Part of Speech Tagging from a Logical Point of View

Torbjörn Lager[1] and Joakim Nivre[2]

[1] Uppsala University, Department of Linguistics, `Torbjorn.Lager@ling.uu.se`
[2] Växjö University, School of Mathematics and Systems Engineering, `Joakim.Nivre@msi.vxu.se`

Abstract. This paper presents logical reconstructions of four different methods for part of speech tagging: Finite State Intersection Grammar, HMM tagging, Brill tagging, and Constraint Grammar. Each reconstruction consists of a first-order logical theory and an inference relation that can be applied to the theory, in conjunction with a description of data, in order to solve the tagging problem. The reconstructed methods are compared along a number of dimensions including ontology, expressive power, mode of reasoning, uncertainty, underspecification, and robustness. It is argued that logical reconstruction of NLP methods in general can lead to a deeper understanding of the knowledge and reasoning involved, and of the ways in which different methods are related.

1 Introduction

Comparing different methods for solving a particular problem is a necessary and natural part of the development of any field of science or engineering. So also in the field of NLP, where, for example, newly proposed methods for part of speech tagging are compared to already existing ones.

Unfortunately, within the field of NLP, the use of different mathematical frameworks, or idiosyncratic formalisms and notations, tends to make such activities hard. It has led to a situation where different solutions to a problem are usually compared only on the level of performance, rather than on the level of knowledge representation and inference, and where attempts to combine different methods often fail to take advantage of their respective strengths.

In an attempt to find a remedy for this situation, we want to explore ways to reconstruct different NLP methods within a single framework. We choose to work with first order predicate logic (FOPL), often considered the *lingua franca* of knowledge representation. In this paper, we reconstruct four methods for part of speech tagging: Finite State Intersection Grammar, HMM tagging, Brill tagging, and Constraint Grammar.

The common framework will allow us to compare these four tagging methods with respect to issues such as the following: What kind of inference engines are these part of speech taggers? What kind of reasoning do they perform? What kind of knowledge do they exploit and how can the required knowledge be represented in first-order predicate logic? In this way, we hope to contribute to a better understanding of these particular methods and, thereby, also to demonstrate the usefulness of the methodology of logical reconstruction in general. (Further motivation for this kind of study can be found in section 2.3 below.)

2 Background

2.1 Reconstruction of Part of Speech Tagging

What does it mean to reconstruct a tagging method as a logical theory? For each method M, the reconstruction consists of two parts:

1. Specifying a FOPL theory T_M, representing the knowledge that the method uses in order to tag a sequence of words.
2. Specifying an inference relation I_M, such that the use of the inference relation for the representation will yield a solution to the tagging problem.

In addition, we need some way of representing sequences of words and their analyses. Let *yield*(s) denote the description of a sequence of words s, in the form of a set (or, equivalently, a conjunction) of ground, atomic sentences. For example, *yield*("the can smells") is:

word(1, the).   word(2, can).   word(3, smells).

Let *analysis*(s) denote the assignment of parts of speech to positions in s, again in the form of a set of ground, atomic sentences. For example, we assume that *analysis*("the can smells") is:

pos(1, dt).   pos(2, nn).   pos(3, vb).

Throughout the paper, we will use the simple example of tagging the string *the can smells*, which has the advantage of not requiring too much space. We will assume that the correct analysis is the one given above.

Finally, we note that, even if we adopt FOPL as our common framework, some of our reconstructions will only make use of a subset of the full system (e.g. Horn clause logic). We will also apply different inference strategies for different theories. However, this does not alter the fact that the methods are reconstructed within the same framework, although it may say something about the expressive power or inference mechanisms required for reconstructing different methods.

2.2 Ontology

A commitment to FOPL does not carry with it a commitment to a particular ontology. Before we continue, we therefore need to say something about our conceptualization of the objects, properties and relations in our domain. For each of the methods we will talk about three kinds of things:

1. Positions in a text (represented by integers)
2. Word forms
3. Part of speech tags

We have tried to be as uniform as possible here, and the only variation is that for one of the methods – the CG approach – we will also talk about sets of part of speech tags. Furthermore, for one of the methods – the HMM approach – we also need to refer to probabilities (represented as real numbers in the interval $[0, 1]$). Thus, positions, word forms, tags, sets of tags, and probabilities, are the individuals in our ontology; they will be values of variables and arguments of predicates in our theories.

As for properties and relations, all theories will include the predicates *word* and *pos* introduced in section 2.1, as well as the ordinary operations of arithmetic (notably $+$ and $-$). Additional properties and relations will differ from one theory to the other. In fact, the comparison will show that there is an interesting tradeoff between expressive power, ontological assumptions, and the kind of properties and relations distinguished, where abundance in one domain can make up for austerity in others.

2.3 Why Logical Reconstruction?

There are several reasons for wanting to logically reconstruct a method M (or a number of methods M_1 to M_n) for solving a particular NLP problem:

- It allows us – even forces us – to clarify and make more precise a formulation of a method which is vague and open to interpretation. And even if the original formulation of M is simple and clear enough, looking at it from another angle may bring new insights. In any case, it may advance our understanding of M, its strengths and weaknesses.
- It may facilitate the comparison of two different methods M_i and M_j for solving the same problem, especially on the level of knowledge representation and inference.
- It may facilitate the combination of two different methods M_i and M_j in a way that leads to a better overall solution to the problem at hand.
- It may put us in a position to find novel implementation techniques for M. In particular, T_M in combination with a theorem prover implementing I_M may turn out to be a perfectly practical implementation of M.
- It allows us to regard standard implementation techniques of M as special purpose inference engines for the kind of representations that T_M exemplifies.
- By factoring out the knowledge component T_M from a method M, we may be able to find interesting novel uses for T_M which M was not designed for in the first place.
- It may allow us to come up with better ideas for interfacing components dealing with different NLP tasks. For example, in the case of part of speech taggers, it may help us to discover new ways to interface a tagger with a lexicon, or a tagger with a parser.
- It may give us more to say about the acquisition of knowledge for a particular method M. For example, expressing the knowledge in a logical framework may enable us to use inductive logic programming methods.

In the sequel, we will return to these points as they become relevant in the discussion of different methods for part of speech tagging.

3 Part of Speech Tagging Reconstructed

3.1 Part of Speech Tagging as Deduction I

In Finite-State Intersection Grammar (Koskenniemi 1990), sequences of words, rules, and analyses are all represented as finite-state automata (FSAs). Applying the rules to a tagged sequence of word means intersecting the automaton representing the tagged sequence with the automata representing the rules. Each path through the resulting FSA represents a possible analysis of the word sequence.

The particular kind of logic that will be used to reconstruct this method has clauses of the following form:

$$a_1; \ldots ; a_k \leftarrow b_1, \ldots , b_n$$

That is, the consequent of a clause may consist of a disjunction of atomic formulas.[1] Explicit negative information can be given as follows:

$$\leftarrow b_1, \ldots , b_n$$

A collection of such clauses forms a disjunctive theory. This logic has the expressive power of full first order predicate logic.

It is well-known that every model of a logic program can be represented by a set of ground atomic formulas. Furthermore, a model M is a minimal model of a theory T if there exists no other model of T which is included (in the set-theoretic sense) in M. Whereas a theory in the language of pure Prolog always has exactly one unique minimal model, a disjunctive theory in general does not (see, e.g., Fernández and Minker 1992).

The (disjunctive) minimal model state of a theory T is the set of positive ground disjunctions all of whose minimal models satisfy T. Thus it provides a single, and often a very compact, representation of a set of minimal models. The minimal model state follows deductively from the set of constraints in union with the goal/query:

$$T \cup g \vdash mms$$

An interesting special case is:

$$T \cup g \vdash \perp$$

This means that $T \cup g$ is inconsistent, and thus has no model.

Theorem provers have been built – so called model generation theorem provers – which given a set of sentences in a disjunctive logic are able to generate (representations of) the corresponding set of minimal models or the corresponding minimal model state. One example is DisLog (Seipel and Thöne 1994).

In Lager (1998), it was proposed that the essence of the knowledge available to a FSIG tagger is captured by a disjunctive theory, such as the following:

[1] For reasons we need not touch upon here, clauses must be range-restricted, which means that all variables occurring in a clause must occur in at least one of the body atoms $b_1, \ldots , b_n$.

```
pos(P, dt) ← word(P, the).
pos(P, nn) ; pos(P,vb) ← word(P, can).
pos(P, nn) ; pos(P,vb) ← word(P, smells).

← pos(P1, dt), P2 is P1+1, pos(P2, vb).
← pos(P1, nn), P2 is P1+1, pos(P2, nn).
```

Tagging consists in trying to find a minimal model state corresponding to a non-empty set of minimal models for the grammar in union with $\mathit{yield}(s)$.

$$T \cup \mathit{yield}(s) \vdash \mathit{analysis}(s)$$

The parts of speech of the individual words can be read off from the minimal model state. In this framework, part of speech tagging can fail, and this happens when

$$T \cup \mathit{yield}(s) \vdash \perp$$

i.e., when the constraints in union with the description of the input has no model. In Finite-State Intersection Grammar, this corresponds to an FSA with just one non-final start state with no outgoing transitions.

Tagging may also fail to eliminate all ambiguity, in which case the minimal model state will contain disjunctive formulas. For example, the theory

```
pos(P, pn) ; pos(P, dt) ← word(P, what).
pos(P, nn) ; pos(P, vb) ← word(P, question).

← pos(P1, dt), P2 is P1+1, pos(P2, vb).

word(1, what).    word(2, question).
```

has three minimal models. The corresponding unique minimal model state is

```
pos(1, dt) ; pos(1, pn).      word(1, what).
pos(2, vb) ; pos(2, nn).      word(2, question).
pos(1, pn) ; pos(2, nn).
```

In Finite-State Intersection Grammar, this corresponds to a determinized and minimized FSA with three paths leading from the start state to the final state, one path for each analysis.

This way of axiomatizing part of speech tagging has the same properties as Finite State Intersection Grammar in terms of uniformity of representations, order independence, (lack of) robustness, and 'packing' of analysis results. In a way, it is also the simplest and most natural way of using logic to capture the fundamental flow of information in part of speech tagging, and – as we shall see – an interesting point of departure for comparisons with other approaches.

3.2 Part of Speech Tagging as Probabilistic Abduction

In a typical Hidden Markov Model (HMM) for part of speech tagging there are states (representing parts of speech or sequences thereof), transitions between states, and symbols (words) emitted by the states. There are two kinds of probabilities associated with a HMM: transition probabilities, i.e. the probability of a transition from one state to another, and output probabilities, i.e. the probability of a certain state emitting a certain word. An HMM tagger tries to find the sequence of state transitions most likely to have generated the input sequence of words.

The key idea here is that if we conceive of the sequence of words as an observation and the sequence of state transitions as a possible explanation for the observation, then HMM tagging has a very abductive flavor. Consequently, HMM tagging will be formalized using the framework of Probabilistic Horn Abduction, which is a simple framework for knowledge representation in Horn-clause logic (Poole 1993a), and for using a special purpose inference engine to reason with it (Poole 1993b). The system is capable of selecting the most probable abductive explanation among the competing ones. The following brief description of the framework closely follows Poole (1993a), to which the reader is referred for further information.

A definite clause is of the form

$$a \leftarrow b_1, \ldots, b_n$$

where a and each b_i are atomic symbols. A disjoint declaration is of the form

$$\text{disjoint}([h_1 : p_1, \ldots, h_n : p_n]).$$

where each h_i is atomic, and each p_i is a real number $0 \leq p_i \leq 1$ such that $\sum_i p_i = 1$. Any variable appearing in one h_i must appear in all of the h_i (i.e., all the h_i share the same variables). The h_i are referred to as possible hypotheses.

A probabilistic Horn abduction theory is a collection of definite clauses and disjoint declarations such that if a ground atom h is an instance of a possible hypothesis in one disjoint declaration, then it is not an instance of another hypothesis in any of the disjoint declarations, nor is it an instance of the head of any clause.

If g is a closed formula describing an observation, an explanation of g from a probabilistic Horn abduction theory is a set of hypotheses $H = \{h_1, \ldots, h_n\}$, where each h_i is a ground instance of a possible hypothesis. Each explanation H is associated with a probability $P(H)$. In fact, all of the probabilistic calculations in the system reduce to finding the probabilities of explanations. More precisely, the probability of an explanation H is computed by multiplying the probabilities of the hypotheses generated, as follows:

$$P(H) = P(h_1, \ldots, h_n) = \prod_{i=1}^{n} P(h_i)$$

In terms of probability theory, this amounts to the assumption that all the hypotheses in an explanation are mutually independent.

The probabilistic model of an HMM tagger is represented by clauses of the following form (where the first set of clauses describes relations between states and symbols, while the second set defines state transitions):

```
word(P, the) ← pos(P, dt), etc(P, the, dt).
word(P, can) ← pos(P, vb), etc(P, can, vb).
word(P, smells) ← pos(P, vb), etc(P, smells, vb).
word(P, can) ← pos(P, nn), etc(P, can, nn).
word(P, smells) ← pos(P, nn), etc(P, smells, nn).

pos(P, dt) ← P1 is P-1, pos(P1, start), etc(P1, dt, start).
pos(P, vb) ← P1 is P-1, pos(P1, dt), etc(P1, vb, dt).
pos(P, nn) ← P1 is P-1, pos(P1, dt), etc(P1, nn, dt).
pos(P, vb) ← P1 is P-1, pos(P1, vb), etc(P1, vb, vb).
pos(P, vb) ← P1 is P-1, pos(P1, nn), etc(P1, vb, nn).
```

The *etc* goals can be thought of as encoding the extra conditions, the satisfaction of which guarantees the truth of the conclusion. These goals can never be proven deductively but only assumed with a certain probability. The probabilities are defined by disjoint declarations of the following kind:

```
disjoint([etc(P, the, dt):1.00]).
disjoint([etc(P, can, nn):0.70, etc(P, smells, nn):0.30]).
disjoint([etc(P, can, vb):0.90, etc(P, smells, vb):0.10]).

disjoint([etc(P, dt, start):1.00]).
disjoint([etc(P, vb, dt):0.01, etc(P, nn, dt:0.99)]).
disjoint([etc(P, vb, nn):0.80, etc(P, nn, nn):0.20]).
disjoint([etc(P, vb, vb):0.20, etc(P, nn, vb):0.80]).
```

The first set of disjoint declarations defines the output probabilities of the corresponding HMM, while the second set defines transition probabilities.

Given a theory of this form, part of speech tagging consists in trying to abductively prove the goal $\mathit{yield}(s)$, i.e. trying to find a set of hypotheses $\mathit{analysis}(s)$ such that:

$$T \cup \mathit{analysis}(s) \vdash \mathit{yield}(s)$$

$$P(\mathit{analysis}(s)) = P(h_1, \ldots, h_n) = \max_{h_1,\ldots,h_n} \prod_{i=1}^{n} = P(h_i)$$

The part of speech tags corresponding to the yield can be read off from the abductive explanation.

Often, logically reconstructing a problem solving method may allow us to implement it in a cleaner way, and sometimes even more efficiently. This has happened before, witness research in automatic planning, where high performance deduction systems are able to outperform special purpose planning systems. In the case of our reconstruction of a statistical part of speech tagger however, it appears that it would make little sense to use a general theorem-prover – and an abductive one at that – for a task for which another very elegant and efficient procedure – the Viterbi algorithm (Viterbi 1967) – is known.

In this case, it is perhaps more interesting to take the opposite perspective, and instead regard the Viterbi algorithm as an algorithm for abduction. As such, it is able to perform a best-first search for the most probable set of hypotheses consistent with the observations, and it is able to do this very efficiently – in time proportional to the length of the word sequence. Actually, we would like to suggest that since the kind of HMM part of speech tagger described in this paper is commonly regarded as the canonical part of speech tagger (Charniak 1997) then tagging as probabilistic abduction has already been used for tagging millions and millions of words from running texts.

Before concluding this section, we will take up one other issue discussed in section 2.3, namely the claim that reconstructing different theories in one formal framework may enable us to see more clearly how they can be combined in a principled way. For example, the reconstructions so far have shown that the knowledge exploited by the HMM approach and the FSIG approach, albeit formalized differently, can nevertheless be related, and that it should therefore be possible to combine the two methods. Let us briefly consider how this could be done.

A simple way of combining an HMM-like and an FSIG-like tagger is by adding a set C of integrity constraints to the HMM framework, explicitly expressed as FOPL clauses. (Adding integrity constraints to an abductive framework is a common strategy for reducing the number of possible explanations.) Presumably, we would like to have clauses in C expressing the kind of knowledge that an HMM cannot (easily) capture, e.g. constraints such as “a sentence must contain a finite verb”.

The inference procedure to go with this is once again abduction, but of a slightly more general form than before. Part of speech tagging consists in trying to find a set of hypotheses *analysis*(s) such that:

$$T \cup \mathit{analysis}(s) \vdash \mathit{yield}(s)$$

$$T \cup C \cup \mathit{analysis}(s) \nvdash \bot$$

$$P(\mathit{analysis}(s)) = P(h_1, \ldots, h_n) = \max_{h_1,\ldots,h_n} \prod_{i=1}^{n} P(h_i)$$

Here, C will effectively constrain *analysis*(s). Just as before, the result can be read off from the abductive explanation.

The combined system can be implemented as follows. At compile time, build automata corresponding to the constraints in C and compute their intersection. At runtime, build an automaton representing the result of performing lexical lookup of each word in the input sequence of words. (At the same time, mark each part of speech link with its lexical probability.) Intersect this automaton with the automaton representing C. The resulting automaton represents the set of minimal models of the constraints in union with the description of the input sequence of words. Finally, use the Viterbi algorithm to compute the most probable path through the automaton.[2]

[2] In the same way as our original reconstruction of the FSIG tagger in section 3.1, this tagger will not be robust, since the set of minimal models may turn out to be empty. However, in the combined system we can always fall back on ordinary HMM tagging and apply the Viterbi algorithm to the original input in case this should happen.

3.3 Part of Speech Tagging as Deduction II

Brill tagging (Brill 1995) consists in assigning default initial tags to each word (usually the most frequent tag for that word), and then applying, in a sequence, replacement rules such as the following:

replace tag *vb* with *nn* if the previous tag is *dt*
replace tag *nn* with *vb* if the previous tag is *nn*

The secret behind the effectiveness of a Brill tagger is that rules that are very effective in the sense that they correct a lot of errors, but also are responsible for introducing new ones, are placed early in the sequence. Rules later in the sequence are not as effective, but they tend to correct some of the errors made by rules earlier in the sequence.[3]

As shown in Lager (1999), a Horn clause theory enhanced with negation can be used to represent the knowledge available to a Brill tagger:[4]

pos(P, T) ← pos3(P, T).

pos3(P, vb) ← pos2(P, nn), P1 is P-1, pos2(P1, nn).
pos3(P, T) ← pos2(P, T), P1 is P-1, ¬ pos2(P1, nn).

pos2(P, nn) ← pos1(P, vb), P1 is P-1, pos1(P1, dt).
pos2(P, T) ← pos1(P, T), P1 is P-1, ¬ pos1(P1, dt).

pos1(P, dt) ← word(P, the).
pos1(P, vb) ← word(P, can).
pos1(P, nn) ← word(P, smells).

The idea is that for each rule in the above sequence of rules a new predicate pos_i is introduced, where the number i indicates where in the sequence the rule belongs. Semantically, pos_i relates a position to a part of speech, and the formulas define this predicate in terms of the predicate pos_{i-1} plus a number of other predicates. Each pos_i corresponding to a replacement rule is defined by two clauses – one stating the conditions under which a part of speech tag is replaced with another part of speech tag, the other one stating the conditions under which the old tag is kept.

The purpose of this scheme is to simulate, in pure logic, the effect of composing a number of contextual replacement rules. Order is important, in that a rule in the sequence is applied to the output of rules earlier in the sequence and feed rules later in the sequence.

[3] The Transformation-Based Learning algorithm is responsible for placing the rules in this order.

[4] To simulate classical negation, we can for example rely on Negation as Failure (NAF). However, note that whereas NAF can be regarded as a non-monotonic reasoning operator by means of which default reasoning can be implemented, it is not used in this way here. The reasoning performed here is pure classical deduction. Furthermore, since there is no recursion through negation, the logic program still has a unique minimal model.

The analysis in this case is entailed by the theory in conjunction with the description of the input sequence of words:

$$T \cup \mathit{yield}(s) \vdash \mathit{analysis}(s)$$

In fact, Brill tagging can be modeled by performing an ordinary Prolog style constructive proof of the goal statement *pos*(P, T) (for all positions P in *s*) from the theory. This provides values for goal variables, which constitute the output of the tagging process.

This is tagging as deduction, and what is more, it is highly practical. In Lager (2000) it was shown that by converting a sequence of 280 Brill tagging rules into this format,[5] applying conventional memoing techniques in order to avoid unnecessary recomputations, and running the whole thing in Prolog, one can build a part of speech tagger capable of tagging a decent 350 words per second on a 166MHz PC. This serves to illustrate one of the points made earlier: The logical reconstruction of an NLP method may result in novel and interesting implementation techniques.

It was also shown that the resulting program can be used, not only for identifying the parts of speech of particular words, but also for searching for words having a particular parts-of-speech. It all hinges on the mode in which we choose to call the predicate. This illustrates yet another point from section 2.3, namely that the very same theory can be used in different ways, for different purposes.

3.4 Part of Speech Tagging as Deduction III

In a Constraint Grammar (CG) tagger (Karlsson *et al* 1995), a lexical lookup module assigns sets of alternative tags to occurrences of words, disregarding context. A rule application module then removes tags from such sets, on the basis of what appears in the local context, using rules such as these:

remove tag *vb* if the previous tag is *dt*
remove tag *nn* if the previous tag is *nn*

However, in order to guarantee that each word token is left with at least one tag, the rule application module adheres to the following principle: "Don't remove the last remaining tag."

A CG tagger can be made to almost never remove the correct, intended tag from a word, which leads to a high recall. However, it is also characteristic of a CG tagger that it does not always resolve all ambiguity introduced in the lexical lookup process, and this means that the precision of such a tagger may be low.

Just like Brill tagging, CG tagging can be construed as the application of an ordered sequence of contextually triggered rules, and thus the approach to axiomatization used in the previous section can be used again:

[5] Actually, Prolog's if-then-else was used instead of negation as failure, but this amounts to the same thing.

```
pos(P, T) ← pos3(P, T).

pos3(P, T) ← pos2(P, T0), diff(T0, [nn], T), T ≠ [ ], P1 is P-1, pos2(P1, [nn]).
pos3(P, T) ← pos2(P, T), (singleton(T) ; P1 is P-1, ¬ pos2(P1, [nn])).

pos2(P, T) ← pos1(P, T0), diff(T0, [vb], T), T ≠ [ ], P1 is P-1, pos1(P1, [dt]).
pos2(P, T) ← pos1(P, T), (singleton(T) ; P1 is P-1, ¬ pos1(P1, [dt])).

pos1(P, [dt]) ← word(P, the).
pos1(P, [nn,vb]) ← word(P, can).
pos1(P, [nn,vb]) ← word(P, smells).
```

Here, *diff* is a predicate that implements set difference. The "don't remove the last tag" principle is implemented by checking that the result of applying *diff* is different from the empty set before removing any tag, and by allowing any singleton to be left intact regardless of its environment.

Just as with the Brill tagger, we are interested in what is entailed by the theory in conjunction with the description of the input sequence of words:

$$T \cup \mathit{yield}(s) \vdash \mathit{analysis}(s)$$

And here too, simple Prolog style deduction is sufficient. For our simple example, we are able to prove $pos(1, [\mathrm{dt}])$, $pos(2, [\mathrm{nn}])$, and $pos(3, [\mathrm{vb}])$, but nothing more.

In a Constraint Grammar we find other kinds of rules as well. For example, consider the following (sub-)sequence:

select tag *jj* if the word is *high* and the next tag is *nn*
remove tag *vb* if the word is *table*
remove tag *wbs*

The first rule removes all tags except *jj* if the given conditions are satisfied. Assuming that the lexical lookup module assigns the set [*nn*,*vb*] to the word *table*, the second rule removes the *vb* tag again unless some rule earlier in the sequence has already removed the *nn* tag. The third rule unconditionally removes the *wbs* tag unless it is the last remaining tag, something which is a sensible thing to do if *wbs* is a very uncommon tag. The order dependence between rules is very evident here, but all of these rules can be logically reconstructed along the lines given above.

Also, the right-hand sides of CG rules may consist of very complex logical conditions looking very far into the left and/or right context of the word to be disambiguated. This means that very difficult disambiguation tasks, which may be beyond the reach of other methods, can be solved in the CG framework by writing very specific rules.

Although reconstructing Constraint Grammar in this way amounts to a claim that CG tagging is in certain respects similar to Brill tagging, there are important differences as well. For example, a CG tagger cannot, once a tag is removed from a set of tags, rely on rules later in the sequence to add it back in again. Therefore, we will have to be very certain about each rule, since if it makes errors a drop in recall will result. Our experience from writing disambiguation rules in the CG style tells us that it is fairly straightforward to write such rules. The downside, of course, is that thousands of rules are needed and that coding them by hand is tedious and time consuming. Reconstructing CG in logic may help us find a way to learn them automatically from tagged corpora.

Whereas the reconstruction of Brill tagging is very faithful to the original, the Constraint Grammar reconstruction may be less so. We feel we have given a reasonable logical interpretation of the essence of Constraint Grammar, but at the same time we admit that the originators perhaps would not agree. On the other hand, this only illustrates yet another one of our points. Logical reconstruction serves to make the formulation of a method more precise, and less open to interpretation. (This of course is just the usual argument for wanting to formalize something.)

4 Discussion

Having reconstructed four different methods of part-of-speech tagging, we first note that whereas all four taggers start from the same formula – from a description of a sequence of words – the HMM tagger solves the tagging problem using abductive reasoning, while the three other taggers perform deduction. We have captured the difference as follows:

$$T \cup \mathit{analysis}(s) \vdash \mathit{yield}(s)$$

$$T \cup \mathit{yield}(s) \vdash \mathit{analysis}(s)$$

The representation of knowledge that each of the described approaches requires is determined by how it is, so to speak, driven by the mode of reasoning. Let us consider, for example, the representation of lexical knowledge, i.e., knowledge of the relation between words and their parts of speech, and let us begin by contrasting the FSIG tagger with the HMM tagger.

Since an FSIG tagger is driven by deduction, it is natural to represent the association between words and tags in a way which supports inference from words to tags, thus:

$$\text{pos(P, vb) ; pos(P, nn)} \leftarrow \text{word(P, can).}$$

Since the statistical tagger, on the other hand, is driven by abduction, the natural form of representation is one which supports inference from tags to words:

$$\text{word(P, can)} \leftarrow \text{pos(P, vb), etc(P, can, vb).}$$
$$\text{word(P, can)} \leftarrow \text{pos(P, nn), etc(P, can, nn).}$$

A moment of reflection reveals that some of what we know about the association between words and parts of speech can be expressed in the form of equivalences such as the following:

$$\text{word(P, can)} \leftrightarrow \text{(pos(P, vb), etc(P, can, vb)) ; (pos(P, nn), etc(P, can, nn)).}$$

The FSIG representation follows logically from this equivalence, and so does the HMM representation. But they capture different parts, where neither follows logically from the other. Thus, in this respect, FSIG tagging and HMM tagging may be said to exploit the same kind of knowledge, but to represent it in different (but not incompatible) ways.

At first sight, the lexical representations of the Brill tagger looks similar to those of the FSIG tagger, since they support reasoning from words to tags:

$$\text{pos1}(\text{P}, \text{vb}) \leftarrow \text{word}(\text{P}, \text{can}).$$

It is important to note, though, that the predicate pos_1 appearing in the head of this clause is not the same as the predicate *pos* appearing in the HMM and FSIG representations (as well as in the final analysis). Thus, the formula above is not a statement about a word and its actual part(s) of speech; it is a statement about a word and its default part of speech. Taken as a statement of the former kind it would simply be false.

This points to a crucial difference between the representation of lexical knowledge in HMM and FSIG, on the one hand, and Brill tagging, on the other. Whereas the former two theories contain statements which purport to be true about words and their actual parts of speech, and which may therefore be evaluated in the absence of any other knowledge (e.g., knowledge about constraints on possible sequences of parts of speech), the latter theory is only meaningful in relation to another theory somehow relating default tags to actual tags and can therefore not be evaluated in isolation.

In Constraint Grammar, finally, lexical rules also support reasoning from words to tags. In contrast to the Brill tagger, but in accordance with the FSIG tagger, the CG lexical rules also say something which purports to be true about words and their parts of speech, although this time expressed in terms of sets of tags, rather than using disjunction.

We can see in these different representations four different ways to handle the uncertainty that pervades knowledge about language. The FSIG approach uses disjunction to represent ambiguity in the relation between words and tags. By contrast, the HMM approach uses a weak logical formalism in which virtually every statement about relations between words and tags would be too strong, if it were not for the probabilities that can be associated with them.

The Brill tagging approach is similar to the HMM approach in that it avoids the use of disjunction and formulates statements that prima facie are too strong. But these statements are not taken to be probabilistic in nature. Instead, they are taken to express defaults, that can be overridden by other parts of the theory. Finally, although the CG approach is similar to the Brill approach in certain ways, it does not handle uncertainty in the same way. Rather, it relies on its different ontology – a world containing sets of tags – in such a way that the uncertainty that possibly remains after we have applied a sequence of rules is captured, not by disjunctions as in FSIG, but by sets that contain more than one tag. Thus, this is an example of how a richer ontology can compensate for lack of expressive power in the formalism.

However, using this simple strategy, the CG approach cannot cope with all the uncertainty that the FSIG approach can cope with. For example, consider again the phrase *what question*, where *what* can be either a determiner or a pronoun, and where *question* can be either a noun or a verb. Given a constraint that rules out determiner followed by verb, there still remains three alternatives (*dt*+*nn*, *pn*+*nn*, and *pn*+*vb*). This poses no problem for the FSIG approach (cf. section 3.1), but the CG approach cannot even represent it, since removing a tag from either word leaves us with only two remaining alternatives.

It is well known that "the expressive power of FOPL determines not so much what can be said, but what can be left unsaid" (Levesque and Brachman 1985). This is exactly the reason why disjunction and negation are used in the FSIG approach: the linguistic knowledge is uncertain and disjunction and negation are two of the instruments provided by FOPL for representing uncertain knowledge.

However, it can be argued that disjunction and negation are blunt instruments, which cannot cope with all subtleties involved in reasoning under uncertainty and thus that we need probability theory as well. That is why probability theory is used in the HMM approach, while the Brill tagging approach uses its fairly weak expressive power to formulate layers of defaults, exceptions to defaults, exceptions to exceptions to defaults, etc.

On the other hand, in the context of sufficient additional information, sound conclusions can perhaps be drawn that (occasionally) lead us to the only possible analyses, not just the most probable ones, and here we have another approach to coping with uncertainty: Just throw more knowledge at it, and it may go away! The moral implicit in this strategy is: "Don't guess if you know." (Tapanainen and Voutilainen 1994). This is the strategy used in both FSIG and CG, while the representations used in HMM tagging and Brill tagging seems to be less amenable to such an approach.

However, the unmitigated use of this strategy is not without drawbacks either. Both FSIG and CG may fail to resolve all ambiguities in the input, in cases where the information available is insufficient, and FSIG may even fail to produce an analysis at all because all the alternatives are eliminated (a pitfall that CG avoids through the "don't remove the last tag" principle). By contrast, both HMM and Brill always assign one and only one tag to each word.

We summarize our comparison in Table 1, where we contrast our reconstructions of the four methods along the dimensions of ontology, expressive power, mode of reasoning, uncertainty, underspecification, and robustness.

5 Conclusion

In this paper, we have presented logical reconstructions of four different methods for part of speech tagging: Finite State Intersection Grammar, HMM tagging, Brill tagging, and Constraint Grammar. Each reconstruction consists of a first-order logical theory and an inference relation that can be applied to the theory, in conjunction with a description of data, in order to solve the tagging problem. We hope to have provided a "proof of concept" for the methodology of logical

Table 1. Comparison, on the level of representation and inference, between four methods for part of speech tagging.

	FSIG	**HMM**	**BRILL**	**CG**
Ontology: What kind of entities does the tagger reason over?	positions words tags	positions words tags probabilities	positions words tags	positions words sets of tags
Expressive means: What do we need in the language in order to simulate the method?	disjunctive logic with negation	Horn clause logic	Horn clause logic with negation in antecedent	Horn clause logic with negation in antecedent
Mode of reasoning	deductive	abductive	deductive	deductive
How does the method deal with uncertainty?	disjunction negation	probabilities	default reasoning	sets of tags
Underspecification: Does the final analysis represent remaining ambiguity?	yes (disjunction)	no	no	yes (sets of tags)
Robustness: Does the method always produce an analysis?	no	yes	yes	yes

reconstruction, in particular by showing how the reconstructed methods can be compared along a number of dimensions including ontology, expressive power, mode of reasoning, uncertainty, underspecification, and robustness, but also, by way of small examples, how the methodology of logical reconstruction may allow us to discover more fruitful ways of combining methods, to explore novel implementation techniques, and more generally, to increase our understanding of the issues involved when knowledge about language is applied to language in use.

References

Brill, E. (1995) Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part of Speech Tagging. *Computational Linguistics* 21, 543–565.

Charniak, E. (1997) Statistical Techniques for Natural Language Parsing. *AI Magazine* 18(4), 33–44.

Fernández, J. A. and Minker, J. (1992) Disjunctive Deductive Databases. In *Proceedings of the Logic Programming and Automated Reasoning Conference.*

Karlsson, F., Voutilainen, A., Heikkilä, J., Anttila, A. (eds) (1995) *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text.* Mouton de Gruyter.

Koskenniemi, K. (1990) Finite-State Parsing and Disambiguation. In *Proceedings of COLING'90*, Helsinki, Finland.

Lager, T. (1998) Logic for Part of Speech Tagging and Shallow Parsing. In *Proceedings of NODALIDA'98*, Copenhagen, Denmark.

Lager, T. (1999) The μ-TBL System: Logic Programming Tools for Transformation-Based Learning. In *Proceedings of CoNLL'99*, Bergen, Norway.

Lager, T. (2000) A Logic Programming Approach to Word Expert Engineering. In *Proceedings of ACIDCA 2000: Workshop on Corpora and Natural Language Processing*, Monastir, Tunisia.

Levesque, H. and Brachman, R. (1985) A Fundamental Tradeoff in Knowledge Representation and Reasoning. In Brachman, R. and Reiter, H. (eds) *Readings in Knowledge Representation*. Morgan Kaufman.

Poole, D. (1993a) Probabilistic Horn Abduction and Bayesian Networks. *Artificial Intelligence* 64(1), 81–129.

Poole, D. (1993b) Logic Programming, Abduction and Probability: A Top-Down Anytime Algorithm for Computing Prior and Posterior Probabilities. *New Generation Computing*, 11(3-4), 377–400.

Seipel, D. and Thöne, H. (1994) DisLog – A System for Reasoning in Disjunctive Deductive Databases. In *Proceedings of the International Workshop on the Deductive Approach to Information Systems and Databases 1994 (DAISD'94)*.

Tapanainen, P. and Voutilainen, A. (1994) Tagging Accurately – Don't Guess If You Know. In *Proceedings of the Fourth Conference on Applied Natural Language Processing*, Stuttgart, Germany.

Viterbi, A. J. (1967) Error Bounds for Convolutional Codes and an Asymptotically Optimal Algorithm. *IEEE Transactions on Information Theory* IT-13(2), 260–269.