Optimizing Rule-Based Morphosyntactic Analysis of Richly Inflected Languages — a Polish Example

Dominika Pawlik
University of Warsaw
Institute of Mathematics

Aleksander Zabłocki
University of Warsaw
Institute of Informatics

Bartosz Zaborowski
Polish Academy of Sciences
Institute of Computer Science

{dominika, olekz}@mimuw.edu.pl, b.zaborowski@ipipan.waw.pl

Abstract

We consider finite-state optimization of morphosyntactic analysis of richly and ambiguously annotated corpora. We propose a general algorithm which, despite being surprisingly simple, proved to be effective in several applications for rulesets which do not match frequently.

1 Introduction

Morphosyntactic analysis of natural language texts is commonly performed as an iterative process of applying pre-defined syntactical rules to a previously tokenised and morphologically annotated input (Aït-Mokhtar and Chanod, 1997). We consider two of its sub-tasks: shallow parsing and disambiguation.

As stated in (Mohri, 1997), such tasks can often be efficiently realized with finite-state transducers (FST), which allow time savings by the classical operations of determinization, minimization and composition (Roche and Schabes, 1995). However, the applicability of the FST model may depend on the richness of annotation and on the expressive power of the rules. These both tend to complicate in richly inflected languages, including Baltic and most of Slavic.

Motivated by the needs which arose during the development of the National Corpus of Polish (NCP, (Przepiórkowski et al., 2012)), we aim at a high-efficiency rule-based morphosyntactic analysis framework suitable for such languages. Despite the existence of many formal methods and tools for this task, we are not aware of any previous result meeting all our needs, listed in Section 2. We discuss the state of the art in Section 3.

Our solution to the problem, though surprisingly single, performs well under the assumption that the rules match *rarely*, i.e. the average number of matches per rule per sentence is significantly below 1. In the case of rules designed for analyzing the Polish corpora (Przepiórkowski, 2008a), this was about 0.05–0.1, depending on the particular corpus and ruleset.

2 Problem statement

Richly inflected languages often require complex annotation as in the Constraint Grammar model (Karlsson, 1990): each token is assigned a list of potentially correct *readings*, each of which specifies the lemmatized form and a *tag* consisting of the values of syntactic attributes (PoS, case, gender etc.). For example, the English word *found* could have the following readings:[1]

found:VB	(The king would *found* a new city.)
find:VBD	(He *found* no money for that.)
find:VBN	(The robber has not been *found*.)

while the Polish word *drogi* the following ones:

drogi:adj:m3:sg:nom	/It is an *expensive* house./
drogi:adj:m3:sg:acc	/I see an *expensive* house./
drogi:adj:m2:sg:nom	/It is an *expensive* dog./
droga:noun:f:sg:gen	/I do not see that *road*./
droga:noun:f:pl:acc	/I see these *roads*./
... (6 other readings[2])	

In this setting, *unification* becomes a key tool in morphosyntactic analysis. For example, since a noun must agree with its modifying adjectives upon case, number and gender, unifying the values of these attributes can be used to remove all but the first two readings of *drogi* in

drogi	*dom*
drogi:adj:m3:sg:nom	dom:noun:m3:sg:nom
drogi:adj:m3:sg:acc	dom:noun:m3:sg:acc
drogi:adj:m2:sg:nom	

[1] In the examples for English language, we use the PoS tags of the Brown corpus. Tagging for Polish roughly follows the National Corpus of Polish, though has been simplified.

[2] This is the number of practically possible readings (with disregard of the context). Note that in general, the complex morphology of Polish forces the morphological analyzers to either under- or over-generate token readings. While the latter choice is often considered as better for the accuracy, it makes the syntactic analysis even more complicated.

Proceedings of the 11th International Conference on Finite State Methods and Natural Language Processing, pages 44–48,
St Andrews–Sctotland, July 15–17, 2013.

... (8 other readings)

/an expensive house/

Hence, we desire at least the following features:

1. Functionality. Our rules should support disambiguation by unifying selected attributes in (selected of) the tokens matched by a regexp, e.g.

[pos~vb] A:([pos~adj]*) B:[pos~noun]
⇒ unify(case number gender, A B)

should look for a sequence consisting of a verb[3], a number of adjectives and a noun, and unify three attributes of the two latter. As for shallow parsing, we should allow creating syntactic structures depending on whether such unification has been successful (e.g. a noun and an adjective can be marked as a nominal group if they agree).

2. (Practical) determinism. By this we mean that single rules should run in an (almost) linear time in the input size. This clearly holds in models relying on FST determinization[4], but we would equally appreciate e.g. deterministic pushdown automata or other not purely finite-state tools.

3. (Practical) composition. By this we mean obtaining the ability to execute a cascade of rules significantly faster than it would take to execute them separately.

Clearly, a trade-off between time and memory efficiency is involved in both conditions 2 and 3.

3 Related work

Building FSTs for given replace rules, their determinization and minimization have been extensively described in theory (see (Roche and Schabes, 1995) and (Mohri, 1997)) and efficiently implemented, e.g. in the XFST toolkit (Beesley and Karttunen, 2003). However, as many authors notice, these methods turn out to be inapplicable in some situations. In our case, this happens for more than one reason.

First, a FST may be not determinizable; as discussed in (Roche and Schabes, 1996, Sec. 3.8) and (van Noord and Gerdemann, 2001), this tends to happen for replace rules acting over unbounded ranges of tokens, including both rule types discussed in Section 2. Trying to by-pass this problem — by assuming that a rule will not have a match longer than n tokens — typically leads to an exponential number of states wrt. n, which practically makes composing the results impossible. As explained in (van Noord and Gerdemann, 2001, Sec. 3.6), the explosion of states can be avoided by equipping FSTs with a queue; however, such devices no longer admit the standard composition (and no other composition method is given).

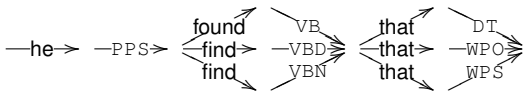
Apparently, the most commonly chosen solution of the problem is to abandon determinism for the rules which turn out to be too complex. This is the case in FASTUS (Hobbs et al., 1997), JAPE (Cunningham et al., 2000) (at least, in its full expressive power) and most probably also in SProUT (Becker et al., 2002), as its authors do not at all discuss the issue, while they refer to the standard methods which we discussed above.

In our situation, the second obstruction to using FSTs is a (yet another) explosion of states caused by unification rules for rich inflection. In order to compute the result of a unification, a determinized FST must remember its temporary result (in the current state) while processing consecutive tokens. This means increasing a part of the state space by a factor of N, where N is the number of possible unification results; in the case of Polish case-number-gender agreement, this is about 100.[5]

The scale of this problem depends on the representation of the input. Choosing a naive string encoding of the form

found/found:VB/find:VBD/find:VBN# (∗)

will increase N hopelessly since the result of unification will be a *set* of tags rather than a single value. This is not the case in the more common model for ambiguous annotation, Finite-State Intersection Grammar (FSIG) (Koskenniemi, 1990). In this approach, the input is represented as a finite-state automaton (FSA) so that alternative readings give rise to parallel paths, e.g. *he found that* will be represented as



The rules are applied by intersecting the above automaton with FSTs representing them. This means that the FSTs operate on the *paths* of the

[3] To be more precise, a *potential* verb, i.e. a token having some verbal reading. The same applies in the sequel.

[4] As in (Roche and Schabes, 1995), by FST *determinization* we mean building an equivalent subsequential form, i.e. a form not which does not backtrack during the execution.

[5] The number of possible values of these three attributes is $7 \cdot 2 \cdot 5 = 70$ in NCP but more than 200 in some other tagsets used for Polish.

input FSA, each of which contains exactly one reading of each token, e.g.

he PPS find VBN that WPO

In this setting, unification leads to at most N-fold increase of the state space even in compound FSTs, which allows hoping for composition. However, the FSIG model has the disadvantage that the execution time is not linear in terms of the input size, particularly for high ambiguity (cf. (Tapanainen, 1999)). Hence, the model does not ensure our efficiency requirements.

Efficient unification seems to be achievable by equipping the FST with additional memory for the temporary result, even though this means leaving the pure finite-state formalism. Note also that this is analogous to introducing a queue in (van Noord and Gerdemann, 2001). These observations have motivated the solution proposed below.

4 The solution

Basic model. We follow the idea present in the old Spejd as well as in JAPE: each rule is split into a simple *match pattern* and a list of more complex *actions* performed outside of the finite-state formalism. In our case, match patterns are regular expressions[6], while the actions can be virtually arbitrary, provided that they operate within a given match. The match patterns are compiled into deterministic automata (DFA). Rather than following the FSIG approach, we traditionally run these DFAs on single strings encoding the input, as in the example ($*$) on page 2, which enables finding matches without backtracking.

Whenever a match is found, we stop the DFA which found it and apply the pre-defined routines corresponding to the actions. For the example input *drogi dom* discussed in Section 2, the routine for unify would scan and store all the readings of *drogi*, do the same for *dom*, intersect the two sets of readings and save the result.[7]

Single-pattern DFAs. Even though our patterns are regular expressions, compiling them to DFAs is not straightforward since the actions may refer to *sub-matches* (e.g. A, B in the example in Section 2). In general, a single run of a classical DFA cannot unambiguously determine the positions of all such sub-matches (Friedl, 2002). For this purpose, we use the enhanced *tagged DFAs* (TDFAs) from (Laurikari, 2000) which use additional integer registers (which Laurikari calls *tags*) to store the references to potential sub-match boundaries.

TDFAs are asymptotically fast as they do not backtrack, but several times slower than the classical DFAs. Hence we adopt the *double pass* technique of (Cox, 2010): we process the input first with a classical DFA deciding solely whether it contains a match for a given rule; only when it does, we execute a more complex TDFA localizing the match and the desired sub-matches, and finally apply the rule actions. In this way, every sentence containing a match is processed twice; however, our assumption that the matches are (averagely) rare guarantees that in total this variant is faster than using only TDFAs.

Practical composition. We have already met the first two requirements from Section 2. It remains to provide means for practical composition. At this point, the (average) rareness of the matches becomes crucial. Intuitively, it means that many rules do not change the input, and it should be advantageous to compose *these* rules together. Among several possible realizations of this idea, we have chosen the following simple algorithm.

Let $M_1, \ldots, M_n$ denote the match specifications of the rules to be applied. We build:

- for each i, a TDFA T_i recognizing M_i;
- a DFA D recognizing $M_1|M_2|\ldots|M_n$.

Then, we slightly modify D in Laurikari's spirit by equipping it with two integer registers, L and R, used as follows: whenever a match for some M_i is found, D shall set $R := i$ provided that $L < i \leq R$. Hence, running D on the whole input with initial $R = \infty$ results in setting R to the least $i > L$ such that M_i has some match.

The main algorithm proceeds as follows:

1. Set $L := 0$. (Start with all rules.)
2. Set $R := \infty$ and run D on the whole input.
3. If $R = \infty$ (no M_i with $i > L$ matched), halt.
4. Run T_R on the whole input. For every match found, apply to it the actions of the R-th rule.
5. Set $L := R$ and jump to step 2.
   (M_L done; proceed only with $\{M_i : i > L\}$.)

[6] A match pattern is a regular expression over token specifications. These are built of attribute requirements, combined by logical connectives and quantifiers over readings, e.g. "there is a reading in which both pos equals noun and case equals nom". For the input encoding of the form ($*$), such patterns clearly translate to character-level regular expressions.

[7] This might be non-linear in terms of the match size but will happen rarely due to the rareness of the matches.

Avoiding explosion. Even though the number of constructed states is clearly smaller than in the FST models (as we have avoided states arising from sophisticated rule actions), it is still very high. We reduced it significantly[8] by a simple technique of *lazy construction*: the transitions and their target states are built only during the execution, just before their first use.

In practice, for a large number of rules, we fix a *composition width* k and apply the above algorithm to groups of k consecutive rules. To achieve best performance, k should be chosen possibly large but so that the states still fit into memory. This makes the running time asymptotically linear in the number of rules; nevertheless, the third requirement from Section 2 has been met.

5 Evaluation

Our approach has been applied to optimize Spejd, a syntactic parser of broad functionality which has been used for corpus development, valence extraction (Przepiórkowski, 2008b), and also sentiment analysis (Buczyński and Wawer, 2008).

composition width	**avg. speed** [tokens/s]	**memory** [MB]
N/A (old Spejd)	802	184
1	6,013	292
10	18,536	488
30	21,746	3,245

Table 1: The speed and memory usage of Spejd in terms of the composition width.

Table 1 presents the overall efficiency of the system. In the tests, performed on a computer with 3.1 GHz AMD FX processor, 467 handwritten rules of (Przepiórkowski, 2008a) were applied to a 15 million token sample from the IPI Corpus (Przepiórkowski, 2004).

Notably, the use of lazy construction increased the achievable width from about 3–4 to about 30.[9]

Conclusion

We have described a variation of the finite-state approach which, although it may seem strange in comparison with the classical solutions, seems to be a good choice for infrequently matching complex morphosyntactic rules, particularly for highly ambiguous rich annotation. In practice, it has allowed over 25-fold acceleration of the Spejd system while preserving its rich functionality.

[8] In some TDFAs in our applications, less than 5% of states are reachable in practice, even for megabytes of input.

[9] Further state space reduction techniques, which are out of scope of this paper, have allowed us to work with widths exceeding 1000. Due to the overhead involved in them, this resulted "only" in about 10-fold acceleration in comparison with $k = 30$.

References

Salah Aït-Mokhtar and Jean-Pierre Chanod. 1997. Incremental finite-state parsing. In *ANLP*, pages 72–79.

Markus Becker, Witold Drożdżyński, Hans-Ulrich Krieger, Jakub Piskorski, Ulrich Schäfer, and Feiyu Xu. 2002. SProUT — shallow processing with typed feature structures and unification. In *Proceedings of the International Conference on NLP (ICON 2002)*, Mumbai, India.

Kenneth R. Beesley and Lauri Karttunen. 2003. *Finite State Morphology*. CSLI Studies in Computational Linguistics. CSLI Publications.

Aleksander Buczyński and Aleksander Wawer. 2008. Shallow parsing in sentiment analysis of product reviews. In Sandra Kübler, Jakub Piskorski, and Adam Przepiórkowski, editors, *Proceedings of the LREC 2008 Workshop on Partial Parsing*, pages 14–18, Marrakech. ELRA.

Russ Cox. 2010. Regular expression matching in the wild. `http://swtch.com/~rsc/regexp/regexp3.html`.

Hamish Cunningham, Diana Maynard, and Valentin Tablan. 2000. JAPE: a java annotation patterns engine (second edition). Technical Report Technical Report CS–00–10, of Sheffield, Department of Computer Science.

Jeffrey E. F. Friedl. 2002. *Mastering Regular Expressions*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2 edition.

Jerry R. Hobbs, Douglas E. Appelt, John Bear, David J. Israel, Megumi Kameyama, Mark E. Stickel, and Mabry Tyson. 1997. FASTUS: A cascaded finite-state transducer for extracting information from natural-language text. *CoRR*, cmp-lg/9705013.

Fred Karlsson. 1990. Constraint grammar as a framework for parsing running text. In *Proceedings of the 13th conference on Computational linguistics - Volume 3*, pages 168–173, Stroudsburg, PA, USA. Association for Computational Linguistics.

Kimmo Koskenniemi. 1990. Finite-state parsing and disambiguation. In *Proceedings of the 13th conference on Computational linguistics - Volume 2*, pages 229–232, Stroudsburg, PA, USA. Association for Computational Linguistics.

Ville Laurikari. 2000. NFAs with tagged transitions, their conversion to deterministic automata and application to regular expressions. In *In Proceedings of the 7th International Symposium on String Processing and Information Retrieval*, pages 181–187.

Mehryar Mohri. 1997. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311.

Gertjan van Noord and Dale Gerdemann. 2001. Finite state transducers with predicates and identities. *Grammars*, 4(3):263–286.

Adam Przepiórkowski, Mirosław Bańko, Rafał L. Górski, and Barbara Lewandowska-Tomaszczyk, editors. 2012. *Narodowy Korpus Języka Polskiego [Eng.: National Corpus of Polish]*. Wydawnictwo Naukowe PWN, Warsaw.

Adam Przepiórkowski. 2004. *The IPI PAN Corpus: Preliminary version*. Institute of Computer Science, Polish Academy of Sciences, Warsaw.

Adam Przepiórkowski. 2008a. *Powierzchniowe przetwarzanie języka polskiego*. Akademicka Oficyna Wydawnicza EXIT, Warsaw.

Adam Przepiórkowski. 2008b. Towards a partial grammar of Polish for valence extraction. In *Grammar & Corpora 2007: Selected contributions from the conference Grammar and Corpora, Sept. 25–27, 2007, Liblice*, pages 127–133. Academia, Prague.

Emmanuel Roche and Yves Schabes. 1995. Deterministic part-of-speech tagging with finite-state transducers. *Comput. Linguist.*, 21:227–253, June.

Emmanuel Roche and Yves Schabes. 1996. Introduction to finite-state devices in natural language processing. Technical report, Mitsubishi Electric Research Laboratories.

Pasi Tapanainen. 1999. *Parsing in Two Frameworks: Finite-state and Functional Dependency Grammar*. University of Helsinki, Department of General Linguistics.