The Extended Lexicon: Language Processing as Lexical Description

Roger Evans
Natural Language Technology Group
Computing Engineering and Mathematics
University of Brighton, UK
R.P.Evans@brighton.ac.uk

Abstract

In this paper we introduce an approach to lexical description which is sufficiently powerful to support language processing tasks such as part-of-speech tagging or sentence recognition, traditionally considered the province of external algorithmic components. We show how this approach can be implemented in the lexical description language, DATR, and provide examples of modelling extended lexical phenomena. We argue that applying a modelling approach originally designed for lexicons to a wider range of language phenomena brings a new perspective to the relationship between theory-based and empirically-based approaches to language processing.

1 The Extended Lexicon

A lexicon is essentially a structured description of a set of lexical entries. One of the first tasks when developing a lexicon is to decide what the lexical entries are. This task has two dimensions: what kind of linguistic object does a lexical entry describe, and what does it say about it. So for example, one might decide to produce a lexicon which describes individual word instances, and provides the orthographic form and part-of-speech tag for each form. It is the first of these dimensions that is most relevant to the idea of the Extended Lexicon. Conventionally, there are two main candidates for the type of linguistic object described by a lexicon: word forms (such as ***sings***, ***singing***, ***sang***[1]), corresponding to actual words in a text and lexemes (such as **SING**, **WALK**, **MAN**), describing abstract words, from which word forms are somehow derived. Choosing between these two candidates might be a matter of theoretical disposition, or a practical consideration of how the lexicon is populated or used.

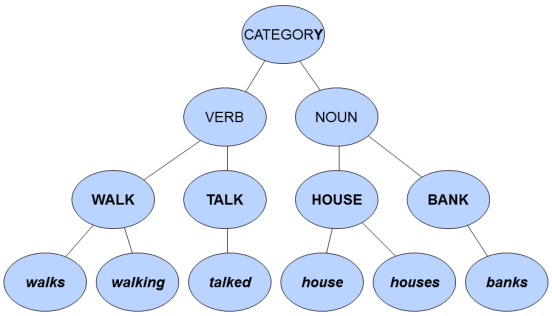

Figure 1: A simple inheritance-based lexicon

In the Extended Lexicon, we introduce a third kind of linguistic object, called word instances (or just instances), consisting of word forms as they occur in strings (sequences of words, typically sentences). For example, a string such as *the cats sat on the mat* contains two distinct instances of the word *the*. *the cats slept* contains further (distinct) instances of *the* and *cats*. However the instances in a repetition of *the cats sat on the mat* are the same as those in the original (because instances are defined relative to strings, that is, string types not string tokens).

So in an extended lexicon, the lexical entries are word instances, and the lexicon itself is a structured description of a set of word instances. In order to explore this notion in more detail, it is helpful to introduce a more specific notion of a 'structured description'. We shall use an inheritance-based lexicon, in which there are internal abstract 'nodes' representing information that is shared by several lexical entries and inherited by them. Figure 1 shows the structure of a simple inheritance-based lexicon with some abstract high-level structure (CATEGORY, VERB, NOUN), then a layer of lexemes (**WALK**, **TALK**, **HOUSE**, **BANK**), and below that a layer of word forms (***walks***, ***walking***,

[1]Typographical conventions for object types: ABSTRACT, **LEXEME**, ***wordform***, *instance*, `code`.

Proceedings of Recent Advances in Natural Language Processing, pages 270–276,
Hissar, Bulgaria, 7-13 September 2013.

talked, ***house***, ***houses***, ***banks***, as well as many more). Thus the word form ***walks*** inherits information from the lexeme **WALK**, which inherits from abstract node VERB and then abstract node CATEGORY.

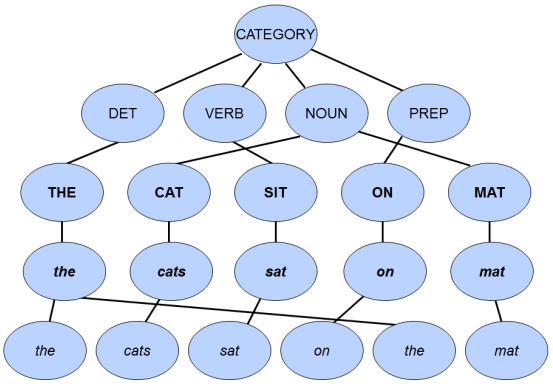

Figure 2: A lexicon with instance nodes

Adding instances to this model is in principle very easy: one just creates a further layer of nodes below the word forms. The word instances are now the lexical entries, and the word form nodes are abstractions, representing information shared by all instances of the form. Figure 2 shows a first pass at adding an instance layer to a lexicon supporting the string *the cats sat on the mat*, by adding new nodes for each instance in the string. However, what is missing from this figure is any representation of the string as a whole – nothing distinguishes the two instance nodes *the* from each other, or indeed from their parent word form node ***the***, and nothing identifies them as members of a specific string. One way this information could be added is simply by stipulating it: each instance node could have a feature whose value is the string, and another whose value is the index in the string of the current instance. However, in the Extended Lexicon, we adopt a structural solution, by linking the instance nodes of a string together into a chain, using inheritance links `prev` ('previous') and `next` to inherit information from this instance's neighbours in the string. Diagrammatically, we represent this as in figure 3.

To summarise, in the Extended Lexicon model, a lexicon is an inheritance-based structured description of a set of word instances. This notion simultaneously captures and combines two important modelling properties: first, that instances of the same word share properties via an abstract word form node, and second that the lexicon implicitly encodes word strings, as maximal chains of linked instances.

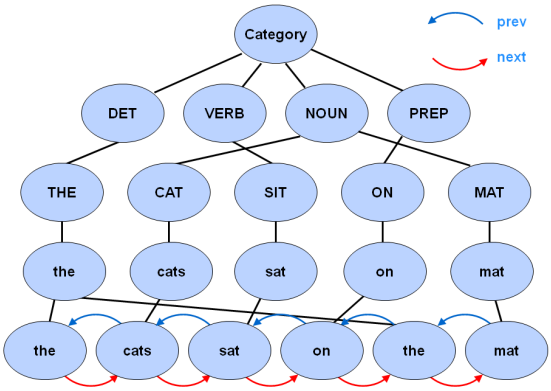

Figure 3: A simple Extended Lexcion, with instance nodes linked into a chain

2 The Extended Lexicon in DATR

2.1 DATR in brief

DATR (Evans and Gazdar, 1996) is a lexical description language originally designed to model the structure of lexicons using default inheritance. The core descriptive unit in DATR is called a node, which has a unique node name (capitalised) and has associated with it a set of definitional path equations mapping paths (sequences of features) onto value definitions.

```
DOG:
  <cat> == noun
  <form> == dog.
```

Figure 4: DATR description – version 1

Figure 4 is a simple example of DATR code. This fragment defines a node called `DOG` with two path equations, specifying that the (syntactic) category is `noun`, and the (morphological) form is `dog`.

```
NOUN:
  <cat> == noun
  <form> == "<root>".
DOG:
  <> == NOUN:<>
  <root> ==  dog.
```

Figure 5: DATR description – version 2

Figure 5 provides a slightly more complex definition. In this version, there is an abstract node, `NOUN`, capturing information shared between all nouns and a new definition for `<form>` which is defined to be the same as the path `<root>`. `DOG` now specifies a value for `<root>`, and inherits everything else from `NOUN`.

Inheritance in DATR operates as follows: to determine the value associated with a path at a particular node, use the definition from the equation for the longest path that matches a leading (leftmost) subpath of the desired path (if none matches, the value is undefined). The definition might give you a value, or a redirection to a different node and/or path, or a combination of these. If the definition contains path values, extend those paths with the portion of the desired path that did not match the left-hand-side and seek the value of the resulting expression.

So in this example, the path <root> at DOG matches a definition equation path exactly, and so has value dog. The path <cat> is not defined at DOG and the longest defined subpath is <>, so this definition is used. It specifies a value NOUN:<>, but the path is extended with the unmatched part of the original path, so the definition becomes NOUN:<cat>. This has the value noun, so this is the value for DOG:<cat> as well. Finally, the path <form> at DOG similarly matches the <> path and is rewritten to NOUN:<form>. This matches the definition in NOUN which specifies "<root>". The quotes here specify this is evaluated as DOG:<root> (without the quotes it would be interpreted locally as NOUN:<root>), and because the entire path matched, there is nothing further to add to the path here, so the value is DOG:<root>, that is, dog.

```
NOUN:
  <cat> == noun
  <num> == sing
  <form> == "<table "<num>" >"
  <table> == "<root>"
  <table plur> == "<root>" s.
DOG:
  <> == NOUN:<>
  <root> ==  dog.
Dog:
  <> == DOG:<>.
Dogs:
  <> == DOG:<>
  <num> == plur.
```

Figure 6: DATR description – version 3

Finally, the version in figure 6 extends the definition of NOUN in several ways: the path <num> defines morphological number (sing or plur); the path <form> now defines the morphological form in terms of a table of forms indexed by the number feature[2]; finally the definition for

[2]Note the use of embedded path expressions here: the inner expression is evaluated first and the result spliced into the outer expression

```
Word1:
  <> == The:<>
  <next> == "Word2:<>".
 Word2:
  <> == Dogs:<>
  <prev> == "Word1:<>"
  <next> == "Word3:<>".
 Word3:
  <> == Slept:<>
  <prev> == "Word2:<>".
```

Figure 7: Instance node for *the dogs slept*

<table> has a default value which is just the root, and a plural value which appends an s to the morphological root. Two word form nodes have also been added, Dog whose form will be dog, and Dogs whose form will be dog s.

2.2 Modelling the Extended Lexicon

Figure 6 provides an example of DATR code to represent lexeme and word form nodes. Extending this to represent instance nodes as well is quite straightforward. The instance nodes themselves inherit directly from the corresponding word form nodes. The prev and next links map between the instance nodes, as shown in figure 7, for the word string *the dogs slept*.

As a first simple example of the Extended Lexicon approach, figure 8 provides a definition for the lexeme A which varies the actual form according to whether the next word starts with a vowel or not. This definition presupposes a feature <vstart> which returns true for words that start with a vowel, false otherwise[3]. A evaluates vstart not on itself, but on the word instance that follows it (signified by <next vstart>) to determine whether its own form is a or an.

```
A:
  <> == DET
  <form> == <table "<next vstart>">
  <table> == a
  <table true> == an.
```

Figure 8: Word form definition for ***A***

This example illustrates some important features of the approach. First, lexeme (or word form) nodes can make assertions about instance nodes which do not hold for the abstract nodes themselves – **A** contains no definition for <next>, so evaluation of <form> is undefined, but an instance node inheriting from it will define <next>

[3]We do not define <vstart> here – its default definition would be at the topmost abstract node, but some lexemes could override it, for example **HISTORIC** in some dialects would set it true (as in *an historic event*).

and hence <form>. Second, these assertions do not need to make direct reference to other lexical definitions – they are entirely local to **A**. Finally, these assertions do not alter the definition of the instance nodes – the only properties unique to an instance node are its parent node and its previous and next instance nodes.

This last point has considerable practical importance. In the Extended Lexicon the number of lexical entries (instances) is unbounded, since the number of possible word strings is unbounded. This is not significant problem as long as it is possible to provide an effective procedure for specifying instance definitions for any desired string dynamically. But this is straightforward: for each string create instance nodes such as shown in figure 7 with unique (but arbitrary) names for each node. The definition of each of these nodes requires only the names of the previous and next instance nodes and the name of the parent word form node. The former are known to the specification algorithm, and various conventions are possible for locating the word form node; in figure 7 we assume the word form itself, capitalised, is the name of the node.

3 Examples

3.1 Part of speech tagging

A more challenging task is part-of-speech (POS) tagging. Conventionally, POS taggers are configured as applications which are applied to texts and use either rule-based algorithms (eg (Brill, 1992)) or statistical algorithms (eg (Garside, 1987)) to provide POS tags for each word in the text. In the Extended Lexicon approach, POS tagging is conceived as a sequence of inferences required to determine the value of the feature path <pos> for a given instance node. As before, the definition is provided entirely by abstract nodes. Figure 9 presents a simple abstract lexicon with five word form nodes and one common root node. Each node is annotated with DATR code to support simple POS tagging.

The definition of the <pos> path is provided at the root node, WORDFORM, and inherited by all other nodes. It defines <pos> to be the value "<table "<prev pos>" "<prev prev pos>" >". In other words, to determine pos use the lookup table table, indexed with the pos of the previous two words. The lookup tables are defined on a per-word form basis, but inherit

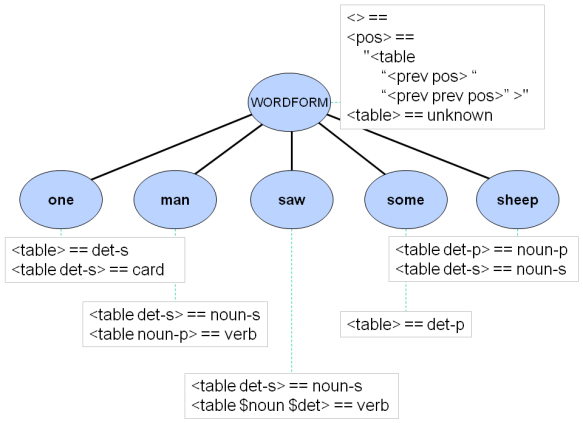

Figure 9: POS tagging

the default definition (the value unknown) from the root when not specified. Finally the root node also specifies a catch-all empty value (<> ==) for unspecified paths (including <prev> paths from Word1).

With just these definitions, the lexicon defines <pos> values for the word form nodes without access to any context (so the <prev paths return nothing). The <pos> for ***one*** is det-s (singular determiner), for ***man***, ***saw***, and ***sheep***, it is unknown (inherited from WORDFORM), and for ***some***, it is det-p. The tables vary this behaviour for instances according to previous context: ***one*** becomes a card if preceded by a det-s, ***man*** is a noun-s if preceded by a det-s, and a verb if preceded by a noun-p. ***saw*** makes use of the full context: if preceded by det-s it is a noun, but it is a verb if preceded by a noun and before that a determiner[4]. ***some*** is always a plural determiner, and ***sheep*** takes its number from the preceding determiner.

Figures 10 and 11 show what happens when we add word instances, linked together into strings. The two strings, *one man saw some sheep* and *some sheep man one saw*, use exactly the same abstract definitions, but derive different POS values for each word.

In this example, it is interesting to note that the POS inference model is specified in one place. It could easily be changed, for example to index on the previous three parts-of-speech, or to use word forms instead of parts-of-speech etc., and could be overridden with a specialised definition for subclasses of words (for example, open class versus

[4]Here DATR variables are used to range over all possible POS tags associated with nouns and variables.

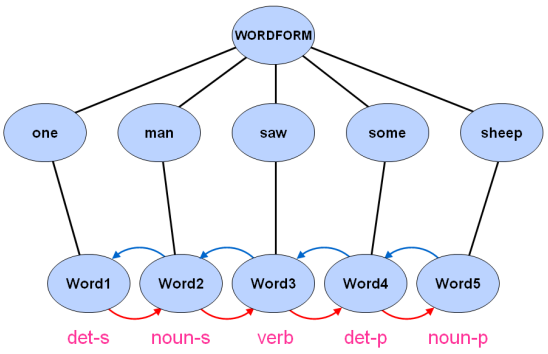

Figure 10: POS mapping for *one man saw some sheep*

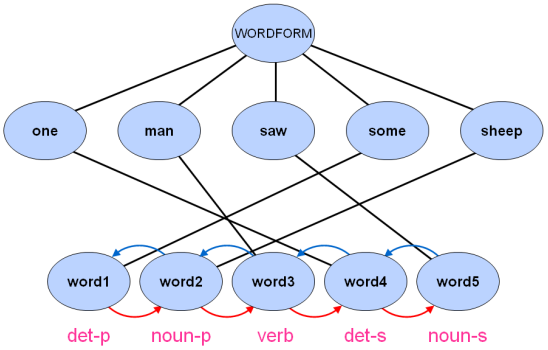

Figure 11: POS mapping for *some sheep man one saw*

closed class words). In addition the table definitions can take advantage of both the longest subpath principle (ignoring previous context they do not care about), and the inheritance hierarchy to produce compact yet highly detailed POS mappings.

3.2 Syntactic recognition

Similar techniques can also be applied to the task of syntactic recognition, which we exemplify for the case of regular languages (Hopcroft and Ullman, 1979)[5]. A simple approach is to take a context-free grammar for a regular language and transform it into left-regular form, where every rule has a rightmost lexical daughter and at most one other non-lexical daughter. Figure 12 illustrates the process for a simple context-free grammar. The key steps are expanding any non-lexical final daughters, introducing new non-terminals to make rules binary, and weeding out redundant productions.

In this form, the grammar rules can be transformed into lexical features. For example, the rule `S → NP VI` can be interpreted as "VI completes an S if the previous word completes an NP". This can be captured by introducing a path `<completes $cat>` (for any non-terminal category `$cat`), which is true for an instance node if that instance is the final lexical item of the corresponding non-terminal. Then the feature definitions in figure 13 correspond to the application of the rules. The root node specifies that by default all `<completes>` paths are `false`. The word form nodes have two kinds of definitions: for lexical categories or unary productions, simply set the corresponding `<completes>` path `true`. For binary productions, this instance completes the parent category if the previous instance completes the left daughter.

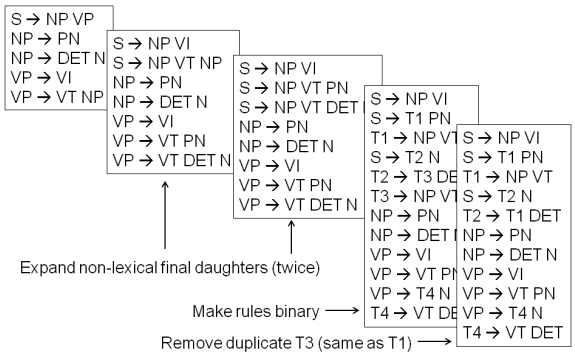

Figure 12: Transforming a grammar into left-regular form

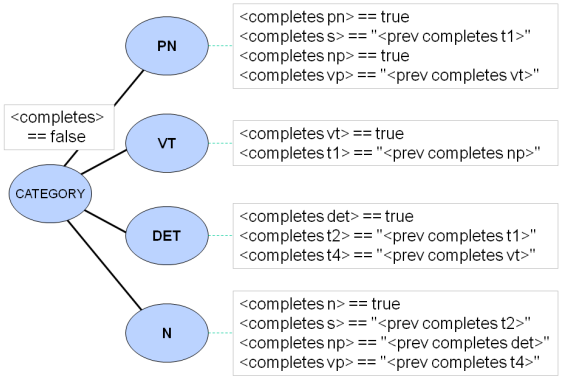

Figure 13: The grammar implemented as recognition features

Finally figure 14 shows the effect of these rules in a simple sentence *john saw the man*. Each instance now has binary features corresponding to all the categories recognised as terminating at that instance.

[5]The techniques described in this paper are at least powerful enough to recognise $a^n b^n$, a non-regular language.

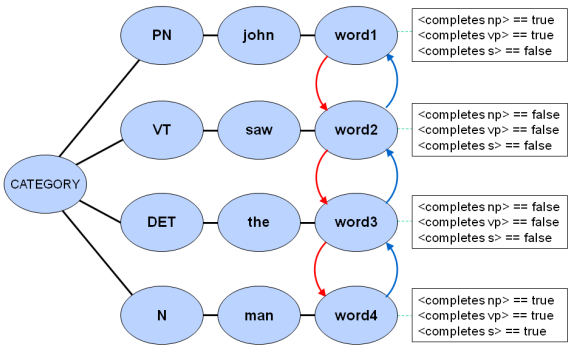

Figure 14: Recognising phrases in *john saw the man*

4 Discussion

The examples above show some of the potential for the Extended Lexicon approach: with a quite small change to the notion of a lexical entry, substantive language processing tasks can be construed as lexical description. Lexical description languages like DATR were designed to bring order to a domain, the lexicon, which exhibits quite a lot of apparent disorder – many regularities, but also sub-regularities, irregularities, strange corner-cases etc..[6] In the Extended Lexicon we bring those descriptive techniques to bear on language processing tasks more broadly, implicitly claiming that grammar is less orderly than grammarians sometimes suggest. Of course, statistical approaches to language processing have similar goals: statistical techniques are a powerful tool for modelling 'messy' systems. And indeed many properties of our model have statistical echoes: inheritance relations which provide symbolic analogues of backing off or smoothing etc. In the POS example above, the processing task was mapped to a table lookup distributed across a lexical hierarchy. An interesting next step would be to learn that table from corpus data, identifying how much context was required for different situations, how to generalise effectively from individual cases etc. This would be empirically-based but purely symbolic NLP.

The approach taken here has some similarities to various forms of Dependency Grammar (Mel'cuk, 1988), in particular because it does not include an explicit notion of phrase structure, in the grammatical sense. However, the Extended Lexicon is intended as a modelling tool, rather than a linguistic theory, and it has no explicit notion of dependency, or any kind of relationship beyond word adjacency.

Construction Grammar is a family of linguistic theories with a common theme that they do not make a sharp distinction between lexicon and grammar. Instead, they have a single framework which can represent words, phrases and sentences and can easily combine idiosyncratic phenomena with regular compositional processes. A recent manifestation of Construction Grammar, Sign-Based Construction Grammar, or SBCG (Boas and Sag, 2012), uses the unification-based type-theoretic framework of HPSG (Pollard and Sag, 1994) to provide a formal foundation for Construction Grammar. Although this framework is essentially monostratal in a similar way to the Extended Lexicon, it is far from lexically-oriented, making use of a considerable range of grammatical description mechanisms to constrain the overall behaviour of the system, in the same way that HPSG does. In essence it has absorbed the lexicon back into the grammar, rather than vice versa.

5 Future directions

The examples presented here are hugely simplified. In current work the Extended Lexicon approach is being applied to Text Mining and Sentiment Analysis, with a more sophisticated layered treatment of the relationships between instances. The core principle remains the same: that language can be described in terms of the behaviour of word instances and word adjacency relations, out of which the behaviour of whole sentences emerges.

Future directions for this work include exploring the use of corpora to build empirically-based Extended Lexicon systems; introducing non-deterministic and statistical processing into the system; and exploring the use of other 'topologies' for word instances – the word-string-based topology described here is appropriate for text processing, but other topologies, such as a lattice topology for speech recognition, or a bag-of-words topology for generation, are also possible.

[6] More recent work on lexical description, such as LMF (Francopoulo et al, 2006) and *lemon* (McCrae et al, 2012), is more concerned with representation and standardisation of surface lexical entries rather than deeper lexical generalisations, and uses less powerful inference mechanisms such as description logics.

References

Hans Boas and Ivan A. Sag (eds.). 2012. *Sign-Based Construction Grammar*. CSLI Publications, Stanford.

Eric Brill. 1992. "A simple rule-based part of speech tagger". In *Proceedings of the third conference on Applied Natural Language Processing (ANLC '92)*. Association for Computational Linguistics, Stroudsburg, PA, USA,

Roger Evans and Gerald Gazdar. 1996 "DATR: a Language for Lexical Knowledge Representation." *Computational Linguistics* , 22(2), pp. 167–216.

Gil Francopoulo, Monte George, Nicoletta Calzolari, Monica Monachini, Nuria Bel, Mandy Pet, and Claudia Soria 2006 "Lexical markup framework (LMF)." *International Conference on Language Resources and Evaluation – LREC 2006* Genoa

Roger Garside. 1987. "The CLAWS Word-tagging System." In: R. Garside, G. Leech and G. Sampson (eds), *The Computational Analysis of English: A Corpus-based Approach*. Longman, London.

John E. Hopcroft and Jeffrey D. Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley.

John McCrae, Guadalupe Aguado-de-Cea, Paul Buitelaar, Philipp Cimiano, Thierry Declerck, Asunción Gómez-Pérez, Jorge Gracia, Laura Hollink, Elena Montiel-Ponsoda, Dennis Spohr and Tobias Wunner, 2012. "Interchanging lexical resources on the semantic web." *Language Resources and Evaluation*, 46(4). pp. 701–719 Springer

Igor A. Mel'cuk. 1988. *Dependency Syntax: Theory and Practice*. State University of New York Pres, Albany, NY.

Carl J. Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago.