Competing Patterns for Language Engineering
Methods to Handle and Store Empirical Data*

Petr Sojka

NLP Laboratory, Faculty of Informatics, Masaryk University in Brno
Botanická 68a, 602 00 Brno, Czech Republic
E-mail: sojka@informatics.muni.cz

Abstract. In this paper we describe a method of effective handling of linguistic data by means of *covering and inhibiting patterns* – patterns that "compete" each other. A methodology of developing such patterns is outlined. Applications in the areas of morphology, hyphenation and part-of-speech tagging are shown. This pattern-driven approach to language engineering allows the combination of linguist expertise with the data learned from corpora – layering of knowledge. Searching for information in pattern database (dictionary problem) is blindingly fast – linear with respect to the length of searching word as with other finite-state approaches.

1 Introduction

There is a need to store empirical language data in almost all areas on natural language engineering (LE). Finite-state methods [21,13,16,17,10] have found their revival in the last decade. The theory of finite-state automata (FSA) and transducers (FST) is a well developed part of theoretical computer science (for an overview, see e.g. [6,2]). As the finite-state machines (FSM) needed tend to grow with increased demand for quality of language processing, more attention is being given to the efficiency of the handling of FSM [18,3]. The size of some FSM used in natural language processing exceeds ten millions states (e.g. weighted finite automata and transducers for speech recognition). Practical need to reduce the size of these data structures without losing their expressiveness and excellent time complexity of operations on them is driving new research activities – trend of experimental Computer Science is seen. Several FSM-based software tools [19,23,28,7] have already been implemented for LE.

In this paper we explore a method of FSM decomposition that allows a significant size reduction of FSM – the idea of storing empirical data using *competing patterns*. The data structure *trie* and pattern technique have been developed by Liang [14] for hyphenation algorithm in TEX [11, Appendix H]. We defined several kinds of patterns and our extensive experiments showed that the method is applicable in several areas of language engineering. *Bootstrapping* and *stratification* techniques allow us to speed up development of such machines – space savings and time of development savings are enormous.

* This research has been partially supported by the Czech Ministry of Education under the Grant VS97028 and by the Grant CEZ:J07/98:143300003.

Petr Sojka, Ivan Kopeček and Karel Pala (Eds.): TSD 2000, LNAI 1902, pp. 157–162, 2000.

This paper is organized as follows. In Section 2, we give basic definitions and short overview of known results. Section 3 discusses pattern development methodology. Section 4 describes in detail applications in the area of Czech morphology. An overview of results for hyphenation and compound word division is given in Section 5. Possible applications like part-of-speech (POS) tagging are described in Section 6.

2 Patterns

We start by formally introducing different kinds of patterns and basic notions (for a detailed discussion and examples, see e.g. [2,8]).

Definition 1 (pattern). *Let's have two disjoint alphabets* Σ *(the alphabet of* terminals*) and* V *(the alphabet of* variables*).* Patterns *are words over monoid* $\langle \Sigma \cup V, \cdot, \varepsilon \rangle$. *Patterns consisting of terminals only called* terminal patterns. *The* language $L(\alpha)$ defined by pattern α *consists of all words obtained from* α *by leaving the terminals unchanged and substituting a terminal word for each variable* v. *The substitution in our case has to be* uniform*: different occurences of* v *are replaced by the same terminal word. If the substitution replaces variables always by* nonempty *word, such language* L_{NE} *is* non-erasing*, and such pattern is called* NE-pattern. *Similarly, we define* erasing *language* L_E *as a language generated by* E-pattern *such that substitution of variable* v *by empty word* ε *is allowed.*

As an example of E-pattern may serve pattern *SVOMPT* for English sentences where the variables denote Subject, Verb, Object, Mood, Place, Time. An obvious useful task is to infer a pattern common to all input words in a given sample by the process of *inductive inference*. It has been shown in [8] that *inclusion problem* is undecidable for both erasing and non-erasing pattern languages. It is easy to show that decidability of *equivalence problem* for non-erasing languages is trivial. The decidability status of the equivalence problem for E-patterns remains open. These results show that trying to infer language description in the form of set of patterns (or the whole grammar) automatically is very hard task. It has been shown that decomposition of the problem by using *local grammars* [5] or building cascades of FSM [7] is a tractable, but very time-consuming task. Methods for the induction of patterns (from corpora) are needed.

Definition 2 (classifying pattern). *Let* $\langle A, \leq \rangle$ *be a partially ordered system,* $\leq$ *be a* lattice order *(every finite non-empty subset of* A *has lower and upper bound). Let* . *be a distinguished symbol in* $\Sigma' = \Sigma \cup \{.\}$ *that denotes the beginning and the end of word –* begin of word marker *and* end of word marker. Classifying patterns *are the words over* $\Sigma' \cup V \cup A$ *such that dot symbol is allowed at the beginning or end of patterns.*

Terminal patterns are "context-free", they apply anywhere in the classified word – dot symbol in a pattern specifies pattern at the beginning and end of word. Classifying patterns allow us to build *tagging hierarchies* on patterns.

Definition 3 (word classification, competing word patterns). *Let* P *be a set of patterns over* $\Sigma' \cup V \cup A$ *(*competing patterns, pattern base*). Let* $w = w_1 w_2 \ldots w_n$ *be a word to classify with* P. *Classification classify*$(w, P) = a_0 w_1 a_1 w_1 \ldots w_n a_n$ *of* w

```
    h y p h e n a t i o n
l1           1n a
l1               1t i o
l2            n2a t
l2                 2i o
l2        h e2n
l3 .h y3p h
l4        h e n a4
l5        h e n5a t
   .h0y3p0h0e2n5a4t2i0o0n.
    h y-p h e n-a t i o n
```

In this example $\langle A, \leq \rangle$ is $\mathbb{N}$ (natural numbers). There are 5 pattern levels – `l1...l5`. Patterns in odd levels are *covering*, in an even levels *inhibiting*. Winner pattern is `.h0y3p0h0e2n5a4t2i0o0n.` Pattern `h e n5a t` wins over `n2a t`, thus hyphenation is possible.

Fig. 1. Competing patterns and pattern levels

with respect to P is computed from pattern base P by competition. All patterns whose projection to Σ match substring of w are collected. a_i is supremum of all values between characters w_i and w_{i+1} in matched patterns. classify(w, P) is also called winning pattern.

An example of competing patterns is shown in Figure 1. Competing patterns extend the power of FST somewhat like adding the complement operator with respect to A. Ideally, instead of storing full FST, we make patterns that embody the same information in even more compact manner. Collecting patterns matching given word can be done in linear time, using trie data structure for pattern storage.

3 Methodology

Pattern Generation Size-optimised full-coverage pattern generation for a given word-list is an NP-complete task. However, there are several pattern generation strategies that allow the choice between size-optimal or coverage-optimal patterns [27] with `patgen` program [15]. A generation process can be parametrised by several parameters whose tuning strategies are beyond the scope of this paper; see [27,24] for details. Parameters could be tuned so that virtually all hyphenation points are covered, leading to about 99.9 % efficiency, and size is not far from optimum. Further investigation and research is necessary to find sufficient conditions for finding optimal results.

Stratification Technique As word-lists from which patterns are generated are rather big (5,000,000 for Czech morphology or hyphenation, even more for other tasks as POS tagging), they may be stratified. Stratification means that from 'equivalent' words only one or small number of representants are chosen for the pattern generation process.

Bootstrapping Technique Developing patterns is usually an iterative process. One starts with hand-written patterns, uses them on input word-list, sees the results, makes the correction, generates new patterns, etc. This technique proved to speed up pattern development process by the order of magnitude. We usually do not start from scratch, but use some previously collected knowledge (e.g. word-list).

4 Application to Czech Morphology

For the information extraction, information retrieval systems, indexing and similar tasks we need information on many kinds of sub-word divisions: dividing a word into atoms (cutting of prefixes, compound words recognition etc.) [12]. We have created several competing pattern sets using the word database of Czech morphological analyser `ajka` [22]. We have taken a word-list of 564974 Czech words (6.6 MB) with marked prefix segmentation and added 51816 similar ones (starting with the same letters, but morphologically different). We were able to build patterns that were able to perform prefix segmentation on 99.9% of words of our input word-list and 98% of words in our test set.

In comparison to naïve storage of word segmentation, there is several order of magnitude higher compression ratio. Even compared to storage of FSM using suffix compression in a trie, patterns compacted in trie data structure gives about tenfold of space reduction, still with linear access time.

5 Application to Hyphenation and Compound Words

We have used the pattern technique to cover Czech and German hyphenation points and compound word borders. From a Czech word-list (372562 words, approx. 4 MB), we were able to create 8239 patterns (40 KB) that cover 99.63% hyphenation points. From a German word-list (368152 words, 5 MB), we were able to create 4702 patterns (25.2 KB) that cover 98.37% hyphenation points.

To cover compound word hyphenation was more difficult, as longer patterns are needed. With slightly different parameters of pattern generation, we were able to create patterns for German compound words with 8825 patterns (70.2 KB) with 95.28% coverage. Higher coverage is at the expense of pattern size growth.

For details of hyphenation pattern generation for compound words in Czech and German using `patgen`, see [27,24,25].

6 Outline of an Application to Part-of-Speech Tagging

Two mainstream approaches are being used for the POS task: *linguistic*, based on hand-coded linguistic rules (constraint grammars) [9,20] and *machine learning* (statistical, transformation-based) approaches, based on learning the language model from corpora. Their combination is probably what is needed – the 'built-in' linguistic knowledge should be communicated to and take preference over e.g. statistical knowledge acquired during learning. We hope that ordered and competing patterns will be a viable unifying carrier of information that will allow combination of both approaches.

Finite-state cascaded methods have already been applied to the POS task [1]. Let us outline one possible approach. Given sentence $w_1\ w_2\ \ldots\ w_n$, an ambiguous tagger gives various possible tags: $p_{11}\ \ldots\ p_{1a_1}$ for the first word, $p_{21}\ \ldots\ p_{2a_2}$ for the second, etc. Writing output as $(p_{11}\ \ldots p_{1a_1})(p_{11}\ \ldots\ p_{2a_2})\ldots(p_{n1}\ \ldots\ p_{na_n})$, the task is to choose the right POS p_{ij} for every w_i. Taking tag set from Brown corpus (the Brown University Standard Corpus of Present-day American English) [4] for the sentence "The representative put chairs on the table.", we get the output

```
. AT  ( NN - JJ )    ( NN  VBD - ) ( NNS - VBZ ) IN AT  NN    .
  The representative put           chairs        on the table .
```

Hyphenation markers immediately after POS tag show good solutions for training. Such 'word-lists' (for each sentence from training corpus we get one 'hyphenated word') are used by `patgen` for disambiguation patterns generation. Sentence borders are explicitly coded. This or similar notation can be used for both formulation of ambiguous tagging decision strategies of variable context length by linguists. In comparison to classical constraint grammars, our experience shows that obligation to write only rules which are true in any context is very hard and only a few linguists are able to do so. Having pattern/rule levels/hierarchy helps to develop disambiguation strategies more easily and quickly. Generalisation for patterns over tree hierarchies is worked on.

7 Conclusion

We have shown effective methods for empirical language data storage and handling by means of competing patterns. Our *pattern-driven approach to language engineering* has been tested in several areas – hyphenation and morphology using prototype solution – programs `patgen` and TEX and their algorithms and data structures were used. Search in the pattern database is blindingly fast (linear with respect to the text length). Optimal pattern generation has non-polynomial time complexity, but sub-optimal solutions can be hand-tuned to meet the requirements. It remains to show that this approach is applicable and useful in areas as phonology, syllabification, speech segmentation, word sense or semantic disambiguation and speech processing. We believe that pattern-driven approach will be explored in NLP systems for various classification tasks soon.

Acknowledgement The author would like to thank reviewers for their suggestions to improve wording of the paper, and Radek Sedláček for providing a prefixed word-list for experiments described in Section 4.

References

1. Steven Paul Abney. Part-of-Speech Tagging and Partial Parsing. pages 118–136, Dordrecht, 1997. Kluwer Academic Publishers Group.
2. J. Richard Büchi. *Towards a Theory of Formal Expressions*. Springer-Verlag, New York, U.S.A, 1989.
3. Cezar Câmpeanu, Nicolae Sânteau, and Sheng Yu. Minimal cover-automata for finite languages. In Jean-Marc Champarnaud, Denis Maurel, and Djelloul Ziadi, editors, *Lecture Notes in Computer Science 1660*, pages 43–56, Berlin, Heidelberg, 1998. Springer-Verlag.
4. Nelson W. Francis and Henry Kučera. *Frequency Analysis of English Usage: Lexicon and Grammar*. Houghton Mifflin, 1982.
5. Maurice Gross. The Construction of Local Grammars. [21], pages 329–354.
6. Jozef Gruska. *Foundations of Computing*. International Thomson Computer Press, 1997.
7. Jerry R. Hobbs, Douglas Appelt, John Bear, David Israel, Megumi Kameyama, Mark Stickel, and Mabry Tyson. FASTUS: A Cascaded Finite-State Transducer for Extracting Information from Natural-Language Text. [21], pages 383–406.

8. Tao Jiang, Arto Salomaa, Kai Salomaa, and Sheng Yu. Decision problems for patterns. *Journal of Computer and Systems Sciences*, 50(1):53–63, 1995.
9. Fred Karlsson, A. Voutilainen, J. Heikkilä, and A. Antilla. *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text*. Mouton de Gruyter, Berlin, 1995.
10. Lauri Karttunen, Jean-Pierre Chanod, Gregory Grefenstette, and Anne Schiller. Regular Expressions for Language Engineering. *Natural Language Engineering*, 2(4):305–328, 1996.
11. Donald E. Knuth. *The TEXbook*, volume A of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986.
12. Gabriele Kodydek. A Word Analysis System for German Hyphenation, Full Text Search, and Spell Checking, with Regard to the Latest Reform of German Orthography. In Sojka et al. [26], pages 51–56.
13. András Kornai. *Extended Finite State Models of Language*. Cambridge University Press, 1999.
14. Franklin M. Liang. *Word Hy-phen-a-tion by Com-put-er*. Ph.D. Thesis, Department of Computer Science, Stanford University, August 1983.
15. Franklin M. Liang and Peter Breitenlohner. `PAT`tern `GEN`eration program for the TEX82 hyphenator. Electronic documentation of `PATGEN` program version 2.3 from web2c distribution on CTAN, 1999.
16. Mehryar Mohri. On some applications of finite-state automata theory to natural language processing. *Natural Language Engineering*, 2(1):61–80, 1996.
17. Mehryar Mohri. Finite-State Transducers in Language and Speech Processing. *Computational Linguistics*, 23(2):269–311, 1997.
18. Mehryar Mohri. Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 234:177–201, 2000.
19. Mehryar Mohri, Fernando C.N. Pereira, and Michael D. Riley. FSM Library – General-purpose finite-state machine software tools, 1998. `http://www.research.att.com/sw/tools/fsm/`.
20. Karel Oliva, Milena Hnátková, Vladimír Petkevič, and Paven Květoň. The Linguistic Basis of a Rule-Based Tagger of Czech. In Sojka et al. [26], pages 3–8.
21. Emmanuel Roche and Yves Schabes. *Finite-State Language Processing*. MIT Press, 1997.
22. Radek Sedláček. Morphological Analyzer of Czech (in Czech). Master's thesis, Faculty of Informatics, April 1999.
23. Max Silberztein. INTEX: an FST toolbox. *Theoretical Computer Science*, 234:33–46, 2000.
24. Petr Sojka. Notes on Compound Word Hyphenation in TEX. *TUGboat*, 16(3):290–297, 1995.
25. Petr Sojka. Hyphenation on Demand. *TUGboat*, 20(3):241–247, 1999.
26. Petr Sojka, Ivan Kopeček, and Karel Pala, editors. *Proceedings of the Third Workshop on Text, Speech and Dialogue — TSD 2000*, LNAI 1902, Brno, Czech Republic, Sep 2000. Springer-Verlag.
27. Petr Sojka and Pavel Ševeček. Hyphenation in TEX – Quo Vadis? *TUGboat*, 16(3):280–289, 1995.
28. Bruce W. Watson. Implementing and using finite automata toolkits. [13], pages 19–36.