Lexical Disambiguation with Fine-Grained Tagsets

Éric Laporte[1]

October 30, 1995

[1]Institut Gaspard-Monge, Université de Marne-la-Vallée, 2, rue de la Butte-Verte, F-93166 Noisy-le-Grand CEDEX, France.

Contents

Abstract

We describe the formal mathematical models underlying two constraint-based, finite-state methods for lexical disambiguation with fine-grained tagsets. They are more powerful variants of the methods described by Roche 1992 and Silberztein 1993. Both have the full theoretical expressive power of finite-state devices.

1

Motivation

Methods for lexical tagging and lexical disambiguation are useful for virtually all types of NLP systems, since they contribute to identifying elementary tokens in texts. From a computational point of view, tagging is often a prerequisite to further structural analysis (Milne 1986). Additionally, many systems make use of tagged corpora. However, tagging and disambiguation continue to raise serious challenges. Methods based on statistical models of language are well-known (Church 1988, Kupiec 1992, Charniak et al. 1993, Weischedel et al. 1993), but another significant part of recent research in lexical tagging (Silberztein 1989, Koskenniemi 1990, Rimon and Herz 1991, Paulussen and Martin 1992, Roche 1992, Tapanainen and Voutilainen 1993, Silberztein 1994) is devoted to an alternative approach which is characterized by three features:

- it involves the use of constraint-based rules that often have to be written down by hand;
- several solutions are produced when several are correct or when the only correct one cannot be found. They are represented as a directed acyclic graph (DAG) or finite-state network;
- the computation is performed by a finite-state device.

Lexical taggers of this type are not easily comparable with state-of-the-art statistical taggers. They have zero silence rate, i.e. they never eliminate the correct solution(s), or the constraint-based rules could be written so that they do, even with fine-grained tagsets. On the other hand, when they are used with fine-grained tagsets, they have high noise rates. Actually, with the most fine-grained existing tagsets in English and in French, if the initial phase of tagging consists of a lexicon lookup, the ambiguity rate before disambiguation commonly averages 2.5 tags/word: the selection of only one tagging becomes a real challenge and cannot realistically be regarded as compatible with a zero silence rate.

One of the incentives to investigate into this direction is the fact that lexical taggers with a zero silence rate would be useful front-ends to parsers: a parser is usually in a better position to rule out wrong hypotheses than to generate alternatives to its own input. Moreover, parsers will probably require more and more detailed lexical information in order to recognize syntactic structures (Roche 1994), i.e. more and more fine-grained tagsets.

Constraint-based, finite-state lexical taggers can be compared with one another. The usual criteria involve error rates and other rates, which obviously depend on the hand-written rules. Another criterion of comparison can be defined: the theoretical expressive power of the rule formalism. This criterion is independent of particular hand-written rules.

Two constraint-based methods for lexical disambiguation with finite-state computation and fine-grained tagsets have been recently implemented. One of them (Roche 1992) is a negative method: the rules describe forbidden sequences of tags. The other is a function of INTEX (Silberztein 1994). It is imperative: the rules contain a condition part and an action part. The expressive power of such systems is theoretically limited by the exclusive use of finite-state computational devices, but this theoretical limit now appears as reasonable. We extended these systems in order to give them the maximum expressive power inside the limits imposed by the principle of a finite-state computation. In this paper, we show that with a slight modification that does not affect time complexity, Roche's and Silberztein's proposals get the full expressive power of finite-state computational devices.

2

Brief description of two lexical disambiguators

In what follows we make a formal description of our variants of Roche's and Silberztein's disambiguators. The two tagsets are equivalent. They provide detailed grammatical information, i.e. not only part of speech and inflectional features, but also

- information needed to obtain the lemma from the word form,
- inflectional pattern codes, e.g. conjugation codes, and
- delimitation of compounds and idioms, in the form of specific tags assigned to compounds, e.g. *Adverb* for *of course.*

The lexicon assigns one or several tags to each (simple or compound) word form. The 3,800 tags and 828,000 (simple or compound) word forms make up 1,250,000 <word, tag> pairs. Let T be this set of <word, tag> pairs. Thus after lexicon lookup, each of the taggings is an element of $T^\star$. For a sentence s, let $Init(s) \subset T^\star$ be the set of such initial taggings. It is represented by a DAG.

The disambiguators make use of constraint-based, handwritten rules. If # is a sentence boundary symbol that allows the rules to recognize the right and left limits of the sentence, the purpose of a rule R is to specify the set $L(R) \subset \#T^\star\#$ of all taggings that satisfy a given constraint. Both disambiguators contain an algorithm for processing a sentence s with a rule R, i.e. for computing the subset $\#Init(s)\# \cap L(R)$ of initial taggings of s that satisfy R. This set is also represented by a DAG. The silence rate is zero if every correct tagging of a correct sentence is in $L(R)$, i.e. if $L(R)$ is a superset of the language.

In Roche's and Silberztein's original proposals, rules are not sensitive to sentence boundaries, and $L(R) \subset T^\star$. The symbol # is the only new feature that we introduce. The computational consequences of this addition are minor, and most of the original algorithms can still be used for the modified version. However, the expressive power is extended.

In our variant of the modified negative disambiguator, handwritten rules take the form of finite-state automata that specify sets of forbidden sequences of elements of $T \mid \#$. If R_1 is such a rule, the set $L_1(R_1)$ of all taggings that do not contain any sequence of R_1 is:

$$\#T^{\star}\# \setminus (T \mid \#)^{\star} R_1 (T \mid \#)^{\star}.$$

The disambiguator builds a DAG for $\#Init(s)\# \cap L_1(R_1)$ from the DAG for $\#Init(s)\#$ and from the automaton for R_1 (Roche 1992). A particularly efficient algorithm for that is described in Mohri 1994; it can still be used in the modified version.

We now describe our variant of the modified imperative disambiguator. In order to accurately describe how it operates, we need a few more definitions[1]. In addition to T, this disambiguator uses "incomplete tags", which specify either a grammatical category or a lemma, with or without a relevant inflectional pattern code, with or without one or several relevant inflectional features (whereas a complete <word, tag> pair in T specifies all relevant grammatical information for a given form). An additional incomplete tag, <WORD>, matches with every simple word form (cf. Silberztein 1993, for details). Let T' be the set of all (complete or incomplete) tags. For each tag $a \in T'$, let $\sigma(a) \subset T$ be the set of complete tags that match with a. If $a \in T$, $\sigma(a) = a$. In addition, $\# \in T'$ and $\sigma(\#) = \#$. Handwritten rules take the form of finite-state transducers whose transitions are labelled by elements of T' and which realize rational transductions in $\mathrm{Rat}(T'^{\star} \times T'^{\star})$. Let R_2 be such a transduction. The set $\mathrm{dom}R_2$, the set of possible input sequences of R_2, constitutes the condition part of the rule. The set $\mathrm{im}R_2$, the set of possible output sequences of R_2, constitutes the action part of the rule. The transduction must preserve length and have the property that $(\varepsilon, \varepsilon) \notin R_2$, where ε is the empty word.

If W is the set of simple word forms in the language, a sentence s is an element of $\#W^{\star}\#$. For each $a \in T$, let $\varphi(a) \in W^{+}$ be the word form described by a. If a describes a compound form, $\varphi(a)$ is a sequence of several simple forms. In addition, let $\varphi(\#) = \#$. The mapping φ can be extended to sequences of elements of $T \mid \#$: for instance, if $s \in W^{\star}$, $\varphi^{-1}(s) = Init(s)$. Now let ρ be the equivalence relation over $(T \mid \#)^{\star}$ defined by:

$$\forall u, v \in (T \mid \#)^{\star} (u \rho v \Longleftrightarrow (|u| = |v| \text{ and } \forall i \in [1, |u|]\ \varphi(u_i) = \varphi(v_i)))$$

[1] We will make use of the mathematical terminology of Berstel 1979 with the exception that the empty word will be denoted by ε.

i.e. the taggings u and v describe the same piece of text and they delimit compounds in the same way. Next, if we also extend the mapping σ to sequences of elements of T', a transduction

$$g(R_2) \in \mathrm{Rat}((T \mid \#)^\star \times (T \mid \#)^\star)$$

is defined by:

$$\forall u, v \in (T \mid \#)^\star$$
$$((u, v) \in g(R_2) \iff \exists (u', v') \in R_2 \ (u \in \sigma(u') \text{ and } v \in \sigma(v') \text{ and } u\rho v))$$

For each sequence $s \in (W \mid \#)^\star$, a transduction

$$f(s, R_2) \in \mathrm{Rat}((T \mid \#)^\star \times (T \mid \#)^\star)$$

is defined by:

$$\forall u, v \in (T \mid \#)^\star$$
$$((u, v) \in f(s, R_2) \iff (u, v \in \mathrm{Pref}(\varphi^{-1}(s)) \text{ and } (u, v) \in g(R_2))$$

where $\mathrm{Pref}(\varphi^{-1}(s))$ is the set of prefixes of sequences of $\varphi^{-1}(s)$. For each $s \in (W \mid \#)^\star$, a set $h(s, R_2) \subset (T \mid \#)^\star$ is defined by:

$$\begin{array}{ll} \text{if } \sigma(\mathrm{dom}R_2) \cap \mathrm{Pref}(\varphi^{-1}(s)) = \emptyset, & \text{then } h(s, R_2) = T \cap \mathrm{Pref}(\varphi^{-1}(s)), \\ & \text{else } h(s, R_2) = \mathrm{im}f(s, R_2). \end{array}$$

Then, $k(R_2) \subset (T \mid \#)^\star$ is defined by defining $k(R_2) \cap \varphi^{-1}(s)$ for each $s \in (W \mid \#)^\star$ by induction on $|s|$. If $|s| = 0$, $k(R_2) \cap \varphi^{-1}(s) = \varepsilon$; if $|s| > 0$,

$$\begin{aligned} & k(R_2) \cap \varphi^{-1}(s) \\ & = \textstyle\bigcup_{u \in h(s, R_2)} u.(k(R_2) \cap Init(\varphi(u)^{-1}s)) \\ & = \{uv \in (T \mid \#)^\star \mid \varphi(u)\varphi(v) = s \text{ and } u \in h(s, R_2) \text{ and } v \in k(R_2)\} \end{aligned}$$

Finally, $L_2(R_2) = k(R_2) \cap \#T^\star\#$. When a sentence s is being processed, the disambiguator constructs a DAG for $\#Init(s)\# \cap L_2(R_2)$ from the DAG for $\#Init(s)\#$ and from the transducer that realizes R_2. Silberztein's algorithm for that must undergo small modifications to be used in the modified version, but its time complexity is not changed.

3

Expressive power of the disambiguators

The two disambiguators share the same general principles and make use of the same tagset. The difference between them lies in the interpretation of the rules, which are negative in one system and imperative in the other. There is no self-evident general way of simulating the negative rules with the imperative ones or vice versa. Therefore it makes sense to compare their expressive power.

When a constraint-based disambiguator makes use of a rule R, the output of its operation on texts exclusively depends on the set $L(R)$ of all taggings that satisfy R. Thus the expressive power of a disambiguator is characterized by the class of all formal languages $L(R) \subset \#T^\star\#$ for all possible rules R. A given disambiguator is at least as powerful as another if the operation of the latter with any rule R_2 can be simulated by the operation of the former with a rule R_1, i.e. if for each rule R_2 one can generate a rule R_1 such that $L_1(R_1) = L_2(R_2)$. Actually, neither of the two original disambiguators is more powerful than the other (Laporte 1994). However, the introduction of sentence boundaries extends their power. Indeed, with $L(R) \subset T^\star$, they compute $Init(s) \cap L(R)$ from $Init(s)$: this operation is simulated by our variants by computing $\#Init(s)\# \cap \#L(R)\#$ from $\#Init(s)\#$.

As to our variant of the negative disambiguator, the class of formal languages characterizing its expressive power is exactly the class of regular languages, i.e. languages recognizable by finite-state computational devices.

First, for each rule R_1, $L_1(R_1) = \#T^\star\# \setminus (T \mid \#)^\star R_1 (T \mid \#)^\star$ is regular since R_1 is a regular language.

Conversely, for each regular language $L \subset \#T^\star\#$, there exists a rule R_1 such that $L_1(R_1) = L$. Indeed, take $R_1 = \#T^\star\# \setminus L$. Since

L is regular, R_1 is regular and can be used as a rule by the negative disambiguator. Then

$$\begin{array}{rcl} L_1(R_1) & = & L_1(\#T^\star\# \setminus L) \\ & = & \#T^\star\# \setminus (T \mid \#)^\star(\#T^\star\# \setminus L)(T \mid \#)^\star \\ & = & L \end{array}$$

since $L \subset \#T^\star\#$. Our variant of the negative disambiguator has thus the full power of finite-state computational devices.

The same holds for our variant of the imperative disambiguator.

First, we will show that for each R_2, $L_2(R_2)$ is a regular language. The proof uses a new symbol @ and the projection

$$\psi : (T \mid \# \mid @)^\star \longrightarrow (T \mid \#)^\star$$

such that $\psi(@) = \varepsilon$. Then the following holds:

$$\begin{array}{rcl} k(R_2) & = & \psi((@(T \mid \#) \mid \mathrm{img}(R_2))^\star \\ & & \setminus (T \mid \# \mid @)^\star @ \psi^{-1}(\varphi^{-1}(\varphi(\sigma(\mathrm{dom}R_2))(W \mid \#)^\star))) \end{array}$$

This equality follows from the definitions above. The proof is laborious but straightforward. It shows that $k(R_2)$, and therefore $L_2(R_2)$, is a regular language.

Conversely, let L be a regular subset of $\#T^\star\#$. There exists a finite set of incomplete tags $a_i, b_i \in T'$ such that $\forall i \in [i, n]\ \sigma(a_i) \cap \sigma(b_i) = \emptyset$ and $\bigcup_{1 \leq i \leq n} \sigma(a_i) = T$. Indeed, the incomplete tag for all simple word forms, <WORD>, can be combined with tags for grammatical categories that contain only compounds so as to satisfy these two assertions. Then we define R_2:

$$R_2 = \{(u, u) \mid u \in L\} \cup \{(\#a_i, \#b_i) \mid 1 \leq i \leq n\}$$

$g(R_2) = \{(u, u) \mid u \in L\}$ since $\forall u \in L\ \sigma(u) = u$. For $s \in \#W^+\#$,

$$f(s, R_2) = \{(u, u) \mid u \in L \cap \mathrm{Pref}(\varphi^{-1}(s))\}$$

$$\begin{array}{rcl} \sigma(\mathrm{dom}R_2) & = & \sigma(L \cup \{\#a_i \mid 1 \leq i \leq n\}) \\ & = & L \cup \# \bigcup_{1 \leq i \leq n} \sigma(a_i) \\ & = & L \cup \#T \end{array}$$

$$\sigma(\mathrm{dom}R_2) \cap \mathrm{Pref}(\varphi^{-1}(s)) \ \supset\ \#T \cap \mathrm{Pref}(\varphi^{-1}(s)) \neq \emptyset$$

$$h(s, R_2) = \mathrm{im}f(s, R_2) = L \cap \mathrm{Pref}(\varphi^{-1}(s)) = L \cap \varphi^{-1}(s)$$

since $\mathrm{Pref}(\varphi^{-1}(s)) \cap \#T^\star\# \subset \varphi^{-1}(s)$.

$$\begin{array}{l} k(R_2) \cap \varphi^{-1}(s) \\ = \{uv \in (T \mid \#)^\star \mid \varphi(u)\varphi(v) = s \text{ and } u \in L \cap \varphi^{-1}(s) \text{ and } v \in k(R_2)\} \\ = \{u \in (T \mid \#)^\star \mid \varphi(u) = s \text{ and } u \in L\} \\ = L \cap \varphi^{-1}(s) \end{array}$$

Thus for each sentence $s \in W^+$,

$$\begin{array}{rcl} L_2(R_2) \cap \#Init(s)\# & = & L_2(R_2) \cap \varphi^{-1}(\#s\#) \\ & = & k(R_2) \cap \varphi^{-1}(\#s\#) \\ & = & L \cap \varphi^{-1}(\#s\#) \\ & = & L \cap \#Init(s)\#. \end{array}$$

Finally $L_2(R_2) = L$.

Conclusion

Both disambiguators described above have the full expressive power of finite-state computational devices. In other words, they can compute the intersection of the sentence automaton with any regular language L. The same operation could be carried out directly with a finite-state automaton that recognizes L. The expressive power would be equivalent.

The computation would not necessarily be slower. In the systems described above, the regular language L is specified by means of a rule whose actual size is usually considerably smaller (a few dozens of states) than that of an automaton for the regular language (hundreds of thousands of states and transitions). However, the recognition of a word by a deterministic automaton depends on the length of the word and in principle not on the size of the automaton. Automata with millions of transitions have already been implemented in a computationally successful way (Roche 1994).

The advantage of the systems described in this paper is rather the readability of the rules. If L were recognized by an automaton of a reasonable size for a human reader, it would probably be interesting to examine or build it directly, one could see exactly the effect of the rules. But automata with more than a few dozens of states are definitely unreadable and cannot be written down by hand.

Therefore, though intersection with a regular language is a simple mathematical operation, lexical disambiguation through grammatical constraints implies the use of a rule formalism and a rule compiler. One can compare the expressive power of such rule formalisms independently of particular rules, and deduce from the comparison whether a formalism can theoretically be simulated by another.

Bibliography

Berstel, Jean. 1979. *Transductions and Context-Free Languages.* Stuttgart: Teubner. 278 p.

Charniak, Eugene, Curtis Hendrickson, Neil Jacobson, and Mike Perkowitz. 1993. Equations for Part-of-Speech Tagging. In *AAAI '93, Ninth National Conference on Artificial Intelligence.*

Church, Kenneth W. 1988. A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. In *Proceedings of the Second Conference on Applied Natural Language Processing, Austin, Texas*, 136–143. Morristown, New Jersey. Association for Computational Linguistics.

Koskenniemi, Kimmo. 1990. Finite-State Parsing and Disambiguation. In *Proceedings of COLING '90*, ed. H. Karlgren, 229–232. Helsinki University.

Kupiec, J.M. 1992. Robust Part-of-Speech Tagging Using a Hidden Markov Model. *Computer, Speech and Language* 6:225–242.

Laporte, Éric. 1994. Experiments in Lexical Disambiguation Using Local Grammars. In *Papers in computational lexicography. COMPLEX '94*, ed. Ferenc Kiefer, Gábor Kiss, and Júlia Pajzs, 163–172. Budapest. Research Institute for Linguistics, Hungarian Academy of Sciences.

Milne, Robert. 1986. Resolving Lexical Ambiguity in a Deterministic Parser. *Computational Linguistics* 12(1):1–12.

Mohri, Mehryar. 1994. Syntactic Analysis by Local Grammars Automata: an Efficient Algorithm. In *Papers in computational lexicography. COMPLEX '94*, ed. Ferenc Kiefer, Gábor Kiss, and Júlia Pajzs, 179–191. Budapest. Research Institute for Linguistics, Hungarian Academy of Sciences.

Paulussen, Hans, and Willy Martin. 1992. DILEMMA-2: a Lemmatizer-Tagger for Medical Abstracts. In *3rd Applied ACL*, 141–146. Trento, Italy.

Rimon, Mori, and Jacky Herz. 1991. The Recognition Capacity of Local Syntactic Constraints. In *5th Conference of the European Chapter of the ACL. Proceedings of the Conference*, 155–160. Berlin.

Roche, Emmanuel. 1992. Text Disambiguation by Finite-State Automata, An Algorithm and Experiments on Corpora. In *COLING '92. Proceedings of* *the Conference.* Nantes, France.

Roche, Emmanuel. 1994. Two Parsing Algorithms by Means of Finite-State Transducers. In *COLING '94. Proceedings of the Conference.* Kyoto.

Silberztein, Max. 1989. *Dictionnaires électroniques et reconnaissance lexicale automatique.* Doctoral dissertation, University of Paris 7. 176 p.

Silberztein, Max. 1993. *Dictionnaires électroniques et analyse automatique de textes. Le système INTEX.* Informatique linguistique. Paris: Masson. 233 p.

Silberztein, Max. 1994. INTEX: A Corpus Processing System. In *COLING '94. Proceedings of the Conference.* Kyoto.

Tapanainen, Pasi, and Atro Voutilainen. 1993. Ambiguity Resolution in a Reductionistic Parser. In *Conference of the European Chapter of the ACL. Proceedings of the Conference*, 394–403. Utrecht.

Weischedel, Ralph, Marie Metter, Richard Schwartz, Lance Ramshaw, and Jeff Palmucci. 1993. Coping with Ambiguity and Unknown Words through Probabilistic Models. *Computational Linguistics* 19(2):359–382.