texttt{every}(e_4, x, e_9)$, '$\longrightarrow$'$(e_9, e_1, \boxed{e_3})$, $\texttt{some}(e_6, y, e_{10})$, '$\wedge$'$(e_{10}, e_2, \boxed{e_4})$, $\texttt{True}(\boxed{e_6})$

The only differences in the flat forms are marked with boxes in the expressions above. We can allow both readings by replacing the variables in the boxes with new variables:

(19) $\texttt{man}(e_1, x)$, $\texttt{woman}(e_2, y)$, $\texttt{date}(e_3, x, y)$, $\texttt{every}(e_4, x, e_9)$, '$\longrightarrow$'$(e_9, e_1, \boxed{e_5})$, $\texttt{some}(e_6, y, e_{10})$, '$\wedge$'$(e_{10}, e_2, \boxed{e_7})$, $\texttt{True}(\boxed{e_8})$

This way the scope of the quantifiers `every` and `some` are not specified, since any of them could take either a wide or a narrow scope. We can refine the ambiguity by adding subordination rules in a way similar to existing theories of expressing ambiguity by underspecification (Reyle 1993; Pinkal 1999; Copestake *et al.* 1997):

(20) $\texttt{man}(e_1, x)$, $\texttt{woman}(e_2, y)$, $\texttt{date}(e_3, x, y)$, $\texttt{every}(e_4, x, e_9)$, '$\longrightarrow$'$(e_9, e_1, e_5)$, $\texttt{some}(e_6, y, e_{10})$, '$\wedge$'$(e_{10}, e_2, e_7)$, $\boxed{e_3 \leq e_5}$, $\boxed{e_3 \leq e_7}$, $\texttt{True}(e_8)$, $\boxed{e_4 \leq e_8}$, $\boxed{e_6 \leq e_8}$

Now, both quantifiers `some` (with the reified entity e_6) and `every` (with the reified entity e_4) are dominated by e_8 (which is true), and indirectly they dominate the dating event e_3 via e_5 and e_7, respectively.

Should the annotator be able to disambiguate the sentence and wish to express one of the scope combinations only, say, narrow scope for `every` and wide scope for `some`, the annotator can simply add the new information:

(21) *every man dates some woman*
$\texttt{man}(e_1, x)$, $\texttt{woman}(e_2, y)$, $\texttt{date}(e_3, x, y)$, $\texttt{every}(e_4, x, e_9)$, '$\longrightarrow$'$(e_9, e_1, e_5)$, $\texttt{some}(e_6, y, e_{10})$, '$\wedge$'$(e_{10}, e_2, e_7)$, $e_3 \leq e_5$, $e_3 \leq e_7$, $\texttt{True}(e_8)$, $e_4 \leq e_8$, $e_6 \leq e_8$ $\boxed{e_5 = e_3}$, $\boxed{e_9 = e_7}$, $\boxed{e_6 = e_8}$

Of course, a human annotator or a program could just remove the subordination predicates and substitute entity names. But this is not necessary, since the information introduced does not create any contradictions.

Incrementality is also needed when we do not yet want (or know) how to represent certain meaning relationships. Thus we might not encode, say, tense and aspect, plurality, modality, and quantification. Any of these features may have to be taken into account later, but one must keep in mind that it is better to keep the semantic information as simple as possible, and extend them only when required. That way, possible ambiguities do not have to be spelled out but can rather be left underspecified. Also, if one is to implement a practical application with real text, simple semantic information will help to improve efficiency in the system and keep the response times down to manageable values. More importantly, further complexities in the semantic annotation could prevent a practical application from being accurate and robust enough, and they would require rather complex inference rules. But this is, of course, a decision to be taken when annotating the text.

4 Application: Answer Extraction

The most obvious application of this notation is an NLP system that can take advantage of partial logical forms of natural language sentences. Answer Extraction (AE) systems are examples of such applications. In this section we briefly explore the use of our notation in AE. Most of the work in this part is a natural extension of the work done in ExtrAns, an existing AE system (Mollá *et al.* 1998; Schwitter *et al.* 1999).

4.1 Answer extraction

Answer extraction systems try to find the smallest parts of contiguous text that individually provide an answer to an arbitrary question phrased in a natural language (such as English). Answer extraction is therefore a type of information retrieval (IR), but AE systems are not typical IR systems. Typical IR systems provide pointers to documents that are relevant to the query. These systems are useful for situations where the user needs to get all the information related to the query. AE systems, on the other hand, are ideal for situations where the user needs to find a specific answer to a particular question under severe time constraints. Typical applications of AE systems include interfaces to software manuals, help-desk systems in large organisations, and public enquiry systems available over the Internet. Given the current information overload, the need for this type of applications is becoming increasingly evident.

AE systems share many features with question answering (QA) systems. A full-fledged QA system, however, needs to tackle problems such as world knowledge, inferences, and language

generation. It is still a long way until the problems of encoding and using world knowledge and inferences are solved, and language generation is unfortunately an area where not much research is being done. AE is far less an ambitious task than QA, but a task that can be used for practical purposes and, more importantly, that can be used *now*.

4.2 (Web)ExtrAns

ExtrAns is an application that performs AE over UNIX manpages (Mollá *et al.* 1998; Schwitter *et al.* 1999). There is a web-based version that uses about 500 manpages for the AE task.[3] A new project, WebExtrAns, has started in November 1999 that will perform AE over the complete maintenance manual of the Airbus 320. The size of this manual (the original SGML manual is over 100Mb) will be conclusive in determining the scalability of such a system.

Both ExtrAns and WebExtrAns share the same architecture. In an off-line stage, the documents are processed by a sequence of modules that perform several types of linguistic analysis. Some of these modules are adaptations of publicly available software. For example, full-parsing is done by the Link Grammar (Sleator & Temperley 1993). All the word forms are normalised by converting them to the root forms, by using a lemmatiser that is provided together with the GATE tools (Gaizauskas *et al.* 1996). Disambiguation is done in several stages (Mollá & Hess 2000), including a corpus-based PP disambiguator (Brill & Resnik 1994). Finally, anaphora resolution is done by adapting an algorithm devised for the Slot Grammar (Lappin & Leass 1994). Other modules are implemented from scratch. One of them is the preprocessor and tokeniser that splits the text into sentences and sends them to the parser. Another module is the logical form generator, which constructs the logical forms that are stored in databases for the AE task (Mollá *et al.* 2000).

In an on-line stage, the user questions are processed with the same linguistic modules and the logical forms of the questions are produced. The answers are found by applying a proof algorithm of the logical forms of the questions over those of the document sentences. The retrieved sentences are displayed to the user, with the words that directly answer the question highlighted according to their contribution to the answer.

A central point of (Web)ExtrAns is the format of the logical forms used, and how they can be constructed and used for the AE task. Whereas the technical details of the actual logical forms used in ExtrAns have been explained elsewhere (Mollá *et al.* 1998; Mollá *et al.* 2000; Schwitter *et al.* 1999), the following sections explain how the notation introduced in this paper (which is more general than the notation actually used in the (Web)ExtrAns projects) can be used for AE.

4.3 Logical forms for answer extraction

The most important feature of our notation for AE is the possibility of expressing partial logical forms. Even if the parser provides a thorough syntactic analysis of the sentences (as the parser used for ExtrAns does), the resulting syntactic structures may turn out to be too difficult to analyse by the logical form generator. And even if they can be analysed, there is no need for such a fine detail in the logical form. This is so because the AE system will return acceptable results even if the sentences retrieved do not give an *exact* answer to the question. In fact, sometimes a near-miss may be informative. Below is a question and some possible answers:

(22) *which command copies files?*

(23) a. *cp copies files*

b. *cp copies the contents of filename1 onto filename2*

c. *if the option -r is specified, cp recursively copies directory1 and all the files and directories within it*

d. *cp refuses to copy a file onto itself*

The best answer to (22) is undeniably (23a). But (23b) is also an acceptable answer, although it is saying that cp copies the *contents* of a specific file. The next sentence, (23c), also contributes to the answer to the question. One

[3] `http://www.ifi.unizh.ch/cl/extrans/`

could even argue that (23d) is also an informative sentence, even when it explicitly says that cp does not copy certain types of files under certain conditions.

An important piece of information that is retained in all the answers above is the verb-argument structure of the sentences retrieved. All of the answers are about commands copying (or not copying) files or parts of them. The following logical forms are partial logical forms of the answers:

(23') a. $\texttt{True}(e_2)$, $\texttt{cp}(e_1, x)$, $\texttt{copy}(e_2, x, y)$, $\texttt{file}(e_3, y)$

b. $\texttt{True}(e_2)$, $\texttt{cp}(e_1, x)$, $\texttt{copy}(e_2, x, z)$, $\texttt{content}(e_4, z, y)$, $\texttt{file}(e_3, y)$, $\texttt{onto}(e_5, e_2, w)$, $\texttt{file}(e_6, w)$

c. $\texttt{True}(e_4)$, $\texttt{if}(e_4, e_5, e_6)$, $\cdots$, $\texttt{cp}(e_1, x)$, $\texttt{copy}(e_2, x, y)$, $\texttt{file}(e_3, y)$, $\cdots$

d. $\texttt{True}(e_4)$, $\texttt{cp}(e_1, x)$, $\texttt{refuse}(e_4, x, e_6)$, $\texttt{copy}(e_2, x, y)$, $\texttt{file}(e_3, y)$, $\texttt{onto}(e_5, e_2, y)$, $\texttt{and}(e_6, e_2, e_3, e_5)$

These logical forms ignore much of the information given by the sentence. For example, tense and aspect, quantification (*all*, etc), modality, plurals, are all ignored. Still, the resulting partial logical forms are not trivial to produce, since one needs to solve anaphoric references and find the arguments of verbs in embedded clauses. Producing these partial logical forms is feasible with the current technology, as ExtrAns has demonstrated (Mollá *et al.* 2000).

And more importantly, these logical forms can be used to decide if the sentence is an answer to the question. This is the topic of next section.

4.4 Finding an answer to the question

Finding an answer to a question in an AE system becomes relatively easy if one uses the flat logical forms described in this paper. We can use the question to produce the partial information that the answer must have. For example, (22) can produce:

(22') $\texttt{command}(e_1, x)$, $\texttt{copy}(e_2, x, y)$, $\texttt{file}(e_3, y)$

Now, provided that we add an axiom that states that *cp* is a command, the predicates in (22') are part of the logical forms of all the answers (23). Note that we do not add $\texttt{True}(e_2)$ to (22') so that we can find embedded clauses, such as in (23c) and (23d).[4]

An answer to a specific question is such that it follows the following definition:

Definition 1 *Let $PLF_{\mathcal{Q}}$ be the partial logical form of a question $\mathcal{Q}$, and let $PLF_{\mathcal{A}}$ be the partial logical form of a sentence $\mathcal{A}$. Then, $\mathcal{A}$ answers $\mathcal{Q}$ iff:*

$$PLF_{\mathcal{A}} \longrightarrow PLF_{\mathcal{Q}}$$

In other words, a sentence $\mathcal{A}$ answers a question $\mathcal{Q}$ if the logical form of $\mathcal{A}$ is more restricted than the logical form of $\mathcal{Q}$. The partial logical form of the question is an underspecified version of the partial logical form of the answer. If we use the notation introduced in this paper, the test cannot be easier:

Definition 2 *Let $PLF_{\mathcal{Q}}$ be the partial logical form of a question $\mathcal{Q}$, and let $PLF_{\mathcal{A}}$ be the partial logical form of a sentence $\mathcal{A}$. Then, $\mathcal{A}$ answers $\mathcal{Q}$ if $PLF_{\mathcal{A}}$ contains all the predicates of $PLF_{\mathcal{Q}}$.*

Using Definition 2, different types of questions find different types of sentences. In particular, this mechanism returns acceptable answers for the following types of questions:

Wh-. The question *which command copies files?* finds sentences that explicitly say that a specific command copies files, such as:

(24) a. *cp copies the contents of filename1 onto filename2*

b. *rcp copies files between machines*

c. *sed copies the filenames to the standard output, edited according to a script of commands*

[4] An inference rule is also needed to retrieve (23b), namely, that the contents of an object is identical to the object itself. In (23b) this means that $z = y$. This and other inference rules (such as that all UNIX commands including *cp* are commands) have been added to ExtrAns.

d. *cp refuses to copy a file onto itself*

Yes/no. The question *does cp copy files?* finds sentences that state that cp copies files:

(25) a. *cp copies the contents of filename1 onto filename2*

b. *cp refuses to copy a file onto itself*

c. *If directory2 does not exist, cp creates it and duplicates*[5] *the files and subdirectories of directory1 within it*

Definition 2 can even work for some types of *how...?* questions. For example, the question *how do I remove a file?* finds all the sentences that specify that files are removed (if we assume that "I" is treated roughly as "anybody/anything"):

(26) a. *rm removes one or more files*

b. *If a single file is compiled and loaded all at once, the intermediate file is deleted*[5]

c. *If pack is successful, filename will be removed*

Other types of questions, such as *why ...?*, or more complex questions such as *what is the best ...?* or *how many ...?* would require less trivial procedures to find the answers because these questions do not directly determine adequate underspecified versions of partial logical forms of the answers.

If there are no sentences in the documents that satisfy the implication, it is still possible to find the best answers by selecting those sentences whose partial logical forms contain the highest overlap with the partial logical form of the question. The process would be similar to how one can determine the similarity between two sentences, which is explained in next section.

[5]To retrieve (25c) and (26b) we would also need to keep a thesaurus that lists synonymy relations, such as between *copy* and *duplicate*. ExtrAns, for example, keeps a simple thesaurus with some of the most frequent synonyms in the world of manpages.

4.5 Sentence similarity

It is fairly easy to use our flat logical forms to compare the semantic contents of two sentences:

Definition 3 *A sentence $\mathcal{A}$ with partial logical form $PLF_{\mathcal{A}}$ is semantically equivalent to another sentence $\mathcal{B}$ with partial logical form $PLF_{\mathcal{B}}$ if $PLF_{\mathcal{A}}$ and $PLF_{\mathcal{B}}$ unify.*

For example, the two following sentences are semantically equivalent:

(27) a. *cp copies the files*
`True`(e_2), `cp`(e_1, x), `copy`(e_2, x, y), `file`(e_3, y)

b. *the files are copied by cp*
`True`(e_2), `file`(e_1, y), `copy`(e_2, x, y), `cp`(e_3, x)

The logical forms unify because e_1 in (27a) maps e_3 in (27b), and e_3 in (27a) maps e_1 in (27b).

The true usefulness of our notation is the comparison of sentences that do not have the same logical forms. Two sentences that are very similar in meaning will have a high overlap of predicates. It is therefore easy to compute the semantic closeness of two sentences by simply computing the best way to unify their logical forms. Given two sentences, $\mathcal{A}$ and $\mathcal{B}$, four indices can be computed:

$\mathcal{O}_{\mathcal{A}}$: The maximum number of predicates in $\mathcal{A}$ that can unify at once with predicates in $\mathcal{B}$.

$\mathcal{O}_{\mathcal{B}}$: The maximum number of predicates in $\mathcal{B}$ that can unify at once with predicates in $\mathcal{A}$.

$\mathcal{D}_{\mathcal{A}}$: The number of predicates that remain in $\mathcal{A}$.

$\mathcal{D}_{\mathcal{B}}$: The number of predicates that remain in $\mathcal{B}$.

For example, let us consider the following sentences in the context of a sports report:

(28) a. $\mathcal{A}$: *Madrid defeated Barcelona in the last match*
`Madrid`(e_1, x), $\boxed{\texttt{defeat}(e_2, x, y)}$, `Barcelona`$(e_3, y)$, `in`$(e_4, e_2, e_5)$, `last`$(e_5, z)$, `match`$(e_6, z)$, `True`$(e_2)$

b. $\mathcal{B}$: *Madrid was defeated by Barcelona in the last match*
$\texttt{Madrid}(e_1, x)$, $\boxed{\texttt{defeat}(e_2, y, x)}$,
$\texttt{Barcelona}(e_3, y)$, $\texttt{in}(e_4, e_2, e_5)$,
$\texttt{last}(e_5, z)$, $\texttt{match}(e_6, z)$, $\texttt{True}(e_2)$

The two sentences generate exactly the same predicates (7 predicates), but the arguments in `defeat` (boxed) are different. For that reason, the logical forms do not unify. The values of our four indices are:

(29) $\mathcal{O}_\mathcal{A} = 6, \mathcal{O}_\mathcal{B} = 6, \mathcal{D}_\mathcal{A} = 1, \mathcal{D}_\mathcal{B} = 1$

These four indices can be combined to form a unique index. For example, we can use a straightforward formula such as:

(30) $\mathcal{I} = \frac{\frac{\mathcal{O}_\mathcal{A}}{\mathcal{O}_\mathcal{A}+\mathcal{D}_\mathcal{A}} + \frac{\mathcal{O}_\mathcal{B}}{\mathcal{O}_\mathcal{B}+\mathcal{D}_\mathcal{B}}}{2} \times 100$

Formula (30) would give a value $\mathcal{I} = 86\%$ for the indices (29).

$\mathcal{O}_\mathcal{A}$ may be different from $\mathcal{O}_\mathcal{B}$ if one of the sentences contains redundant information. For example:

(31) a. $\mathcal{A}$: *the cp command copies files*
$\texttt{cp}(e_1, x)$, $\texttt{command}(e_2, x)$, $\texttt{copy}(e_3, x, y)$,
$\texttt{file}(e_4, y)$, $\texttt{True}(e_3)$

b. $\mathcal{B}$: *the cp command is a command that copies files*
$\texttt{cp}(e_1, x)$, $\texttt{command}(e_2, x)$,
$\texttt{command}(e_5, z)$, $\texttt{be}(e_6, x, z)$,
$\texttt{copy}(e_3, z, y)$, $\texttt{file}(e_4, y)$, $\texttt{True}(e_6)$

If we assume that $\texttt{be}(e, x, z)$ implies $x = z$, then both $\texttt{command}(e_2, x)$ and $\texttt{command}(e_5, z)$ in (31b) unify with $\texttt{command}(e_2, x)$ in (31a), giving the following indices:

(32) a. $\mathcal{O}_\mathcal{A} = 4, \mathcal{O}_\mathcal{B} = 5, \mathcal{D}_\mathcal{A} = 1, \mathcal{D}_\mathcal{B} = 2$

b. $\mathcal{I} = 76\%$

It is of course possible to assign different weights to different types of predicates. For example, the verb of a main sentence could produce a predicate with higher weight than the predicates of sentence complements. This way we may obtain more intuitive values for the indices.

5 Conclusions

In this paper we have introduced a notation for semantic information that uses reification extensively to produce flat logical forms. The resulting notation is incrementally extensible, and therefore it can be used to provide partial semantic information of the text in such a way that it can be extended later without necessarily having to destroy old information. The fact that this notation is based on the notation of the logical forms used by a running system that performs answer extraction over unedited text shows that it is possible to design tools that can assist the human annotator or, indeed, that the annotation procedure can include an automatic stage where partial information is produced. The annotator can later refine the annotation manually with further information, or leave as it is. This hybrid approach would make it possible to quickly produce partial and accurate semantic annotations of large volumes of text.

References

Brill, Eric & Philip Resnik, 1994. A rule-based approach to prepositional phrase attachment disambiguation. In *Proc. COLING '94*, volume 2, pp.998–1004, Kyoto, Japan.

Copestake, Ann, Dan Flickinger & Ivan A. Sag, 1997. Minimal recursion semantics: an introduction. Technical report, CSLI, Stanford University, Stanford, CA.

Davidson, Donald, 1967. The logical form of action sentences. In Nicholas Rescher (ed.) *The Logic of Decision and Action*, pp.81–120. Univ. of Pittsburgh Press.

Gaizauskas, Robert, Hamish Cunningham, Yorick Wilks, Peter Rodgers & Kevin Humphreys, 1996. GATE: an environment to support research and development in natural language engineering. In *Proceedings of the 8th IEEE International Conference on Tools with Artificial Intelligence*, Toulouse, France.

Hobbs, Jerry R., 1985a. The logical notation: Ontological promiscuity. Manuscript.

——, 1985b. Ontological promiscuity. In *Proc. ACL'85*, pp.61–69. University of Chicago, Association for Computational Linguistics.

Lappin, Shalom & Herbert J. Leass, 1994. An algorithm for pronominal anaphora resolution. *Computational Linguistics* 20(4): 535–561.

Mollá, Diego, Jawad Berri & Michael Hess, 1998. A real world implementation of answer extraction. In *Proc. of the 9th International Conference and Workshop on Database and Expert Systems. Workshop "Natural Language and Information Systems" (NLIS'98)*, pp.143–148, Vienna.

—— & Michael Hess, 2000. Dealing with ambiguities in an answer extraction system. In *Workshop on Representation and Treatment of Syntactic Ambiguity in Natural Language Processing*, pp.21–24, Paris. ATALA.

——, Gerold Schneider, Rolf Schwitter & Michael Hess, 2000. Answer extraction using a dependency grammar in ExtrAns. *T.A.L.* 41(1): 127–156.

Parsons, Terence, 1985. Underlying events in the logical analysis of English. In Ernest Lepore & Brian P. McLaughlin (eds.) *Actions and Events: Perspectives on the philosophy of Donald Davidson*, pp.235–267. Oxford: Blackwell.

Pinkal, Manfred, 1999. On semantic underspecification. In *Proc. 3rd International Workshop on Computational Semantics*, Tilburg. ITK, Tilburg University.

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

Schwitter, Rolf, Diego Mollá & Michael Hess, 1999. Extrans — answer extraction from technical documents by minimal logical forms and selective highlighting. In *Proc. Third International Tbilisi Symposium on Language, Logic and Computation*, Batumi, Georgia. `http://www.ifi.unizh.ch/cl/`.

Sleator, Daniel D. & Davy Temperley, 1993. Parsing English with a link grammar. In *Proc. Third International Workshop on Parsing Technologies*, pp.277–292.