# Segmentation from 97% to 100%
## Is It Time for Some Linguistics?

Petr Sojka

Faculty of Informatics, Masaryk University
Botanická 68a, 602 00 Brno, Czech Republic
sojka@fi.muni.cz

**Abstract.** Many tasks in natural language processing (NLP) require *segmentation* algorithms: segmentation of paragraph into sentences, segmentation of sentences into words is needed in languages like Chinese or Thai, segmentation of words into syllables (*hyphenation*) or into morphological parts (e.g. getting word stem for indexing), and many other tasks (e.g. tagging) could be formulated as segmentation problems. We evaluate methodology of using *competing patterns* for these tasks and decide on the complexity of creation of space-optimal (minimal) patterns that completely (100 %) implement the segmentation task. We formally define this task and prove that it is in the class of *non-polynomial* optimization problems. However, finding space-efficient competing patterns for real NLP tasks is feasible and gives efficient scalable solutions of segmentation task: segmentation is done in *constant* time with respect to the size of segmented dictionary. Constant time of access to segmentations makes competing patterns attractive data structure for many NLP tasks.

**Keywords:** competing patterns, segmentation, hyphenation, NP problems, pattern generation, patgen, context-sensitive patterns, machine learning, natural language engineering

> Everything is a symbol, and symbols can be combined to form *patterns*. Patterns are beautiful and revelatory of larger truths. These are the central ideas in the thinking of Kurt Gödel, M. C. Escher, and Johann Sebastian Bach, perhaps the three greatest minds of the past quarter-millennium. (Hofstadter [1])

## 1 Introduction

Many tasks in NLP require *segmentation* algorithms: segmentation of paragraph into sentences, segmentation of sentences into words is needed in languages like Chinese or Thai, segmentation of words into syllables (*hyphenation*) or into morphological parts (e.g. getting word stem for indexing), phoneme (speech) segmentation, and many other tasks (e.g. tagging) could be expressed as segmentation problems. Solution of segmentation task is an important brick in every NLP framework.

Aleš Horák, Pavel Rychlý (Eds.): Proceedings of Recent Advances in Slavonic Natural Language Processing, RASLAN 2012, pp. 121–131, 2012. 

As the available computing power steadily grows, new approaches recently deemed impossible are becoming reality – *empirical approaches* are used for machine learning of language phenomena: from huge language data (corpora, wordlists), language models and patterns are learnt by sophisticated algorithms through *machine learning* techniques. As examples of this shift, successful unsupervised learning of natural language morphology from language word lists has been reported in [2], and as overviewed in [3], supervised learning approaches are very successful for various types of segmentation. These merely statistical approaches work quite well for many tasks in the area of computational linguistics, and quickly reach above 90% efficiency in tasks such as part of speech tagging, sentence segmentation, speech recognition or probabilistic parsing. The main drawback of a solely statistical approach is that the results of learning methods are usually not understandable by expert linguists, as the language models are hidden in weights of synapses of neural nets or in zillions of probabilities or conditioned grammar rules. It appears that going the "last mile", increasing the remaining few percent purely statistically is not feasible, and ways to cover the remaining exceptions by usage of symbolic, linguistic descriptions are being sought [4].

Recognition of patterns is considered as the central issue in intelligence. Artificial intelligence needs *statistical emergence* [5]: for real semantics, symbols must be *decomposable,* complex, autonomous – active. A rule-based approach, such as, when the results of the learning process are human-understandable *rules* or *patterns*, allows for the merging of hand-crafted and machine learnt knowledge. It is becoming clear that a close cooperation between computer scientists and linguists is necessary [6] – both sides need each other. Neither rigorous computational analysis and formal models nor linguistic introspection and language models should be absent in successful approaches. First, symbolical descriptions should be sought, and only when not sufficient a dice should be drawn.

Patterns can be identified as a set of objects that share some common property. During the emergence of patterns covering the rules in data, some *exceptions* may occur. Remaining errors and exceptions covered in the first level can be viewed again as set of objects and described by *inhibiting patterns*. The next layer of *covering patterns* may describe the patterns in the data not handled by previous rules, and so on. By this process, knowledge from the data can be learnt, either by an automatic procedure, or by information fusion from different sources.

There is plethora of methods of machine learning, data mining and knowledge management. However, up to now, we are not aware of an systematic attempt made to deal with the large-scale exception handling that is so widespread across linguistic data in machine learning methods and data mining. This work is one of the first attempts to formalize and fully employ the theory of competing patterns for the utilization of language data in the areas of natural language processing and computer typesetting.

## 2 Basic Notions

The two fundamental problems are pattern definition and pattern recognition/generation from input data. There are many ways of formalizing patterns – sets of objects sharing some recognizable properties (attributes, structure, ...).

**Definition 1 (pattern).** *By alphabet we mean a finite, nonempty set. Let us have two disjoint alphabets* $\Sigma$ *(the alphabet of* terminals*, called also called* characters *or* literals*) and* $V$ *(the alphabet of* variables*).* Patterns *are words over the free monoid* $\langle(\Sigma \cup V)^*, \varepsilon, \cdot\rangle$*. The length* $|\varepsilon|$ *of an empty word* $\varepsilon$ *is zero. Patterns having only terminals are called* terminal patterns *or* literal patterns*. The length of a literal pattern* $p$*, denoted by* $|p|$*, is the number of literals in it. The* language $L(\alpha)$ defined by a pattern $\alpha$ *consists of all words obtained from* $\alpha$ *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 always replaces variables by a* nonempty *word, such language* $L_{NE}$ *is* non-erasing*, and such pattern is called* NE-pattern*. Similarly, we define an* erasing *language* $L_E$ *as a language generated by an* E-pattern *such that substitution of variable* $v$ *by empty word* $\varepsilon$ *is allowed.*

The pattern *SVOMPT* for English sentences where the variables denote Subject, Verb, Object, Mood, Place, Time may serve as an example of E-pattern. A 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 by Jiang et al. [7] that the *inclusion problem* is undecidable for both erasing and non-erasing pattern languages. It is easy to show that the decidability of the *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 a set of patterns (or the whole grammar) automatically is very difficult task.

We focus our attention in the further study to literal patterns only.

**Definition 2 (classifying pattern).** *Let* $A$ *be alphabet, 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 only at the beginning or end of patterns.*

Terminal patterns are "context-free" and apply anywhere in the classified word. It is important to distinguish patterns applicable at the beginning and end of word by the dot symbol in a pattern.[1] Classifying patterns allow us to build *tagging hierarchies* on patterns.

[1] It is itmrnopt to dgtusisinh ptatren apcbliplae at the bngninieg and end of wrod by the dot sobmyl in a ptarten.

**Definition 3 (word classification, competing word patterns).**
*Let $P$ be a set of patterns over $\Sigma' \cup V \cup A$ (*competing patterns, pattern set*). Let $w = w_1 w_2 \ldots w_n$ be a word to be classified with $P$. Classification classify$(w, P) = a_0 w_1 a_1 w_1 \ldots w_n a_n$ of $w$ with respect to $P$ is computed from a pattern set $P$ by a competing procedure: all patterns whose projection to $\Sigma$ match a 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 the* winning pattern.

It is worth noting that the classification procedure can be implemented very efficiently even for large pattern bases. Its effectiveness depends on the data structure where the patterns are stored. When an indexed trie is used, the classification of a word can be realized in linear time with respect to the word length $|w|$ and does not depend on $|P|$.

Our motivation for studying of competing patterns was the word division (hyphenation) problem. It is related to a dictionary problem – the problem of effective storage of a huge word list. An enumerated list of Czech words may have well above 6,000,000 words. Storage of such a large table even using hashing requires considerable space. Another idea is to use finite-state methods – finite-state automata and transducers. It has been shown that decomposition of the problem by using *local grammars* [8] or building cascades of finite state machines [9] is a tractable, even though very time-consuming task. The main problem with these approaches is that they do not generalize well – they do not perform well on unseen words. A structural decomposition of $W$ into patterns is the key idea here, and brings better generalization qualities:

**Definition 4 (word division problem).** *Let $W$ be a set of words over $\Sigma \cup \{0, 1\}$ such that placing 1 between two letters in $w$ denotes the possibility of word division at that point (placing 0 or nothing means the opposite). We want to find pattern set $P$ such that winning patterns classify$(w, P)$ encode the same information as $w$. In this case we say that $P$ or classify$(w, P)$* covers *$w$.*

We want to find a pattern set that is minimal in size and maximal in performance; we have to define these performance measures.

**Definition 5 (precision, recall, F-score).** *Let $W = (\Sigma \cup \{0, 1\})^*$, and $P$ a set of patterns over $\Sigma' \cup \mathbb{N}$. Let good$(w, P)$ is the number of word divisions where classify$(w, P)$ covers $w$, good$(W, P) = \sum_{w \in W}$ good$(w, P)$. bad$(w, P)$ is the number of word divisions where classify$(w, P)$ classifies word division that is not in $w$, bad$(W, P) = \sum_{w \in W}$ bad$(w, P)$. missed$(w, P)$ is the number of word divisions where classify$(w, P)$ fails to classify word division that is in $w$, missed$(W, P) = \sum_{w \in W}$ missed$(w, P)$. The definition of the measures is then as follows:*

$$\mathit{precision}(W, P) = \frac{\mathit{good}(W, P)}{\mathit{good}(W, P) + \mathit{bad}(W, P)} \tag{1}$$

$$\mathit{recall}(W, P) = \frac{\mathit{good}(W, P)}{\mathit{good}(W, P) + \mathit{missed}(W, P)} \tag{2}$$

The precision and recall scores can be combined into a single measure, known as the *F-score* [10]:

**Definition 6 (F-score).**

$$F(W,P) = \frac{2 \times precision(W,P) \times recall(W,P)}{precision(W,P) + recall(W,P)} \tag{3}$$

An F-score reaches its maximum when both precision and recall is maximal; in the case $F(W,P) = 1$ all information about word division is compressed into the pattern base $P$.

**Definition 7 (lossless compression, cross validation).** *If $F(W,P) = 1$ we say that we* losslessly compressed *$W$ into $P$. We can test performance of $P$ on an unseen word list $W'$ to measure the generalization properties of pattern set $P$ – in the machine learning community, the term* cross validation *is used.*

## 3 Generation of Minimal Patterns is Non-Polynomial Task

Here we see one advantage of the pattern approach. In the case where we have solved the hyphenation problem by storing all the words with the division point in a hash table or using a finite state transducer, we do not know how to segment new, unseen words. On the other hand, pattern $P$ trained on $W$ can perform well on unseen words (typically new long words or compounds) – as in patterns the *rules* are generalized.

There are many pattern sets $P$ that losslessly compress (cover) $W$; one straightforward solution is having just one pattern for every word $w \in W$ by putting dot symbol around the word with division points marked by 1. Such a pattern set $P$ is a *feasible solution*. But we want to obtain minimal pattern set. Minimality can be measured by the number of patterns, by the number of characters in patterns, or by the space the patterns occupy when stored in some data structure. Even if we take the simplest measure by counting the patterns, and try to find a minimal set of patterns that cover $W$, we will show how hard the task is. To formulate it more precisely, we need to define:

**Definition 8 (minimum set cover problem).** *An instance of set cover problem is finite set $X$ and a family $\mathcal{F}$ of subsets of $X$, such that $X = \bigcup_{S \in \mathcal{F}} S$. The problem is to find a set $C \subseteq \mathcal{F}$ of* minimal size *which covers $X$, i.e. $X = \bigcup_{S \in C} S$.*

The minimum set cover problem (MSCP) is known to be in the class of NPO problems (optimization problems analogical to NP decision problems), [11]. A variant of MSCP, in which the subsets have positive weights and the objective is to minimize the sum of the weights in a set cover, is also NPO. Weighted version of minimum set cover problem is approximable within $1 + ln|X|$ as shown by Chvátal [12].

**Theorem 1 (pattern minimization problems).** *Let $W$ be a set of words with one division only. Problem of finding minimal number of patterns $P$ that losslessly compress $W$ is equivalent to the (weighted) minimum set cover problem.*

*Proof.* We show that the problem reduces to the minimal set cover problem. For every subset $C \in W$ there exists at least one feasible solution $P_C$ such that $P_C$ covers $C$ and does not cover any word in $W \setminus C$, e.g., pattern set $\{.c. \mid c \in C\}$. Between all such feasible solutions we choose a canonical representative $P'_C$ – a set which is smallest by some measure (e.g., number of patterns, or number of characters in the pattern set). We now have a one to one correspondence between all pattern sets that cover exactly $C$ represented by $P'_C$ and $C$. Thus we showed that a pattern coverage minimization problem is equivalent to the weighted minimum set cover [12] in NPO class.

We have shown that even a pattern covering problem without competition is already NPO. When trying to cover $W$ by competing patterns, complicated interactions may arise – we need some approximation of the optimal solution.

Liang's main concern in the pattern covering problem was the size of the patterns stored in a packed trie (indexed trie with packing the different families of the trie into a single large array in computer memory. He discusses NP-completeness of finding a minimum size trie [13, page 25] by pointing to the problem transformation from graph coloring by Pfleger [14].

Competing patterns extend the power of finite state transducer somewhat like adding the "not" operator to regular expressions.

Methods for the induction of covering patterns from $W$ are needed.

Attempts to catch the regularities in empirical data ($W$ in our case) can be traced back to the 1960s, when Chytil and Hájek started to generate unary hypotheses on finite models using the GUHA method [15].

**Definition 9 (matrix representation of the data).** *Let us have $m \times n$ matrix $W = w_{ij}$ of data that describe $m$ objects with $n$ binary attributes $P_1, P_2, \ldots, P_n$ (unary predicates). Either $P_j$ or $\neg P_j$ holds.* Elementary conjunction *is a conjuction of literals $P_j$, $1 \leq j \leq n$, where every predicate appears once at most. Similarly,* Elementary disjunction *is a disjunction of literals $P_j$ with the same condition. We say that the object $i$ fulfills elementary conjunction $\Phi$ if the formula exactly describes the attributes in line $i$ of $W$. We say that $\Phi$ holds for $W$ if $\Phi$ holds for all objects (lines in $W$). We say that formula $\Phi$ is* $p$-truth *if $\Phi$ holds for at least $100p\%$ of objects, $p \in \mathbb{R}, 0 < p \leq 1$.*

We immediately see that we can represent our hyphenation problem by a matrix $W$: the attribute in column $j$, $P_j$ tells whether a word can be divided (true or 1) or not (false or 0).

GUHA method searches for such elementary conjunctions $A$ (*antecedents*) and elementary disjunctions $S$ (*succedents*) with no common predicates, such that implication $A \rightarrow S$ is $p$-truth; it searches for hypotheses with highest $p$ to detect dependencies in data. Observational language in this case is *propositional logic*. There are many general approaches using first-order predicate calculus or even higher formalisms [16], but these are not necessary for our task.

**Definition 10 ($p$-truth pattern $\alpha$).** *Let us have m hyphenated words represented in matrix W as in Definition 9 on the facing page. We say that pattern $\alpha$ is $p$-*truth pattern *if it covers at least* $100p\%$ *of applicable word segmentation points.*

The greedy approach for pattern search consists in collecting $p$-truth patterns with the highest $p$ of the shortest length. Short patterns give a high generalization and good minimalization of space for pattern storage. But during its generation some heuristics have to be used, as maximal coverage of covering patterns does not imply good performace in the succeeding phases of pattern generation (of inhibiting patterns). Further discussion on pattern preparation could be found in [13,17,3].

## 4 Methods of Competing Patterns Generation

The idea of competing patterns is taken from the method developed by Liang [13] for his English hyphenation algorithm. It has been shown by extensive studies [18,19,20] that the method scales well and that parameters of the pattern generator – PATGEN program [21] – could be fine-tuned so that virtually all hyphenation points are covered, leading to about 99.9% efficiency.

The methodology consists of several parts:

**stratification** – for repetitive pattern generation, it is practical to have a stratified word list with 'information bearing' samples only;

**bootstrapping** – input data (word list with marked hyphenation points) preparation;

**goal-driven threshold setting heuristics** – the quality of generated patterns depends on many parameters that have to be set in advance;

**data filtering by threshold setting heuristics** – we can filter out 'dangerous' data – data that are hard to learn for manual inspection.

### 4.1 Stratification

Word lists from which patterns are generated may be rather big. A full list of Czech word forms has about 6,000,000 entries when generated by the Czech morphological analyzers ajka or majka. It may be even more than that for other tasks with huge input data collections such as POS tagging, or Thai text segmentation [22]. Context necessary for ambiguity resolution is often repeated several times – a word list may be stratified. Stratification means that from 'equivalent' words only one or small number of representatives are chosen for the pattern generation process.

With the stratification procedure described in [19] we have downsampled 3,300,000 Czech word forms to a word list of 372,562 word forms (samples) for PATGEN input. The same approach was used also for Slovak.

Stratified sampling is less important when we insist on lossless compression, or when we have enough computing power for pattern generation.

### 4.2 Bootstrapping

The preparation of data for machine learning is often a time-consuming task and for extremely large data sets, a technique called *bootstrapping* is used. It was used for tagging the ORCHID corpus [22] and for tagging word divisions it is also usefull. The idea is to tag only small initial data set (word list), and generate patterns from this input. Then, these bootstrap patterns are used for the automatic tagging of a bigger input list, and checked before the next pattern generation phase.

Bootstrapping may bring errors especially with overlapping prefixes (ne-, nej-, po-, pod-). It is worth the trouble marking these points separately, e.g., with the help of a morphological analyzer.

### 4.3 Pattern Generation

Pattern generation processes are driven by several threshold parameters whose settings are essential for the quality and properties (precision and recall) of generated patterns. Our experience shows that parameter setting not only depends on the requested pattern behaviour but to a certain extent on the problem at hand. Parameter setting has to be tuned for every pattern generation project.

PATGEN runs at various levels. At every level, a new set of patterns is generated. It starts with short patterns (counting frequencies of substrings of a given length), generating longer ones in the next level as 'exceptions', and making 'exceptions of exceptions' in the next level, etc. With this model, we can learn exact dependencies between contexts of hyphenation points in words that are used in a much wider context than can standard (bi|tri)gram or other statistical methods taken into consideration – there are examples when the segmentation decision depends on the word segment that is six characters away.

There is no known algorithm that helps with setting of the parameters of the learning process. Liang's original patterns (`hyphen.tex`) that are in every TEX distribution as a default patterns for (American) English are very inefficient and have very low recall. They cover only 89.3% [13, page 37] – of very small word list (Webster's Pocket Dictionary) of 49,858 words. The threshold used in pattern generation were not tuned at all, and better choices can lead to smaller pattern size and higher (actually complete) coverage of hyphenation points in an input word list.

### 4.4 Pattern Parameter Setting Optimization

Our extensive experience shows that parameter setting is highly language dependent – it differs when generating patterns for Thai segmentation [22] for Czech and Slovak hyphenations [19] Scannel [23] reports that using this methodology he generated a new pattern for Irish that does not produce any hyphen points which are not in the database and miss just 10 out of 314,639

hyphen points. This is consistent with our findings that the methodology is usable as very effective lossless compression algorithm, and there is the power of competing patterns to cover *all* exceptions from data.

We may experiment with parameter setting so that generated patterns are *nearly* lossless. Words that were not covered in this phase are in some way different than the rest. This difference may well be right, but usually show an input data tagging error. We suggest manually checking this small set of words esspecially when developing and marking new word lists from scratch.

### 4.5 Layering of Disambiguation Patterns

There can be a different version of the input data (different variants of segmentation, tagging), with different patterns. As competing patterns are decomposable into layers, we can "plug-in" patterns developed by experts on the problem and merge or compare them with those generated. We can let the patterns "compete" – or adjust them so that, for example, expert knowledge takes preference over generated patterns, or we can take the expert patterns as initial set of patterns and generate the patterns to cover the rest of the input data. It has been shown [19] that hyphenation patterns were often done by hand, or by a combination of hand crafted and generated patterns. Having several layers of expert patterns, we can easily set up their priorities by changing the classification numbers in the patterns. This priority handling is necessary in most information fusion tasks.

## 5 Summary and Conclusions

In this paper, we have formally proved the hardness of creation of space-optimal competing patterns for segmentation tasks. Even though the theoretical result seems negative, in practical applications like hyphenation, we are still able to find space efficient patterns that are able to solve segmentation task in constant time, e.g. irrespectively of the number of segmented inputs.

Techniques like bootstrapping, stratification and parameter generation setting heuristics allows for efficient working with language data to be segmented – a work for language experts. Fixing errors in data then allows for better and smaller pattern sets that are close to the theoretical optimum.

This all opens new horizons on usage of competing patterns as a new compact data structure in many NLP tasks.

**Acknowledgements** This work has been partially supported by the European Union through its Competitiveness and Innovation Programme (Information and Communications Technologies Policy Support Programme, "Open access to scientific information", Grant Agreement No. 250503).

## References


1. Hofstadter, D.R.: Gödel, Escher, Bach: An Eternal Golden Braid. Basic Books (1979)
2. Goldsmith, J.: Unsupervised Learning of the Morphology of a Natural Language. Computational Linguistics **27**(2) (2001) 153–198
3. Sojka, P.: Competing Patterns in Language Engineering and Computer Typesetting. PhD thesis, Masaryk University, Brno (2005)
4. Manning, C.: Part-of-Speech Tagging from 97% to 100%: Is It Time for Some Linguistics? In Gelbukh, A., ed.: Computational Linguistics and Intelligent Text Processing, 12th International Conference CICLing 2011, Part 1, LNCS 6608, Springer (2011) 171–189
5. Hofstadter, D.R.: Artificial intelligence: Subcognition as computation. (1983)
6. Brill, E., Florian, R., Henderson, J.C., Mangu, L.: Beyond N-Gram: Can Linguistic Sophistication Improve Language Modeling? In: Proceedings of the ACL '98. (1998)
7. Jiang, T., Salomaa, A., Salomaa, K., Yu, S.: Decision problems for patterns. Journal of Computer and Systems Sciences **50**(1) (1995) 53–63
8. Gross, M.: The Construction of Local Grammars. [24] 329–354
9. Hobbs, J.R., Appelt, D., Bear, J., Israel, D., Kameyama, M., Stickel, M., Tyson, M.: FASTUS: A Cascaded Finite-State Transducer for Extracting Information from Natural-Language Text. [24] 383–406
10. Manning, C.D., Schütze, H.: Foundations of Statistical Natural Language Processing. MIT Press (1999)
11. Ausiello, G., Gambosi, G., Crescenzi, P., Kann, V.: Complexity and Approximation. Springer-Verlag (1999)
12. Chvátal, V.: A Greedy Heuristic for the Set Covering Problem. Mathematics of Operations Research **4** (1979) 233–235
13. Liang, F.M.: Word Hy-phen-a-tion by Com-put-er. PhD thesis, Department of Computer Science, Stanford University (1983)
14. Pfleger, C.P.: State Reduction in Incompletely Specified Finite-State Machines. IEEE Trans. Computers C **22**(4) (1973) 1099–1102
15. Hájek, P., Havránek, T.: Mechanising hypothesis formation – Mathematical foundations for a general theory. Springer-Verlag (1978)
16. Lloyd, J.W.: Learning Comprehensible Theories from Structured Data. In Mendelson, S., Smola, A., eds.: Advanced Lectures on Machine Learning, LNAI 2600. (2003) 203–225
17. Sojka, P.: Competing Patterns for Language Engineering. In Sojka, P., Kopeček, I., Pala, K., eds.: Proceedings of the Third International Workshop on Text, Speech and Dialogue—TSD 2000. Lecture Notes in Artificial Intelligence LNCS/LNAI 1902, Brno, Czech Republic, Springer-Verlag (2000) 157–162
18. Sojka, P., Ševeček, P.: Hyphenation in TEX – Quo Vadis? TUGboat **16**(3) (1995) 280–289
19. Sojka, P.: Notes on Compound Word Hyphenation in TEX. TUGboat **16**(3) (1995) 290–297
20. Sojka, P.: Hyphenation on Demand. TUGboat **20**(3) (1999) 241–247
21. Liang, F.M., Breitenlohner, P.: `PAT`tern `GEN`eration program for the TEX82 hyphenator. Electronic documentation of `PATGEN` program version 2.3 from web2c distribution on CTAN (1999)
22. Sojka, P., Antoš, D.: Context Sensitive Pattern Based Segmentation: A Thai Challenge. In Hall, P., Rao, D.D., eds.: Proceedings of EACL 2003 Workshop on Computational Linguistics for South Asian Languages – Expanding Synergies with Europe, Budapest (2003) 65–72

23. Scannell, K.P.: Hyphenation patterns for minority languages. TUGboat **24**(2) (2003) 236–239
24. Roche, E., Schabes, Y.: Finite-State Language Processing. MIT Press (1997)