Ontologically Promiscuous Flat Logical Forms for NLP

Diego Mollá

Macquarie University
Sydney
Diego.Molla-Aliod@mq.edu.au

Abstract

In this paper a flat notation for logical forms is described. This notation allows the logical forms to be easy to build, easy to work with, and able to deal with ambiguity by underspecification. The main mechanism to convert a logical form into the corresponding flat form is the reification of all the predicates and operators used in an otherwise nested expression. The resulting flat logical forms are convenient for natural language processing applications that require the use of partial logical forms. In particular, it is shown how partial logical forms (encoded in flat notation) can be used to perform answer extraction.

Keywords: flat logical form, reification, underspecification, partial logical form, answer extraction

In this paper we explore a notation for logical forms that can be useful for several Natural Language Processing (NLP) applications. The main goal is to provide a notation that is easy to build, easy to work with, and able to deal with ambiguity by underspecification. In Section 1 we introduce the notation. In Section 2 we mention some of the most important features of this notation and how the resulting logical forms can be used for NLP. Finally, in Section 3 we explain the use of this notation for a particular NLP application, Answer Extraction.

1 The notation

1.1 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 `exp` are changed 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).

1.2 Ontological promiscuity

A method to convert a (nested) logical form into a flat form is to reify all the partial expressions that appear in the logical form, and use the reified entities to refer to these partial expressions. In our analogy with computing programs, these reified entities are the "results" of the partial expressions.

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 1985) carried to its extremes. The flat form of (3b) becomes:

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

This is a consequence of Hobbs' ontological promiscuity: all the morphemes in the sentence are subject to reification. 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).

1.3 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 2000) 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 the relation between a nested form and its flat form, we do not make use of 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`. The use of the name `True` also allows a better parallelism with `False`, as we will see later.

What we propose is this: Reify all the predicates and logical operators in the logical expression, and add $\texttt{True}(e)$ 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*
$\texttt{bachelor}(e_1, j)$, $\texttt{married}(e_2, j)$, $\neg(e_3, e_2)$, $\longrightarrow (e_4, e_1, e_3)$, $\texttt{True}(e_4)$

2 Working with the notation

The notation introduced in this paper has several properties that make it very interesting for some NLP applications. In this part we will briefly

mention some of these properties.

2.1 Partial logical forms

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

(12) *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. An NLP system may have difficulty to provide a correct logical form for the sentence. Let us assume that there is an agreement about the logical form of (12), say:

$$(13)\quad \texttt{all}(x,\ \texttt{apple}(x) \wedge \texttt{in}(x,b) \longrightarrow \texttt{may}(\texttt{perfect}(\texttt{want}(j,\texttt{eat}(j,x)))))$$

It is fairly easy to find the corresponding flat form:

$$(14)\quad \texttt{True}(e_1),\ \texttt{all}(e_1,x,e_2),\ \longrightarrow(e_2,e_3,e_4),\ \wedge(e_3,e_5,e_6),\ \texttt{apple}(e_5,x),\ \texttt{in}(e_6,x,b),\ \texttt{may}(e_4,e_7),\ \texttt{perfect}(e_7,e_8),\ \texttt{want}(e_8,j,e_9),\ \texttt{eat}(e_9,j,x)$$

Given (14), it is trivial to produce a partial representation of it, one need just remove some of the predicates. More importantly, the parser of a NLP system may fail to provide a full parse of (12) because it ignores, say, quantification, modality, complex tenses, and PP modifiers. But the output given by the parser can still be used to form a partial logical form. For example, the following could be generated from the information of this hypothetical parser:

$$(15)\quad \texttt{eat}(e_9,j,x),\ \texttt{apple}(e_5,x),\ \texttt{in}(e_6,a_1,b)$$

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 (15) provides partial information with respect to the complete logical form (14). We would say that (15) is a partial logical form.

2.2 Inferences

The predicate `True` becomes very useful for systems that need to make inferences. Every partial expression that can be inferred to be true can be marked by adding `True` over its reified entity. Parallel to this, every partial expression that can be inferred to be false can also be marked by a new predicate, say, `False`. This information can be used in subsequent inference steps.

The following equivalence rules can be used for inferences over some of the logical operators:

$$
\begin{aligned}
\exists e_1(\wedge(e_1, e_2, e_3) \wedge \texttt{True}(e_1)) &\equiv \texttt{True}(e_2) \wedge \texttt{True}(e_3) \\
\exists e_1(\wedge(e_1, e_2, e_3) \wedge \texttt{False}(e_1)) &\equiv \texttt{False}(e_2) \vee \texttt{False}(e_3) \\
\exists e_1(\vee(e_1, e_2, e_3) \wedge \texttt{True}(e_1)) &\equiv \texttt{True}(e_2) \vee \texttt{True}(e_3) \\
\exists e_1(\vee(e_1, e_2, e_3) \wedge \texttt{False}(e_1)) &\equiv \texttt{False}(e_2) \wedge \texttt{False}(e_3) \\
\exists e_1(\longrightarrow(e_1, e_2, e_3) \wedge \texttt{True}(e_1)) &\equiv \texttt{True}(e_2) \longrightarrow \texttt{True}(e_3) \\
\exists e_1(\longrightarrow(e_1, e_2, e_3) \wedge \texttt{False}(e_1)) &\equiv \texttt{True}(e_2) \wedge \texttt{False}(e_3) \\
\exists e_1(\neg(e_1, e_2) \wedge \texttt{True}(e_1)) &\equiv \texttt{False}(e_2) \\
\exists e_1(\neg(e_1, e_2) \wedge \texttt{False}(e_1)) &\equiv \texttt{True}(e_2)
\end{aligned}
$$

These axioms do not encode inferences about quantification,[2] but they suffice for simple inferences.

Note that we are not trying to prove the logical equivalence between a flat form such as (14) and the nested form (13). We are instead trying to construct a notation that keeps as many of the inferences as possible. Therefore, we do not need axioms like Hobbs' Axiom 1.

2.3 Ambiguity by underspecification

The reified flat forms introduced in this paper have some resemblances with approaches that treat ambiguity by means of underspecification (Reyle 1993; Pinkal 1999; Copestake *et al.* 1997). Like in these approaches, partial expressions have identifiers that can be used for reference. We could create dominance graphs by adding dominance predicates over reified entities in the same way as these approaches use dominance predicates over meta-variables.

[2] This is still a topic for further research. One could try to use Hobbs' quantification mechanism (Hobbs 1983; Hobbs 1996), for example.

As a way of illustration, the following expression could be used to account for scopal underspecification:

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

Here, both quantifiers $\texttt{some}\ e_6$ and $\texttt{every}\ e_4$ are dominated by e_8 (which is true), and indirectly they dominate the dating event e_3 via e_5 and e_7, respectively.

3 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 introduce AE and how it can take advantage of our notation. 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).

3.1 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*.

3.2 (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 size of 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 had to be 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.* forthcoming).

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 ex-

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

plained elsewhere (Mollá *et al.* 1998; Mollá *et al.* forthcoming; 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.

3.3 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:

(17) *which command copies files?*

(18) 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 (17) is undeniably (18a). But (18b) is also an acceptable answer, although it is saying that cp copies the *contents* of a specific file. The next sentence, (18c), also contributes to the answer to the question. One could even argue that (18d) 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:

(18') 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.* forthcoming).

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 the next section.

3.4 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, (17) produces the following partial logical form:

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

Now, provided that we add an axiom that states that *cp* is a command, the predicates in (17') are part of the logical forms of all the answers (18). Note that we do not add $\texttt{True}(e_2)$ to (17') so that we can find embedded clauses, such as in (18c) and (18d).[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:*

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

[4] An inference rule is also needed to retrieve (18b), namely, that the contents of an object is identical to the object itself. In (18b) 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.

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:

(19) 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*

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:

(20) 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"):

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

[5] To retrieve (20c) and (21b) 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.

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 the next section.

3.5 Evaluating the system: a proposal

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:

(22) 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 (22a) maps e_3 in (22b), and e_3 in (22a) maps e_1 in (22b).

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:

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

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:

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

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

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

Formula (25) would give a value $\mathcal{I} = 86\%$ for the indices (24).

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

(26) 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 (26b) unify with $\texttt{command}(e_2, x)$ in (26a), giving the following indices:

(27) 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.

Sentence comparison can be used for an evaluation of the performance of QA systems in general. For example, it would be possible to compile a set of questions for the evaluation. The evaluation team would write the best answers to each question as complete sentences, very much like a learner of a new language is supposed to answer questions: *what is your name? – my name is Peter*, etc. These answers would be stored as "gold standard" answers. The results given by the system can now be compared with the "gold standard" by using the indices explained in this section.

The "gold standard" answers can also be used to evaluate the retrieval performance of AE systems (such as ExtrAns). The most difficult problem in evaluating AE systems is to find the set of all the answers available in the documents. Apart from the obvious problem of searching the documents to find the candidates, sometimes it is difficult to decide if a particular sentence actually answers a particular question, and the decision may depend on the person that does the judgement, or even on the person's mood at that particular moment. Instead of using a human, the "gold standard" answers can be used to help finding the best answers in the documents. Those sentences in the documents that are close enough to the "gold standard" are selected. The selection can now be revised manually by an expert (which, as we have just said, may be problematic, but at least the expert has to look at less data), or they may be taken as they are. We believe that the evaluation with these selected sentences would be fairly acceptable, at a fraction of the cost of a full evaluation by a human.

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., 1983. An improper treatment of quantification in ordinary English. In *Proc. ACL'83*, pp.57–63, Cambridge, MA.

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

——, 1996. Monotone decreasing quantifiers in a scope-free logical form. In Kees van Deemter & Stanley Peters (eds.) *Semantic Ambiguity and Underspecification*, chapter 3, pp.55–76. Stanford, CA: CSLI Publications.

——, 2000. The logical notation: Ontological promiscuity.

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, forthcoming. 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. 5th 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.