DRESDEN UNIVERSITY of TECHNOLOGY

DEPARTMENT of ELECTRICAL ENGINEERING and INFORMATION TECHNOLOGY

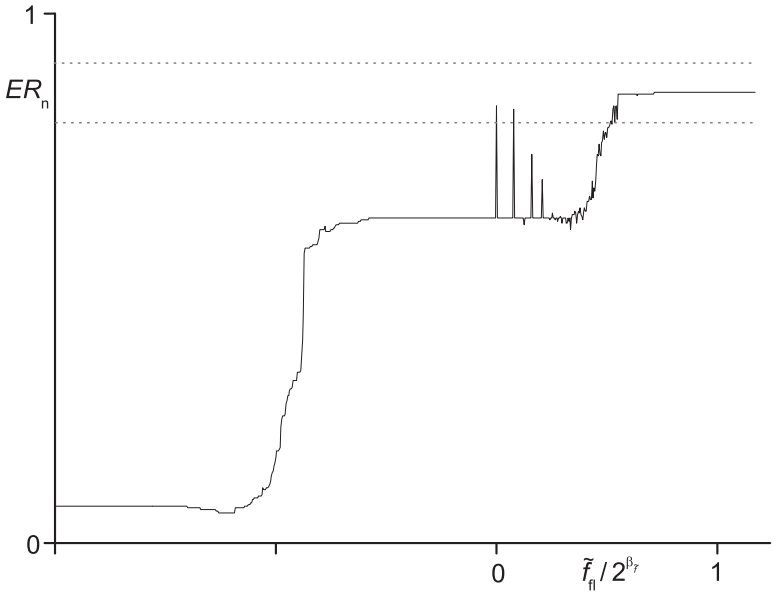
Laboratory of Acoustics and Speech Communication

Development of a Tagger for the Text-To-Speech System **Papageno**

Thesis of

David Sündermann

to attain the academical degree DIPLOMINGENIEUR (Dipl.-Ing.)



author:	David Sündermann
date and place of birth:	August the 8th 1977 in Magdeburg
care:	Dipl.-Ing. Horst-Udo Hain
responsible university teacher:	Prof. Dr.-Ing. habil. Rüdiger Hoffmann
date of handing in:	April the 2nd 2002

Acknowledgements

Above all I owe the fruitful contact with the SIEMENS CT IC 5 speech synthesis group to my teacher Professor Rüdiger Hoffmann. Kindly received among the motivated collaborators of the research department in Munich, Mr. Horst-Udo Hain, whose great engagement I want to emphasize, has stood by me in many matters. Also my colleagues Mrs. Petra Witschel and Mrs. Gabriele Bakenecker have assisted by means of their constructive suggestions and unrenounceable working up of training data. I want to apologize to my fiancée Miss Christiane Oeft for a lot of suffered privations on account of my frequent absence. Finally the self-sacrificing solicitude of my parents, Mrs. Sabine and Mr. Uwe Sündermann, during my work's terminal phase and the friendly review of the manuscript by Mr. Renko Geffarth are to be appreciated.

Dresden, March the 27$^{\text{th}}$ 2002 David Sündermann

Task Definition

Subject:
Development of a tagger for the text-to-speech system **Papageno**

Target:
A data-driven multilingual tagger for the SIEMENS TTS system **Papageno** is to be developed. Its input consists of possibly ambiguous word categories of any text. As output is expected a definite category for each word. Apart from a good quality the scarcity of memory and processing performance is to be considered. Dependent on application and language the category list must be definable without restrictions. To serve as training data hand-tagged texts and linguistic dictionaries are available.

Summary

After defining the notion *tagging* and giving an overall view of typical solution methods, an approach on the basis of n-grams is explained more closely. With regard to the demand for efficiency of computing time and memory the $\ltimes$-transformation is introduced to organize economically the extensive data quantity which comes into being while training. Also an extended version of the Viterbi Algorithm, disseminated in several fields of information technology, is realized by means of the $\ltimes$-transformation. Mathematically modeling statistic characteristics of the input material, the processing time behaviour is formally described. Besides a parallelization strategy is stated and its positive effect on the necessary calculation time is proven. With the help of examples the function of these algorithms is illustrated.
In the second part of this work **synther's** (the designation of **Papageno's** new tagging module) mode of operation and its integration into the TTS system are shown. After checking the fulfillment of further demands on taggers (robustness, adaptability, multilinguality) the attention is directed to **synther's** exactness resp. tagging errors by comparison with other taggers. Several improvement procedures are introduced and finally an outlook on future measures and tagging applications is given.

This Work Online

Authorized collaborators of the Dresden University of Technology resp. of SIEMENS CT IC 5 can have a look at an online version of this work, **synther's** source code and its manual in the internet:

`http://www.synther.de`

Please contact the author in order to receive the necessary password.

Contents

Chapter 1

Foundations

1.1 Why Tagging?

Similar to the body of mathematical rules, which makes possible intelligible, for scientists, engineers and a multitude of further users, unrenounceable and above all unmistakable expressions, natural languages dispose of a fundamental formulary that generally has come into being as result of several very ambitious men's efforts for purity and well-shape of language[1]. Putting a stop to the annoyed evolution of formally undescribed languages[2] already the elementary school confronts its pupils with an extensive body of rules and regulations. An instance is the determination of grammatical category, case, number, gender[3] etc. of words in the sentence context, technically said: their *tagging*.

The endeavors to utilize the meanwhile sufficiently described natural languages for human-machine communications brought the subject *tagging* onto the agenda again: Looking at speech synthesis applications for instance in several languages we meet words (gen. text units) whose pronunciation depends on their category-features-combination. Here we have some German examples:

text unit	pronunciation	exemplary c.-f.-combination
1984	*neunzehnhundert-vierundachtzig*	`ADJJ`[4]
	eintausendneun-hundertvierundachtzig	`ADJC`

[1] E.g. the endeavors of the *Fruchtbringende Gesellschaft* to standardize the German spelling and grammar cf. [Ko64]

[2] The early definition of the Latin grammar seems to be the reason why it is true that we are able to understand Cicero's philosophic educational writings by means of "scanty" Latin knowledge, but interpreting medieval Old High German poems turns out to be almost impossible for a native speaker.

[3] In the following designated as *category* and *features* resp. *category-features-combination*.

[4] This tag set is explained in the appendix.

text unit	pronunciation	exemplary c.-f.-combination
1.	*erste*	`ADJO:neMy`
	erstem	`ADJO:deMx`
	Ersten	`NOMEN:nmF`
	erstens	`ADV`
	...	
u. Ä.	*und Ähnliche*	`CONJ NOMEN:nmN`
	und Ähnliches	`CONJ NOMEN:neN`
Status	[ˈʃtaːtUs]	`NOMEN:neM`
	[ʃtaˈtuːs]	`NOMEN:nmM`
Knie	[kniː]	`NOMEN:neN`
	[ˈkniːə]	`NOMEN:nmN`

Table 1.1: Pronunciation of some words in dependence on the c.-f.-combination

This table illustrates how useful a tagger (an algorithm that assigns unique c.-f.-combinations to the words of an input sequence) can be in cases of ambiguous pronunciation variants.

Besides in text-to-speech systems tagging results are utilized as input data for the prosody generation: With the help of these data e.g. the **Papageno** TTS system (cf. [Ro00]) determines the position of symbolic prosody marks (phrase accent etc.), the fundamental frequency at syllable level and the sound duration of each phoneme. The methods explained in this work are partially implemented in the C-program **synther** and included in the **Papageno** system. Other common tagger applications are

- → speech recognition,
- → information retrieval,
- → disambiguation of meaning,
- → lexicography and
- → automatic translation.

The great number of application fields points to just as many different kinds of taggers. They generally can be distinguished regarding their

- ➡ **target domain**
  - → morphological,
  - → syntactic,
  - → semantic and
  - → POS (part-of-speech) taggers

and their

- ➡ **implementation approach**
  - ➞ rule-based and
  - ➞ statistic taggers, the latter being based on
    - → neural networks
    - → Hidden-Markov-Models (HMM) or
    - → n-grams.

In this work an n-gram POS tagger is designed and optimized regarding the requests formulated in [Cu92]:

① robustness:
it is to process any input (including unknown words, special characters),

② efficiency:
it has to be economical of memory and computing time,

③ exactness:
it works with a low error rate (<5%) and

④ adaptability:
it can easily be introduced into different application fields

and besides

⑤ multilinguality:
it is to be adaptable to any language.

1.2 The ⋉-Transformation

1.2.1 Motivation

Apart from several well-tried n-gram handling approaches there is a very efficient procedure being based on the ⋉-transformation developed by the author which is implemented for the first time in the above mentioned **Papageno** TTS system. Here this method is to be shortly commented upon.

For the generation of n-gram[5] statistics we need an input symbol sequence to be analyzed:

$$\mathbf{S} = [S^1, \ldots, S^i, \ldots, S^{i_{max}}]. \tag{1.1}$$

[5]The general definition of the notion *n-gram* can be found in [Ho98].

These symbols derive from a set with x_{max} different symbols[6]:

$$\vec{S}^{\top} = (S_1 \ \cdots \ S_x \ \cdots \ S_{x_{max}}). \tag{1.2}$$

In **S** several frequencies are counted:

$$\underline{f}_1^{\top} = (f_1 \ \cdots \ f_{x_{max}}) \tag{1.3}$$

is a vector containing the occurrence frequencies of the symbols S_x,

$$\underline{f}_2 := \begin{pmatrix} f_{1,1} & \cdots & f_{1,x_{max}} \\ \vdots & & \vdots \\ f_{x_{max},1} & \cdots & f_{x_{max},x_{max}} \end{pmatrix} \tag{1.4}$$

is the transition frequency matrix of two symbols, and analogously we have the tensors

$$\underline{f}_3, \ldots, \underline{f}_\nu, \ldots, \underline{f}_n \tag{1.5}$$

representing the transition frequencies of ν^{th} order.
To store the complete set $\underline{f}_1, \ldots, \underline{f}_n$ we would have to make memory available for

$$\sigma = \sum_{\nu=1}^{n} (x_{max})^\nu = x_{max} \frac{(x_{max})^n - 1}{x_{max} - 1} \tag{1.6}$$

frequency values, for instance 466 GB to generate a trigram language model based on 5000 different words and 32-bit frequency numbers. This example extremely contradicts the demand ② on taggers (cf. section 1.1).

1.2.2 The Forward Transformation and Its Conditions

The $\ltimes$-transformation is used to replace the index-vectors of frequency tensor elements. Representing the indices of the element $f_{x_1,\ldots,x_\nu}$ contained in $\underline{f}_\nu$ it is defined as follows:

$$\boxed{\ltimes(x_1, \ldots, x_\nu) = \sum_{k=1}^{\nu} x_k \cdot \hat{x}^{\nu-k}.} \tag{1.7}$$

This equation called ***forward transformation***[7] is a one-to-one mapping between the ν-tuple $(x_1 \ \cdots \ x_\nu)^{\top}$ and the natural number $\ltimes(x_1, \ldots, x_\nu)$:

$$\boxed{(x_1 \ \cdots \ x_\nu)^{\top} \multimap\!\bullet \ltimes(x_1, \ldots, x_\nu).} \tag{1.8}$$

[6]Each form of the variable x, here utilized as simple index, will be element of the set of whole numbers: $x, x_{max}, x_\nu, \hat{x}$ etc. $\in \mathbb{W} = \{0, 1, 2, \ldots\}$.

[7]The *inverse* $\ltimes$*-transformation* is discussed below.

Very important with regard to the advantageous computing operations in the map domain of the $\ltimes$-transformation is the satisfaction of the condition

$$\boxed{\hat{x} > x_{max}.} \tag{1.9}$$

In doing so we have to take into consideration that $\ltimes(x_1, \ldots, x_\nu)$ must not exceed a limit stipulated by the bit-width $\beta_\ltimes$ of the designated data type:
The request

$$\ltimes(x_1, \ldots, x_\nu) < 2^{\beta_\ltimes} \tag{1.10}$$

and the relation

$$\ltimes(x_1, \ldots, x_\nu) \leq \sum_{k=1}^{n} x_{max} \cdot \hat{x}^{n-k} = x_{max} \frac{\hat{x}^n - 1}{\hat{x} - 1} \tag{1.11}$$

result in the final demand

$$\boxed{x_{max} \frac{\hat{x}^n - 1}{\hat{x} - 1} < 2^{\beta_\ltimes}.} \tag{1.12}$$

By means of these two equations we can estimate the minimum bit-width of the data type used to store the $\ltimes$-values:

$$\boxed{\beta_\ltimes > \mathrm{ld}((x_{max} + 1)^n - 1).} \tag{1.13}$$

If this requirement is met we can calculate closed solutions of $\hat{x}$'s upper bound for $n \leq 5$. Here they are displayed for $n \leq 3$:

n	value range of $\hat{x}$
1	$x_{max} < \hat{x}$
2	$x_{max} < \hat{x} < \frac{2^{\beta_\ltimes}}{x_{max}} - 1$
3	$x_{max} < \hat{x} < \sqrt{\frac{2^{\beta_\ltimes}}{x_{max}} - \frac{3}{4}} - \frac{1}{2}$

Table 1.2:
Value Range of $\hat{x}$

Now we introduce a hash table which includes each frequency of the tensors $\underline{f}_1, \ldots, \underline{f}_n$ unequal to zero and their indices' $\ltimes$-transforms sorted after the latter's size (the first column merely serves for clearness):

ν	$\ltimes(x_1, \ldots, x_\nu)$	$f(\ltimes(x_1, \ldots, x_\nu)) \coloneqq f_{x_1,\ldots,x_\nu}$
1	$\ltimes(1) = 1$	$f(\ltimes(1))$
	$\ltimes(2) = 2$	$f(\ltimes(2))$
	$\ldots$	
	$\ltimes(x_{max}) = x_{max}$	$f(\ltimes(x_{max}))$
2	$\ltimes(x_1, x_2) = x_1\hat{x} + x_2$	$f(\ltimes(x_1, x_2))$
	$\ltimes(\bullet_1, \bullet_2)$[8]	$f(\ltimes(\bullet_1, \bullet_2))$
	$\ldots$	
$\ldots$		
n	$\ltimes(\bullet_1, \ldots, \bullet_n)$	$f(\ltimes(\bullet_1, \ldots, \bullet_n))$

Table 1.3:
Frequency Hash Table

[8] $\bullet_k$ replaces any x in the k^{th} place.

1.2.3 An Instance

Now it is time to illustrate the forward transformation by an example. We look at the symbol sequence

$$\mathbf{S} = [C, A, D, C, D, B, D, B, D, D, A] \tag{1.14}$$

which is to be training material of a trigram, i.e. $n = 3$. The first step is to determine the present symbol set; we get in order of their appearance

$$\vec{S}^{\top} = (S_1 \ \cdots \ S_{x_{max}}) = (C, A, D, B). \tag{1.15}$$

This implies the correlation between the symbols and their indices as well as their number: $x_{max} = 4$. The prescribed machine-intern $\ltimes$-data type disposes of 8 bit, thus after checking the requirement (1.13) we are able to determine the value range of $\hat{x}$ using the third line of table **1.2**:

$$4 < \hat{x} < 7.45 \longrightarrow \hat{x} \in \{5, 6, 7\} \tag{1.16}$$

After choosing $\hat{x} = 6$ we scan the input sequence counting all occurrence and transition frequencies up to $\nu = 3$, calculate the $\ltimes$-transforms and size them into the hash table (the first two columns are merely included to simplify the insight, the last is to be used in section 1.3.3):

ν	partial sequence	$\ltimes(x_1, \ldots, x_\nu)$	$f(\ltimes)$	$\tilde{f}(\ltimes)$
1	$[C]$	1	2	110
	$[A]$	2	2	110
	$[D]$	3	5	255
	$[B]$	4	2	110
2	$[C, A]$	8	1	0
	$[C, D]$	9	1	0
	$[A, D]$	15	1	0
	$[D, C]$	19	1	0
	$[D, A]$	20	1	0
	$[D, D]$	21	1	0
	$[D, B]$	22	2	110
	$[B, D]$	27	2	110
3	$[C, A, D]$	51	1	0
	$[C, D, B]$	58	1	0
	$[A, D, C]$	91	1	0
	$[D, C, D]$	117	1	0
	$[D, D, A]$	128	1	0
	$[D, B, D]$	135	2	110
	$[B, D, D]$	165	1	0
	$[B, D, B]$	166	1	0

Table 1.4:
Example Statistics

1.2.4 Computing Time Behaviour

Since each new frequency entry into the hash table consists of merely two values the demand for memory efficiency is met. Now we want to view the computing time behaviour of this procedure.
The standard method to find an entry in a sized hash table with the lines number σ_t is the *binary search* (v. [Ni00]). Beginning in the middle of the table we either strike directly on the sought $\ltimes$-transform or can decide in which table half the search must be continued. The same procedure is carried out in this part, etc. So the maximum number of searching steps can be computed by

$$\sigma_{s,max} = \mathrm{int}(\mathrm{ld}(\sigma_t)) + 1^9. \tag{1.17}$$

This formula constitutes a good estimation of the seeking step number's expectation on the strength of a relatively great number of records that are not contained in the table, i.e. whose frequency is zero. A short sufficiency proof of the estimation

$$E(\sigma_s) \approx \sigma_{s,max} \tag{1.18}$$

is given here: Assuming that all sought records are contained in the table and all entries are looked for with the same probability $p_r = \frac{1}{\sigma_t}$ we directly hit on the record searched for with the likelihood $p(\sigma_s = 1) = p_r$. Otherwise we need a second step and continue our search in the determined table half, where we strike on the correct $\ltimes$-value with the estimated probability $p(\sigma_s = 2) = \frac{1}{\frac{\sigma_t}{2}}$, and so on. The likelihood of needing $\sigma_{s,max}$ steps can be assessed by the probability principle

$$\sum_{k=1}^{\sigma_{s,max}} p(\sigma_s = k) = \frac{1}{\sigma_t}\left(1 + 2 + 2^2 + \cdots + 2^{\sigma_{s,max}-2}\right) + p(\sigma_s = \sigma_{s,max}) = 1 \tag{1.19}$$

which is converted into

$$\begin{aligned} p(\sigma_s = \sigma_{s,max}) &= 1 - \frac{2^{\sigma_{s,max}-2}}{\sigma_t}\left(\sum_{k=0}^{\infty}\left(\frac{1}{2}\right)^k - \frac{1}{2^{\sigma_{s,max}-1}}\sum_{k=0}^{\infty}\left(\frac{1}{2}\right)^k\right) \\ &= 1 + \frac{1}{\sigma_t}\left(1 - 2^{\sigma_{s,max}-1}\right). \end{aligned} \tag{1.20}$$

Now the expectation of σ_s can be calculated:

$$\begin{aligned} E(\sigma_s) &= \sum_{k=1}^{\sigma_{s,max}} k \cdot p(\sigma_s = k) \\ &= \frac{1}{\sigma_t}\sum_{k=1}^{\sigma_{s,max}-1} k \cdot 2^{k-1} + \sigma_{s,max} + \frac{\sigma_{s,max}}{\sigma_t}(1 - 2^{\sigma_{s,max}-1}) \\ &= \frac{1}{\sigma_t}(1 + \sigma_{s,max} - 2^{\sigma_{s,max}}) + \sigma_{s,max}. \end{aligned} \tag{1.21}$$

[9] $\mathrm{int}(r)$ is the integer part of the real number r and $\mathrm{ld}(r) = \log_2 r$.

Introducing an approximation for big tables ($\sigma_t \gg 1$) we get

$$E(\sigma_s) \approx \sigma_{s,max} - \frac{2^{\sigma_{s,max}}}{\sigma_t}\,. \tag{1.22}$$

The difference term between result (1.22) and the assertion (1.18) is

$$\Delta_E = -\frac{2^{\sigma_{s,max}}}{\sigma_t} \tag{1.23}$$

whose bounds we calculate as follows:
When we replace σ_t by the real number r we can regard Δ_E as a function of r with the aid of (1.17):

$$\Delta_E(r) = -\frac{2^{\mathrm{int}(\mathrm{ld}(r))+1}}{r}\,. \tag{1.24}$$

Between the jumps of the int-function the numerator of $\Delta_E(r)$ is constant, the function itself is monotone. It remains the examination of the discontinuity points: $r = 2^w \pm \varepsilon$ with $w \in \mathbb{W}$ and $\varepsilon \in \mathbb{R}$. Inserting this substitution into (1.24) and computing the lower and upper limit for ε towards zero we get

$$\lim_{\varepsilon\to 0} \Delta_E(2^w + \varepsilon) = -2 \quad \text{resp.} \quad \lim_{\varepsilon\to 0} \Delta_E(2^w - \varepsilon) = -1. \tag{1.25}$$

Accordingly the greatest expected deflection between the approximation value of (1.18) and the actual one are two searching operations.

To investigate the computing time behaviour of this storage method a general model of the n-gram composition is to be introduced: In 1935 the linguist George Kingsley Zipf proposed a law that describes the occurrence probabilities of words in natural languages [cf. Le02]:

$$p_{zipf}(S_t) = \frac{A}{B+t}\,. \tag{1.26}$$

Here A and B are constants depending on the language and $p_{zipf}(S_t)$ is the estimated likelihood of the t^{th} most probable word. On the basis of an example is to be made plausible that this law is also suitable for modeling the $\ltimes$-hash table:

- 1660 sentences of the *Frankfurter Allgemeine Zeitung* (FAZ) with $i_{max} = 42201$ words,
- tag set of $x_{max} = 215$ symbols,
- trigram analysis ($n = 3$).

As result we get an $\ltimes$-hash table with $t_{max} = \sigma_t = 14312$ entries, thus we can estimate the Zipf constant A assuming that B is negligibly small:

$$B = 0 \;\rightsquigarrow\; A \approx \frac{1}{ln(\sigma_t) + C} = 0.099^{10}. \tag{1.27}$$

[10] cf. [Sü01] p. 24; C is the Euler constant: $C = 0.57721566\ldots$.

The following graphic demonstrates the resemblance of the real frequency distribution $f(\ltimes_t)$ and the predicted one $f_{zipf}(\ltimes_t) = j_{max} \cdot p_{zipf}(\ltimes_t)$[11]:

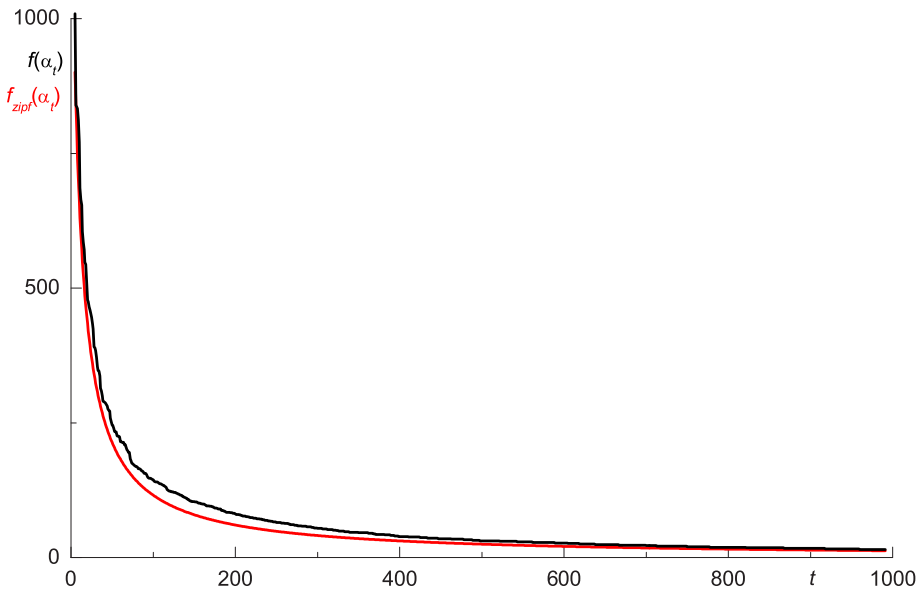


Figure 1.1: Modeling the Frequency Distribution of the Hash Table

Now we consider the j_{max} $\ltimes$-transforms which were arranged into the table during the training to be a sequence of t_{max} different symbols:

$$\ltimes = [\ltimes^1, \ldots, \ltimes^j, \ldots, \ltimes^{j_{max}}]\,; \tag{1.28}$$

in doing so each symbol is to be uniformly distributed within this sequence. Here this arrangement is illustrated by means of the symbol $\ltimes_t$:

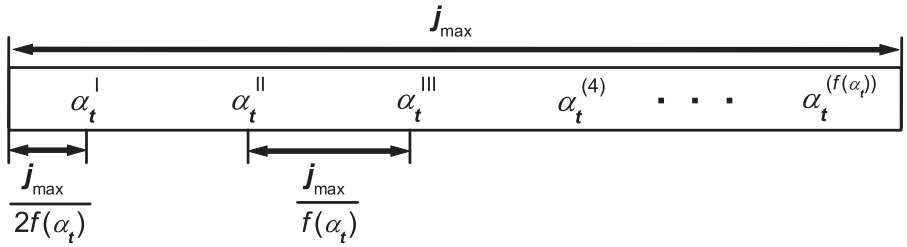


Figure 1.2: Uniform Distribution of the Symbol $\ltimes_t$

[11] $j_{max} = n \cdot (i_{max} - \frac{n+1}{2})$ is the sum of all frequencies in the hash table.

On the above conditions the first appearance of the symbol $\ltimes_t$ is expected in the place (utilizing equation (1.26) with $B = 0$)

$$j \approx \frac{j_{max}}{2f(\ltimes_t)} = \frac{t}{2A}\,. \tag{1.29}$$

When the t^{th} symbol emerges, the table has a size of $\sigma_t = t$ records, thus the latter can be estimated in dependence on the current index j. However we must not forget the maximum table size prescribed by ZIPF's constant A (cf. formula (1.27)) that results in a switching index j_s:

$$\sigma_t(j) \approx \begin{cases} 2Aj & \text{for } j \le j_s \\ e^{\frac{1}{A}-C} & \text{else} \end{cases} \quad \text{with } j_s \approx \frac{e^{\frac{1}{A}-C}}{2A} \tag{1.30}$$

The total computing expenditure to produce the table is composed of the searching part and a part that calculates the j_{max} $\ltimes$-transforms:

$$\sigma_\Sigma = \sigma_{\Sigma,\,s} + \sigma_{\Sigma,\,\ltimes}\,. \tag{1.31}$$

Replacing (1.17) by its real number representation $\sigma_s = \text{int}(\text{ld}(\sigma_t)) + 1 \approx \text{ld}(\sigma_t)$ we get

$$\begin{aligned} \sigma_{\Sigma,\,s} &\approx \sum_{j=1}^{j_{max}} \text{ld}(\sigma_t(j)) = \sum_{j=1}^{j_s} \text{ld}(2Aj) + \sum_{j=j_s+1}^{j_{max}} \text{ld}(e^{\frac{1}{A}-C}) \\ &= \text{ld} \prod_{j=1}^{j_s} 2Aj + \sigma_{\Sigma,\,s\text{II}} = \text{ld}((2A)^{j_s} j_s!) + \sigma_{\Sigma,\,s\text{II}} \\ &\text{with } \sigma_{\Sigma,\,s\text{II}} = (j_{max} - j_s)(\frac{1}{A} - C)\frac{1}{\ln 2}\,.^{12} \end{aligned} \tag{1.32}$$

Consulting the STIRLING formula (v. [Br96])

$$w! \approx \sqrt{2\pi w}\left(\frac{w}{e}\right)^w \tag{1.33}$$

equation (1.32) becomes

$$\sigma_{\Sigma,\,s} \approx \text{ld}\,\sqrt{2\pi j_s} + j_s \text{ld}\frac{2Aj_s}{e} + \sigma_{\Sigma,\,s\text{II}}\,. \tag{1.34}$$

As modern processors need double the time of an integer addition to execute a multiplication the operations number to compute an $\ltimes$-value is $\sigma_\ltimes = 2\nu + (\nu - 1)$ (cf. (1.7)). Since most of the table records are of the order n merely setting $\nu = n$ for all j_{max} computations is reasonable:

$$\sigma_{\Sigma,\,\ltimes} < j_{max} \cdot (3n - 1)\,. \tag{1.35}$$

[12] In case of $j_s > j_{max}$ the value of j_s must be replaced by j_{max}.

The first addend of (1.34) is negligible for great j_{max} so finally we have

$$\sigma_\Sigma(j_{max}) \approx \begin{cases} j_{max}(\mathrm{ld}\dfrac{Aj_{max}}{e} + 3n) & \text{for } j_{max} \le j_s \\ j_{max}\left(\dfrac{\frac{1}{A} - C}{\ln 2} + 3n - 1\right) - \dfrac{js}{\ln 2} & \text{else} \end{cases} \tag{1.36}$$

Looking at the above instance the computing expenditure is to be estimated. First we calculate the sum of all hash table frequencies[13]

$$j_{max} = 126\,597 \tag{1.37}$$

and the switching index (cf. (1.30))

$$j_s \approx 72\,605. \tag{1.38}$$

On the strength of these conditions the second line of equation (1.36) is chosen:

$$\sigma_\Sigma \approx 2\,655\,692 \text{ ops}^{13}. \tag{1.39}$$

1.2.5 Parallelization

The strong demand for ever increasing computing velocity on the one hand and the processor speed limitation because of physical realities on the other have stimulated extensive discussions about the parallelization of mathematical algorithms. One possibility to continue the way of exponential performance development is the employment of multiprocessor computers.

Creating n-gram statistics with the help of the $\ltimes$-transformation can be distributed as follows: Instead of sequentially making one table analyzing i_{max} input symbols now separately m hash tables are produced, each of them processing $\frac{i_{max}}{m}$ input symbols. After the training they are combined utilizing a simple **scanning procedure** which is to be explained on the basis of two sorted lists: $A = \{a_1, a_2, \ldots\}$ and $B = \{b_1, b_2, \ldots\}$.

① a_1 is taken and B is scanned from its start until the first element b_i greater than a_1.

② a_1 is inserted in front of b_i.

③ Now the next element of A is taken and the scanning goes on starting at the place of the last inserted element.

④ This action is continued until the end of A.

Again this method is realizable in parallel manner: When we have $m = 2^\mu$ tables, $2^{\mu-1}$ combinations of two tables are shaped simultaneously. $2\sigma_t(\frac{j_{max}}{2^\mu})$[15] compar-

[13] v. footnote 11.

[14] ops means *processor operations*.

[15] Two lengths of a table including the information of merely $\frac{i_{max}}{m}$ input symbols.

isons and at the most half this number for summing up the frequencies of records with equal $\ltimes$-values are needed. In the next step again always two of the hash tables which have come into being are combined consuming less than $\sigma_t(\frac{j_{max}}{2^{\mu-1}})$ additions plus twice as many comparisons etc. Finally the combining duration can be estimated by the following operations number (cf. equation (1.30)):

$$\sigma_{\Sigma,\,c} \approx 3 \sum_{k=1}^{\mu} \sigma_t(\frac{j_{max}}{2^k}) \leq 3 \sum_{k=1}^{\mu} \frac{2Aj_{max}}{2^k} = 6Aj_{max}(1 - \frac{1}{m})\,. \tag{1.40}$$

Proceeding on the assumption that the computing time of each task processing one table part is nearly the same, the final expenditure is merely

$$\sigma_{\Sigma\|} = \sigma_\Sigma(\frac{j_{max}}{m}) + \sigma_{\Sigma,\,c}\,. \tag{1.41}$$

The ratio between the operations number employing merely one processor and that of the parallel architecture is a useful quantity:

$$\frac{\sigma_\Sigma}{\sigma_{\Sigma\|}} = \frac{\sigma_\Sigma(j_{max})}{\sigma_\Sigma(\frac{j_{max}}{m}) + \sigma_{\Sigma,\,c}}\,, \tag{1.42}$$

and looking at formulas (1.36) and (1.40) we notice its limit as m approaches infinity; in doing so the quotient $\frac{j_{max}}{m}$ is held constant:

$$\lim_{\substack{m\to\infty \\ \frac{j_{max}}{m}=\text{c.}}} \frac{\sigma_\Sigma}{\sigma_{\Sigma\|}} = \frac{\frac{\frac{1}{A}-C}{\ln 2} + 3n - 1}{6A}\,. \tag{1.43}$$

Finally we discuss a practical instance: A 5-gram is to be developed on the basis of an input symbol vector with one million elements. The symbol set is strongly limited, thus the expected maximum hash table size is merely $\sigma_t = 200\,000$. Searched for is the operations number σ_Σ using the sequential mode and the acceleration ratio $\frac{\sigma_\Sigma}{\sigma_{\Sigma\|}}$ if a high performance computer with 16 processors is employed.

- → The first step is to lay down $j_{max} \approx n \cdot i_{max} = 5\,000\,000$;
- → $A \approx 0.078$ arises from (1.27);
- → because $j_s \approx 1\,278\,329 < j_{max}$ (v. equation (1.30)) the second line of (1.36) is utilized and we get $\underline{\sigma_\Sigma(j_{max}) \approx 156 \cdot 10^6 \text{ ops}}$;
- → now we want to calculate $\sigma_\Sigma(\frac{j_{max}}{m})$ and notice that $\frac{j_{max}}{m} = 321\,500$ is less than j_s, so this time we fall back on the first line of (1.36): $\sigma_\Sigma(\frac{j_{max}}{m}) \approx 8.79 \cdot 10^6$ ops;
- → the last step is to estimate $\sigma_{\Sigma,c} \approx 2.2 \cdot 10^6$ ops by formula (1.40)
- → and we get the ratio sought for: $\underline{\frac{\sigma_\Sigma}{\sigma_{\Sigma\|}} \approx 14.2}$.

1.3 Viterbi Algorithm of n^{th} Order Utilizing the $\ltimes$-Transformation

1.3.1 Motivation

Many tasks in several engineering sciences require an algorithm which determines the most probable path through a symbol grid. For example methods of error protection coding, of object processing or speech recognition call for effective approaches to solve this problem. We will see that such strategies also represent an important part of the realization of n-gram taggers. Here we want to look at a symbol grid that consists of $\imath_{max}$ symbol hypothesis vectors of the lengths $\underline{h}_{max} = \{h^1_{max}, h^2_{max}, \ldots, h^{\imath_{max}}_{max}\}$:

$$\overline{S} = (\vec{S}^1, \vec{S}^2, \ldots, \vec{S}^{\imath_{max}}) = \left(\begin{pmatrix} S_{x^1_1} \\ \vdots \\ S_{x^1_{h^1_{max}}} \end{pmatrix}, \ldots, \begin{pmatrix} S_{x^{\imath_{max}}_1} \\ \vdots \\ S_{x^{\imath_{max}}_{h^{\imath_{max}}_{max}}} \end{pmatrix} \right). \tag{1.44}$$

In the following merely the symbol indices $x^\imath_{h^\imath}$ are viewed. Each element of vector $\vec{x}^\imath$ is connected with each of vector $\vec{x}^{\imath+1}$; other edges are forbidden. On this condition the number of different paths from $\vec{x}^1$ to $\vec{x}^{\imath_{max}}$ is

$$\sigma_p = \prod_{\imath=1}^{\imath_{max}} h^\imath_{max} \,. \tag{1.45}$$

Here an example grid with the vector lengths $h_{max} = \{2, 3, 1, 2, 4\}$ is displayed; the above formula results in 48 different paths:

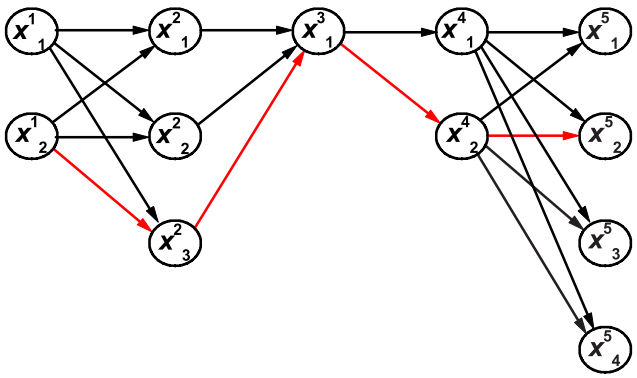


Figure 1.3: Example Grid

Every path stands for a symbol chain of the length $\imath_{max}$, e.g. the red path of Figure 1.3 corresponds to the index sequence

$$\mathbf{x} = [x^1_2, x^2_3, x^3_1, x^4_2, x^5_2] \,. \tag{1.46}$$

To simplify matters in the following we want to utilize the definition $x^\imath := x^\imath_{h^\imath}$ *if it is not expressly annulled.* So the probability of a sequence $\mathbf{x} = [x^1, \ldots, x^{\imath_{max}}]$ utilizing the results of n-gram statistics can be estimated by means of (cf. [Sü01]):

$$p(\mathbf{x}) = \prod_{k=1}^{n-1} p(x^k|x^1, \ldots, x^{k-1}) \prod_{l=n}^{\imath_{max}} p(x^l|x^{l-n+1}, \ldots, x^{l-1}) \tag{1.47}$$

$$\text{with } p(x^\imath|x^\ell, \ldots, x^{\imath-1}) = p(x^\imath) \text{ for } \ell = \imath\,.$$

Special cases of this equation for the orders $n = 1 \ldots 3$ can be found in [Be99]:

n	$p(\mathbf{x})$
1	$\prod_{\imath=1}^{\imath_{max}} p(x^\imath)$
2	$p(x^1) \prod_{\imath=2}^{\imath_{max}} p(x^\imath|x^{\imath-1})$
3	$p(x^1)p(x^2|x^1) \prod_{\imath=3}^{\imath_{max}} p(x^\imath|x^{\imath-2}, x^{\imath-1})$

Table 1.5: Estimated Likelihood of the Sequence $\mathbf{x}$ in Dependence on the Maximum Order n

In section 1.2.1 we found that we generally do not store the probabilities needed in (1.47) but the frequencies which can be used to estimate the first-mentioned by shaping the relative frequencies: The occurrence likelihood arises from

$$p(x^\imath) \approx \frac{f(x^\imath)}{i_{max}} \tag{1.48}$$

and the transition probability is

$$\begin{aligned} p(x^\imath|x^\ell, \ldots, x^{\imath-1}) &= \frac{p(x^\ell, \ldots, x^\imath)}{p(x^\ell, \ldots, x^{\imath-1})} \\ &\approx \frac{f(x^\ell, \ldots, x^\imath) \cdot (i_{max} - \imath + 1 + \ell)}{f(x^\ell, \ldots, x^{\imath-1}) \cdot (i_{max} - \imath + \ell)} \end{aligned} \qquad (\ell < \imath)\,. \tag{1.49}$$

On these conditions equation (1.47) becomes

$$\begin{aligned} p(\mathbf{x}) &\approx c \prod_{k=1}^{n-1} \frac{f(x^1, \ldots, x^k)}{f(x^1, \ldots, x^{k-1})} \prod_{l=n}^{\imath_{max}} \frac{f(x^{l-n+1}, \ldots, x^l)}{f(x^{l-n+1}, \ldots, x^{l-1})} \\ &= c \cdot f(x^1, \ldots, x^{n-1}) \prod_{l=n}^{\imath_{max}} \frac{f(x^{l-n+1}, \ldots, x^l)}{f(x^{l-n+1}, \ldots, x^{l-1})} \end{aligned} \tag{1.50}$$

$$\text{with } f(x^\ell, \ldots, x^\imath) = 1 \text{ for } \ell > \imath \text{ and } c = \begin{cases} 1 & \text{for } n = 1 \\ \frac{(i_{max}-n+2)^{\imath_{max}-n}}{(i_{max}-n+1)^{\imath_{max}-n+1}} & \text{else} \end{cases}$$

The determination of the most probable path through the symbol grid is expressible with the help of

$$\mathbf{x}_p = \arg \max_{\substack{x^1_{h^1}|h^1=1,\ldots,h^1_{max} \\ \vdots \\ x^{\imath_{max}}_{h^{\imath_{max}}}|h^{\imath_{max}}=1,\ldots,h^{\imath_{max}}_{max}}} p(\mathbf{x})\,. \tag{1.51}$$

Analogously to the behaviour of the relation sign of inequations, the arg max operation ignores additions and multiplications with positive constants within its argument, thus the factor c in formula (1.50) can be omitted. Besides the utilization of logarithms to replace the great number of multiplications by additions in order to halve the computing expenditure seems to be advantageous. E.g. the **Papageno** tagger **synther** uses the following technique: While training, the hash table's frequencies are logarithmically converted and normalized to exhaust the stipulated data field. When $\beta_{\tilde{f}}$ is the bit-width of the data type receiving the converted values, these steps can be expressed formally by

$$\tilde{f}(\ltimes_t) \approx (2^{\beta_{\tilde{f}}} - 1)\frac{\ln f(\ltimes_t)}{\ln \max\limits_{t=1,\ldots,\sigma_t} f(\ltimes_t)}\,. \tag{1.52}$$

Now the new quantity $\varpi(\mathbf{x})$ which does not agree with a probability anymore takes the place of the estimated likelihood $p(\mathbf{x})$:

$$\varpi(\mathbf{x}) = \sum_{\imath=n}^{\imath_{max}} \varpi^\imath(\ltimes(x^{\imath-n+1},\ldots,x^\imath),\ltimes(x^{\imath-n+1},\ldots,x^{\imath-1})) \tag{1.53}$$

$$\text{with } \varpi^\imath(\ltimes_{\mathrm{I}},\ltimes_{\mathrm{II}}) = \begin{cases} 0 & \text{for } \imath < n \\ \tilde{f}(\ltimes_{\mathrm{I}}) & \text{for } \imath = n \\ \tilde{f}(\ltimes_{\mathrm{I}}) - \tilde{f}(\ltimes_{\mathrm{II}}) & \text{else} \end{cases}\,; \tag{1.54}$$

$$\text{in doing so } \tilde{f}(\ltimes(x^\ell,\ldots,x^\imath)) = \tilde{f}(0) = 0 \text{ for } \ell > \imath\,.$$

It is true that the number of operations to determine the result of (1.51) already has been strongly reduced by the above reflections but even the current size is unacceptable:

$$\sigma_{\Sigma,p} = 2\sigma_p(\imath_{max} - n + 1)\,. \tag{1.55}$$

To assess this number's proportions the expectation of h_{max} is necessary:

$$E(h_{max}) = \sum_{k=1}^{x_{max}} p(h_{max} = k)\cdot k\,, \tag{1.56}$$

furthermore the expectation of σ_p results from

$$E(\sigma_p) = E(\prod_{\imath=1}^{\imath_{max}} h^{\imath}_{max}) = \sum_{k^1=1}^{x_{max}} \sum_{k^2=1}^{x_{max}} \cdots \sum_{k^{\imath max}=1}^{x_{max}} \prod_{\imath=1}^{\imath_{max}} p(h^{\imath}_{max} = k^{\imath}) \cdot k^{\imath} \,. \tag{1.57}$$

When the individual $h^{\imath}_{max}$ are independent of each other and of their horizontal position (expressed by the index $\imath$), this expression can be simplified:

$$E_i(\sigma_p) = \left(\sum_{k=1}^{x_{max}} p(h_{max} = k) \cdot k \right)^{\imath_{max}} = E(h_{max})^{\imath_{max}} \,. \tag{1.58}$$

In tagging applications the number of ambiguous category-features-combinations of a word S^i corresponds to its h^i_{max}. Returning to the instance of 1.2.4 we have $i_{max} = 42201$ and the following frequency distribution:

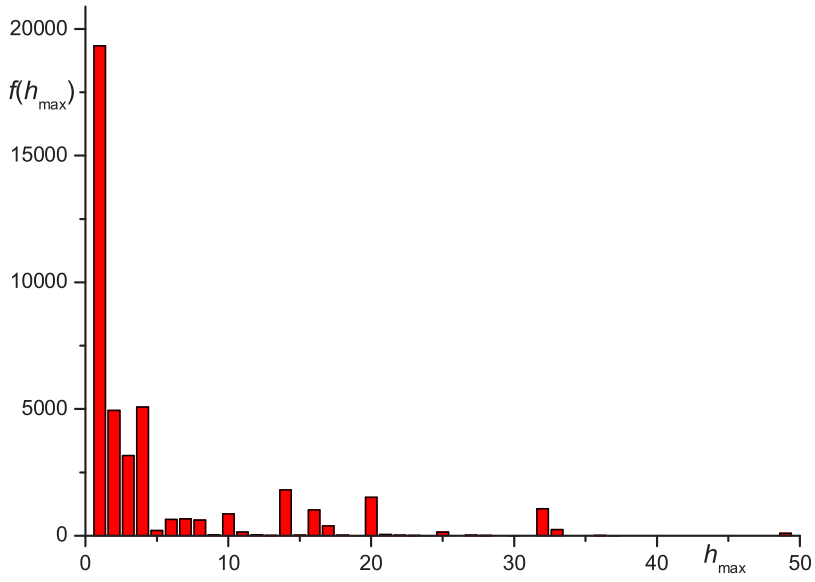


Figure 1.4: Frequency Distribution of h_{max}

In this example we guess $E(h_{max}) \approx 5.2$, and prescribing the small grid-length $\imath_{max} = 10$ we expect $E_i(\sigma_p) \approx 14 \cdot 10^6$ (14 million different paths).

1.3.2 Viterbi Algorithm

In view of the huge number of grid paths it is not recommendable to follow each of them. The principle of the *Viterbi Algorithm* (also designated as *Dynamic Programming*) shows how to reduce the path variety and nevertheless find the most probable way. In relevant literature (e.g. [Ho98]) this approach is discussed for bigrams, here is to be developed a method of any order $n \geq 1$ which falls back on the $\ltimes$-transformation.

First some agreements are reached: The designation of vector elements

$$\vec{v} = (v_1 \; v_2 \; \cdots \; v_{max})^\top \tag{1.59}$$

and the *vector-of-vectors dissolution*

$$\begin{pmatrix} \vec{v} \\ \vec{w} \end{pmatrix} = \begin{pmatrix} v_1 \\ \vdots \\ v_{max} \\ w_1 \\ \vdots \\ w_{max} \end{pmatrix} \tag{1.60}$$

are prearranged, besides a *vector-scalar addition* is introduced

$$\vec{v} + s = \begin{pmatrix} v_1 + s \\ \vdots \\ v_{max} + s \end{pmatrix}. \tag{1.61}$$

Some further scalar operations (like division etc.) are applied to vectors by carrying them out element by element.
Now some preconditions are defined

$$\vec{\kappa}^{*\,0} = (0), \qquad \vec{\varpi}^{*\,0} = (0), \qquad h^{\imath}_{max} = 1 \text{ for } \imath < 1 \tag{1.62}$$

and the forward-part of the Viterbi Algorithm is executed recursively for $\imath \in \{1, \ldots, \imath_{max}\}$ particularly using equation (1.54):

① **The $\ltimes$-Vector**

$$\vec{\kappa}^{\,\imath} = \begin{pmatrix} \hat{x}\vec{\kappa}^{*\,\imath-1} - \hat{x}^n(\vec{\kappa}^{*\,\imath-1} \operatorname{div} \hat{x}^{n-1}) + x^{\imath}_1 \\ \vdots \\ \hat{x}\vec{\kappa}^{*\,\imath-1} - \hat{x}^n(\vec{\kappa}^{*\,\imath-1} \operatorname{div} \hat{x}^{n-1}) + x^{\imath}_{h^{\imath}_{max}} \end{pmatrix} \tag{1.63}$$

② **The Converted Frequency Vector**

$$\vec{\varpi}^{\,\imath} = \begin{pmatrix} \vec{\varpi}^{*\,\imath-1} \\ \vdots \\ \vec{\varpi}^{*\,\imath-1} \end{pmatrix} + \varpi^{\imath}(\vec{\kappa}^{\,\imath}, \vec{\kappa}^{\,\imath} \operatorname{div} \hat{x}) \tag{1.64}$$

③ **The Reduced Index Vector**

$$\vec{k}^{*\imath} = \begin{pmatrix} \underset{k_1=1,\ldots,h_{max}^{\imath-n+1}}{\arg\max}\, \varpi_{k_1}^{\imath} \\ \underset{k_2=h_{max}^{\imath-n+1}+1,\ldots,2h_{max}^{\imath-n+1}}{\arg\max}\, \varpi_{k_2}^{\imath} \\ \vdots \\ \underset{k_{l^\imath}=(l^\imath-1)h_{max}^{\imath-n+1}+1,\ldots,l^\imath\cdot h_{max}^{\imath-n+1}}{\arg\max}\, \varpi_{k_{l^\imath}}^{\imath} \end{pmatrix} \text{ with } l^\imath = \prod_{k=\imath-n+2}^{\imath} h_{max}^k \tag{1.65}$$

④ **The Reduced $\ltimes$-Vector**

$$\vec{\ltimes}^{*\imath} = (\ltimes_{k_1^{*\imath}}^{\imath} \ \cdots \ \ltimes_{k_{l^\imath}^{*\imath}}^{\imath})^\top \tag{1.66}$$

⑤ **The Reduced Converted Frequency Vector**

$$\vec{\varpi}^{*\imath} = (\varpi_{k_1^{*\imath}}^{\imath} \ \cdots \ \varpi_{k_{l^\imath}^{*\imath}}^{\imath})^\top \tag{1.67}$$

After finishing these recursions the final $\ltimes$-transform of the most probable sequence $\mathbf{x}_p$ is determined by

$$\ltimes_p^{\imath_{max}} = \ltimes_k^{*\imath_{max}} \text{ with } k = \underset{l=1,\ldots,l^{\imath_{max}}}{\arg\max}\, \varpi_l^{*\imath_{max}} . \tag{1.68}$$

$\ltimes_p^{\imath_{max}}$ is directly convertible into the last n symbol indices by means of the *inverse* $\ltimes$*-transformation* explained below:

$$x_p^\imath = (\ltimes_p^{\imath_{max}} \operatorname{div} \hat{x}^{\imath_{max}-\imath}) \bmod \hat{x} \quad \text{for } \imath \in \{\imath_{max}-n+1,\ldots,\imath_{max}\} . \tag{1.69}$$

Generally the symbols for $\imath \in \{\imath_{max}-1,\ldots,1\}$ result from the again recursive Viterbi back-tracking:

① **The $\ltimes$-Transform**

$$\ltimes_p^\imath = \ltimes_k^{*\imath} \quad \text{with } k = l|\ltimes_l^{*\imath} \bmod \hat{x}^{n-1} = \ltimes_p^{\imath+1} \operatorname{div} \hat{x} . \tag{1.70}$$

② **The Symbol Index**

$$x_p^\imath = \ltimes_p^\imath \bmod \hat{x} \tag{1.71}$$

The Viterbi Algorithm highly cuts down the great variety of included paths which now is represented by the dimension number of the $\ltimes$-vector. Its expectation is analogously to (1.58)

$$E_i(\sigma_{p,V}) = E(h_{max})^n . \tag{1.72}$$

1.3.3 Examples

Since the last section's formulas eventually appear to be quite difficult to the reader, now they are underpinned by some examples. As basis statistics we want to utilize those of section 1.2.3, where the converted frequencies already are registered for a bit-width of $\beta_{\tilde{f}} = 8$ (cf. equation (1.52)). Besides the example grid of 1.3.1 with the following symbol index assignment is used:

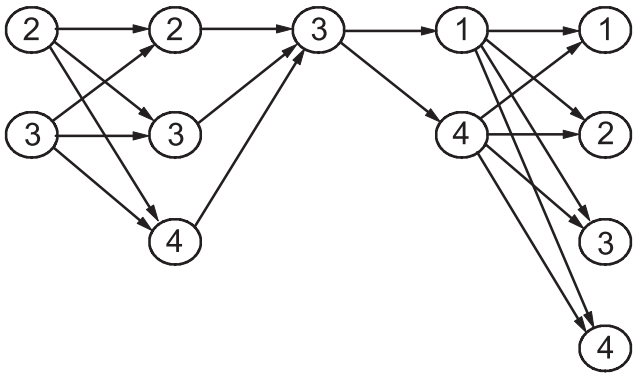


Figure 1.5: Example Grid

Now the calculations for the orders $n \in \{1, 2, 3\}$ are realized and documented in the form of a table (to simplify matters the converted frequency $\tilde{f}_{\mathrm{fl}} \to -\infty$ is assigned to an $\ltimes$-transform which was not seen during the training; a more accurate approach is discussed in the next chapter):

n	status						
1	forward-part	$\imath$	$\vec{\ltimes}^{\imath}$	$\vec{\varpi}^{\imath}$	$\vec{k}^{*\imath}$	$\vec{\ltimes}^{*\imath}$	$\vec{\varpi}^{*\imath}$
		1	$(2\ 3)^{\top}$	$(110\ 255)^{\top}$	(2)	(3)	(255)
		2	$(2\ 3\ 4)^{\top}$	$(365\ 510\ 365)^{\top}$	(2)	(3)	(510)
		3	(3)	(765)	(1)	(3)	(765)
		4	$(1\ 4)^{\top}$	$(875\ 875)^{\top}$	(1)	(1)	(875)
		5	$(1\ 2\ 3\ 4)^{\top}$	$(985\ 985\ 1130\ 985)^{\top}$	(3)	(3)	(1130)
	back-tracking	$\imath$	$\ltimes_p^{\imath}$	$x_p^{\imath}$			
		5	3	3			
		4	1	1			
		3	3	3			
		2	3	3			
		1	3	3			
	result:	$\mathbf{x}_{p,n=1} = [3, 3, 3, 1, 3]$					

n status

2 forward-part

$\imath$	$\vec{\ltimes}^{\imath}$	$\vec{\varpi}^{\imath}$	$\vec{k}^{*\imath}$	$\vec{\ltimes}^{*\imath}$	$\vec{\varpi}^{*\imath}$
1	$(2\ 3)^\top$	$(0\ 0)^\top$	$(1\ 2)^\top$	$(2\ 3)^\top$	$(0\ 0)^\top$
2	$\begin{pmatrix} 14 \\ 10 \\ 15 \\ 21 \\ 16 \\ 22 \end{pmatrix}$	$\begin{pmatrix} \tilde{f}_{\text{fl}} \\ 0 \\ 0 \\ 0 \\ \tilde{f}_{\text{fl}} \\ 110 \end{pmatrix}$	$\begin{pmatrix} 2 \\ 3 \\ 6 \end{pmatrix}$	$\begin{pmatrix} 20 \\ 15 \\ 22 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 110 \end{pmatrix}$
3	$(15\ 21\ 27)^\top$	$(-110\ -255\ 110)^\top$	(3)	(27)	(110)
4	$(19\ 22)^\top$	$(-145\ -35)^\top$	$(1\ 2)^\top$	$(19\ 22)^\top$	$(-145\ -35)^\top$
5	$\begin{pmatrix} 7 \\ 25 \\ 8 \\ 26 \\ 9 \\ 27 \\ 10 \\ 28 \end{pmatrix}$	$\begin{pmatrix} \tilde{f}_{\text{fl}} - 255 \\ \tilde{f}_{\text{fl}} - 145 \\ -255 \\ \tilde{f}_{\text{fl}} - 145 \\ -255 \\ -35 \\ \tilde{f}_{\text{fl}} - 255 \\ \tilde{f}_{\text{fl}} - 145 \end{pmatrix}$	$\begin{pmatrix} 2 \\ 3 \\ 6 \\ 8 \end{pmatrix}$	$\begin{pmatrix} 25 \\ 8 \\ 27 \\ 28 \end{pmatrix}$	$\begin{pmatrix} \tilde{f}_{\text{fl}} - 145 \\ -255 \\ -35 \\ \tilde{f}_{\text{fl}} - 145 \end{pmatrix}$

back-tracking

$\imath$	$\ltimes_p^{\imath}$	$x_p^{\imath}$
5	27	3
4	22	4
3	27	3
2	22	4
1	3	3

result: $\mathbf{x}_{p,n=2} = [3, 4, 3, 4, 3]$

3 forward-part

$\imath$	$\vec{\ltimes}^{\imath}$	$\vec{\varpi}^{\imath}$	$\vec{k}^{*\imath}$	$\vec{\ltimes}^{*\imath}$	$\vec{\varpi}^{*\imath}$
1	$(2\ 3)^\top$	$(0\ 0)^\top$	$(1\ 2)^\top$	$(2\ 3)^\top$	$(0\ 0)^\top$
2	$\begin{pmatrix} 14 \\ 10 \\ 15 \\ 21 \\ 16 \\ 22 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{pmatrix}$	$\begin{pmatrix} 14 \\ 10 \\ 15 \\ 21 \\ 16 \\ 22 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$
3	$\begin{pmatrix} 87 \\ 123 \\ 93 \\ 129 \\ 99 \\ 135 \end{pmatrix}$	$\begin{pmatrix} \tilde{f}_{\text{fl}} \\ \tilde{f}_{\text{fl}} \\ \tilde{f}_{\text{fl}} \\ \tilde{f}_{\text{fl}} \\ \tilde{f}_{\text{fl}} \\ 110 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 3 \\ 6 \end{pmatrix}$	$\begin{pmatrix} 87 \\ 93 \\ 135 \end{pmatrix}$	$\begin{pmatrix} \tilde{f}_{\text{fl}} \\ \tilde{f}_{\text{fl}} \\ 110 \end{pmatrix}$

	$\imath$	$\vec{\ltimes}^{\imath}$	$\vec{\varpi}^{\imath}$	$\vec{k}^{*\imath}$	$\vec{\ltimes}^{*\imath}$	$\vec{\varpi}^{*\imath}$
	4	$\begin{pmatrix} 91 \\ 127 \\ 163 \\ 94 \\ 130 \\ 166 \end{pmatrix}$	$\begin{pmatrix} \tilde{f}_{\text{fl}} \\ 2\tilde{f}_{\text{fl}} \\ \tilde{f}_{\text{fl}} \\ 2\tilde{f}_{\text{fl}} \\ \tilde{f}_{\text{fl}} \\ 0 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 6 \end{pmatrix}$	$\begin{pmatrix} 91 \\ 166 \end{pmatrix}$	$\begin{pmatrix} \tilde{f}_{\text{fl}} \\ 0 \end{pmatrix}$
	5	$\begin{pmatrix} 115 \\ 133 \\ 116 \\ 134 \\ 117 \\ 135 \\ 118 \\ 136 \end{pmatrix}$	$\begin{pmatrix} 2\tilde{f}_{\text{fl}} \\ \tilde{f}_{\text{fl}} - 110 \\ 2\tilde{f}_{\text{fl}} \\ \tilde{f}_{\text{fl}} - 110 \\ \tilde{f}_{\text{fl}} \\ 0 \\ 2\tilde{f}_{\text{fl}} \\ \tilde{f}_{\text{fl}} - 110 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{pmatrix}$	$\begin{pmatrix} 115 \\ 133 \\ 116 \\ 134 \\ 117 \\ 135 \\ 118 \\ 136 \end{pmatrix}$	$\begin{pmatrix} 2\tilde{f}_{\text{fl}} \\ \tilde{f}_{\text{fl}} - 110 \\ 2\tilde{f}_{\text{fl}} \\ \tilde{f}_{\text{fl}} - 110 \\ \tilde{f}_{\text{fl}} \\ 0 \\ 2\tilde{f}_{\text{fl}} \\ \tilde{f}_{\text{fl}} - 110 \end{pmatrix}$

back-tracking

$\imath$	$\ltimes_p^{\imath}$	$x_p^{\imath}$
5	135	3
4	166	4
3	135	3
2	22	4
1	3	3

result: $\mathbf{x}_{p,n=3} = [3, 4, 3, 4, 3]$

Table 1.6: Viterbi Algorithm Instances for Unigram, Bigram and Trigram

1.4 Formulas of the $\ltimes$-Transformation

Here a few calculation rules of the $\ltimes$-transformation are collected. One of them is exemplarily proven after the list.

- **Conditions**

$$\vec{x} = (x_1 \ \cdots \ x_\nu)^\top; \quad \vec{x}^{\imath} = (x^{\imath} \ \cdots \ x^{\imath+\nu-1})^\top; \quad x_k \in \{1, \ldots, \hat{x} - 1\} \tag{1.73}$$

- **Forward Transformation**

$$\ltimes = \sum_{k=1}^{\nu} x_k \cdot \hat{x}^{\nu-k} \tag{1.74}$$

- **Inverse Transformation**

$$\vec{x} = \begin{pmatrix} \ltimes \operatorname{div} \hat{x}^{\nu-1} \\ \vdots \\ (\ltimes \operatorname{div} \hat{x}^{\nu-k}) \bmod \hat{x} \\ \vdots \\ \ltimes \bmod \hat{x} \end{pmatrix} \quad \text{with } k = 1, \ldots, \nu \tag{1.75}$$

- **Calculation Rules**
  - **Linearity**

$$a \cdot \vec{x} + b \cdot \vec{y} \multimap\bullet\ a \cdot \ltimes_x + b \cdot \ltimes_y \text{[15]} \tag{1.76}$$

  - **Shortening**

$$\begin{aligned} (x_1 \ \cdots \ x_{\nu-k})^\top &\multimap\bullet\ \ltimes \operatorname{div} \hat{x}^k; \quad k \in \mathbb{W} \\ (x_{\nu-k+1} \ \cdots \ x_\nu)^\top &\multimap\bullet\ \ltimes \bmod \hat{x}^k; \quad k \in \mathbb{W} \wedge k \leq \nu \end{aligned} \tag{1.77}$$

  - **Shift**

$$\begin{aligned} \vec{x}^{\imath+1} &\multimap\bullet\ \hat{x} \ltimes^\imath -\hat{x}^\nu (\ltimes^\imath \operatorname{div} \hat{x}^{\nu-1}) + x^{\imath+\nu} \\ \vec{x}^{\imath-1} &\multimap\bullet\ \ltimes^\imath \operatorname{div} \hat{x} + \hat{x}^{\nu-1} \cdot x^{\imath-1} \end{aligned} \tag{1.78}$$

- **Example Correspondences**
  - **Geometric Series**

$$(1 \ \cdots \ 1)^\top \multimap\bullet\ \frac{1 - \hat{x}^\nu}{1 - \hat{x}} \tag{1.79}$$

  - **Binomial Series**

$$(1 \ \cdots \ \nu)^\top \multimap\bullet\ \frac{\nu - \hat{x} - \nu\hat{x} + \hat{x}^{\nu+1}}{(1 - \hat{x})^2} \tag{1.80}$$

  - **Alternating Series**[17]

$$(1 \ \cdots \ (-1)^{\nu-1})^\top \multimap\bullet\ \frac{1 - (-\hat{x})^\nu}{1 + \hat{x}} \tag{1.81}$$

  - **Pascal's Triangle**

$$\begin{aligned} (1\ 1)^\top &\multimap\bullet\ \ltimes \quad \curvearrowright \\ (1\ 2\ 1)^\top &\multimap\bullet\ \ltimes^2 \\ (1\ 3\ 3\ 1)^\top &\multimap\bullet\ \ltimes^3 \quad \text{etc.} \end{aligned} \tag{1.82}$$

[16] On condition that $a \cdot x_k + b \cdot y_k \in \{1, \ldots, \hat{x} - 1\}$ for $k \in \{1, \ldots, \nu\}$.

[17] As negative values within $\vec{x}$ are not allowed, this correspondence merely should be used in coherence with the linearity rule in order to return into an admissible range of values.

As an instance the validity of the forward shift rule is to be proven. For that purpose in the first place the $\ltimes$-transforms of $\vec{x}^{\,\imath}$ and $\vec{x}^{\,\imath+1}$ are computed:

$$\ltimes^{\imath} = \sum_{k=\imath}^{\imath+\nu-1} x^k \cdot \hat{x}^{\imath+\nu-1-k}\,, \tag{1.83}$$

$$\ltimes^{\imath+1} = \sum_{k=\imath+1}^{\imath+\nu} x^k \cdot \hat{x}^{\imath+\nu-k}\,. \tag{1.84}$$

Executing the div-operation on $\ltimes^{\imath}$ we get

$$\ltimes^{\imath} \operatorname{div} \hat{x}^{\nu-1} = \left(x^{\imath} \cdot \hat{x}^{\nu-1} + \sum_{k=\imath+1}^{\imath+\nu-1} x^k \cdot \hat{x}^{\imath+\nu-1-k} \right) \operatorname{div} \hat{x}^{\nu-1} = x^{\imath} \tag{1.85}$$

since

$$\sum_{k=\imath+1}^{\imath+\nu-1} x^k \cdot \hat{x}^{\imath+\nu-1-k} \leq \sum_{k=\imath+1}^{\imath+\nu-1} (\hat{x}-1) \cdot \hat{x}^{\imath+\nu-1-k} = \hat{x}^{\nu-1} - 1 < \hat{x}^{\nu-1}\,. \tag{1.86}$$

Now equation (1.78) is consulted

$$\hat{x} \cdot \left(x^{\imath} \cdot \hat{x}^{\nu-1} + \sum_{k=\imath+1}^{\imath+\nu-1} x^k \cdot \hat{x}^{\imath+\nu-1-k} \right) - \hat{x}^{\nu} \cdot x^{\imath} + x^{\imath+\nu} = \sum_{k=\imath+1}^{\imath+\nu} x^k \cdot \hat{x}^{\imath+\nu-k}\,, \text{ q.e.d.} \tag{1.87}$$

In conclusion the use of the above calculation rules is demonstrated on the basis of the $\ltimes$-transform of the vector $(\nu \ \cdots \ 1)^{\top}$:

$$\ltimes_{\nu,\ldots,1} \bullet\!\!-\!\!\circ \begin{pmatrix} \nu \\ \vdots \\ 1 \end{pmatrix} = (\nu+1) \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} - \begin{pmatrix} 1 \\ \vdots \\ \nu \end{pmatrix} \circ\!\!-\!\!\bullet\ (\nu+1) \cdot \ltimes_{1,\ldots,1} - \ltimes_{1,\ldots,\nu} \ \curvearrowright \tag{1.88}$$

$$\ltimes_{\nu,\ldots,1} = (\nu+1)\frac{1-\hat{x}^{\nu}}{1-\hat{x}} - \frac{\nu - \hat{x} - \nu\hat{x} + \hat{x}^{\nu+1}}{(1-\hat{x})^2} = \frac{1 - \hat{x}^{\nu} - \nu\hat{x}^{\nu} + \nu\hat{x}^{\nu+1}}{(1-\hat{x})^2}\,. \tag{1.89}$$

Chapter 2

The Application: synther

2.1 synther as a Module of the TTS System Papageno

The tagger **synther** consists of two general components:

- the **Training Part**
  This element reads an ASCII-symbol sequence (the training material) of any length and symbol number. It automatically produces a list (symbol table) of all emerged symbols and assigns indices to them. Simultaneously it computes all $\ltimes$-transforms up to the order n and shapes the $\ltimes$-hash table. Accordingly this approach is application-independent, it can be utilized to realize several statistics, to construct a language model (when the symbols are words of a natural language) or as basis of a tagger (here the symbols are category-features-combinations). In the latter case the procedure is completely language-independent since it automatically learns its symbol set by means of the training material.

- and the **Viterbi Part**
  It is a module of the TTS system **Papageno** between the multilingual text processing engine that contains the tokenizer[1], number handling etc. and the phonetic transcription module (cf. [Ro00]). As input it expects a symbol index grid (the indices of the ambiguous c.-f.-combinations of a sentence) and returns a symbol index sequence representing the disambiguated c.-f.-combinations. Data basis is the $\ltimes$-hash table created in the training part.

[1] After separating the unicode input text to be tagged into the individual words, their ambiguous c.-f.-combinations are determined enlisting a linguistic dictionary. Now this symbol grid is converted into the desired symbol index grid integrating the symbol table.

The following figure containing all components directly communicating with **synther's** parts illustrates the principles just commented upon:

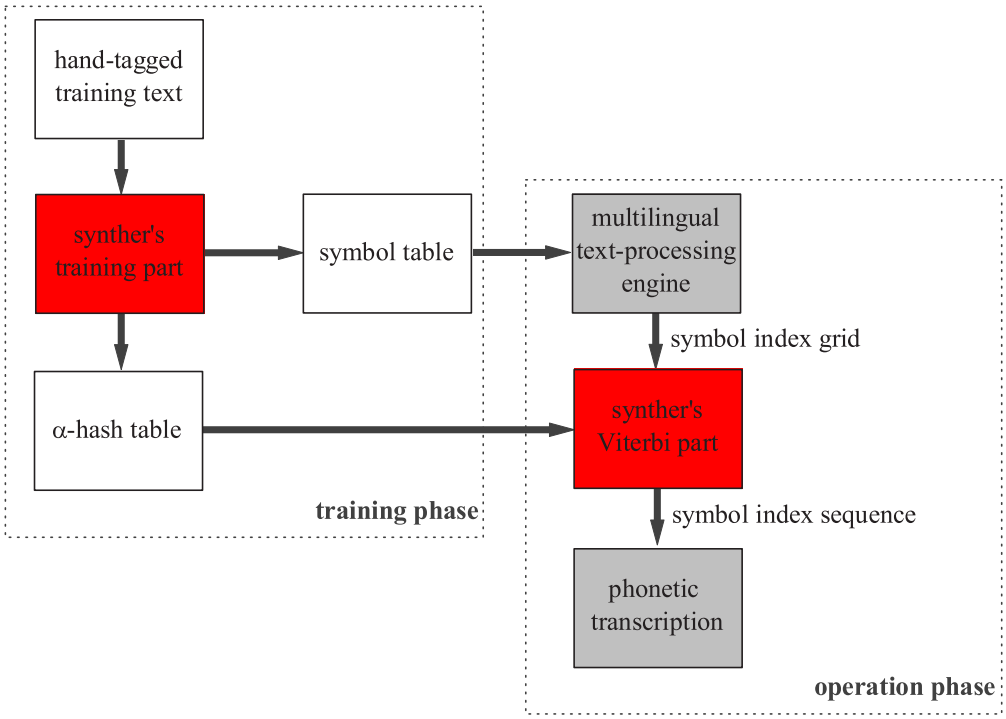


Figure 2.1: Connection between **synther's** parts and their neighboring components

2.2 Aphorisms with Reference to synther's Exactness

Returning to the five demands on a tagger formulated in 1.1 we notice that apart from the efficiency of which a detailed account is given in chapter 1 also the robustness, adaptability and multilinguality are met, since **synther's** design explained in the last section is totally independent of the nature of training material. During the operation phase **synther** merely communicates with its neighbored modules within the symbol index layer, thus these requests are met here, too. It remains the examination of item ③: exactness.

2.2.1 Maximum and Average Error Rate before Training

It is true that this section's title sounds paradoxical as we actually possess no information at all, but the secret is revealed when we bear in mind the linguistic dictionary which is consulted by the tokenizer while generating the symbol index grid. In dependence on language and tag set there is a considerable number of words with one single c.-f.-combination. Their share in the text material to be tagged during the operation phase determines the maximum error rate which is expressible with the help of their estimated occurrence probability:

$$ER_{max} = 1 - p(h_{max} = 1)\,. \tag{2.1}$$

Invoking the example of 1.2.4 resp. 1.3.1 we get $ER_{max,1} \approx 0.542$ at $x_{max,1} = 215$ different c.-f.-combinations. Considering essentially less tags the maximum error rate strongly goes down. The following table shows the h_{max}-distribution for a highly reduced tag set of only $x_{max,2} = 20$ different c.-f.-combinations:

k	$f(h_{max} = k)$	$p(h_{max} = k)$
1	33015	0.782
2	8412	0.199
3	629	0.015
4	145	0.003

Table 2.1:
Distribution of Frequency Resp. Estimated Probability of h_{max}

For this tag set the expected maximum error rate amounts to $ER_{max,2} \approx 0.218$.

The simplest imaginable tagger is a routine that purely coincidentally or observing a fixed model settles on a possible symbol. In this case we expect an average error rate of

$$\overline{ER} = 1 - \sum_{k=1}^{x_{max}} \frac{p(h_{max} = k)}{k}\,. \tag{2.2}$$

According to this equation the above instances result in $\overline{ER}_1 = 0.410$ resp. $\overline{ER}_1 = 0.112$.

As matters stand the efforts of the next sections are aimed at an essential improvement of these rates. So the effect of the following measures actually has to be gauged in relation to the average error rate, however with regard to the comparability with other taggers the absolute error rate is to be equally considered.

2.2.2 The Coherence between Training Material Size and Tagging Error Rate

Having developed the theory of $\ltimes$-Viterbi Tagging it is time to evaluate **synther's** recognition performance (we should accept this designation in connection with the determination of correct tags which actually are to be *recognized*). In literature, e.g. concerning speech processing, the unanimous opinion that trigrams in principle represent an error rate improvement as against bigrams is held. In fact this claim is correct in case of disposing of a certain minimum training material size. Generally the consideration of wider contexts provides better results if the n-gram statistics are nearly saturated. The greater the order the more extensive basis material is necessary to meet this condition. Still in areas of scarcity with the help of smaller orders it has well-balanced statistics generated. Against this background the expected course of the family of error rate curves in dependence on the number of trained symbols i_{max} can be guessed:

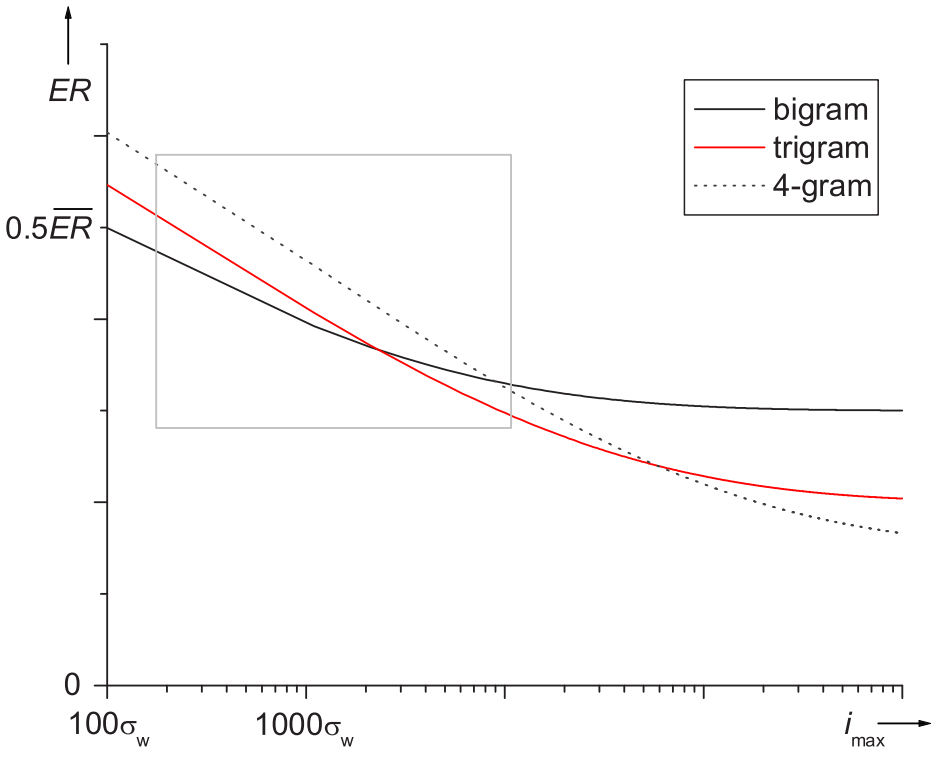


Figure 2.2: Expected Tagging Error Rates in Dependence on i_{max}

Because the mathematical modeling of tagging errors would go beyond the scope of this work, the above family of curves is intuitively defined by the characteristics worked out in the beginning of this section and within the previous one. In particular it was set store by the limit situation, after all we have

$$ER_n(i_{max}) = (\overline{ER} - \overrightarrow{ER}_n)e^{-\frac{1}{n}\sqrt[\sigma_l]{\frac{i_{max}}{\sigma_w}}} + \overrightarrow{ER}_n\,. \tag{2.3}$$

Figure 2.2 corresponds to the parameters

$$\overrightarrow{ER}_2 = 0.3\overline{ER}, \qquad \overrightarrow{ER}_3 = 0.2\overline{ER}, \qquad \overrightarrow{ER}_4 = 0.15\overline{ER}, \qquad \sigma_l = 5, \qquad \sigma_w = 1\,.$$

To illustrate these reflections now the hand-tagged newspaper articles (v. example of 1.2.4) serve as basis material for an n-gram training. Consulting 219 sentences of different internet pages, particularly selected to verify the number pronunciation (cp. section 1.1 and the first two rows of table 1.1) and utilizing them as test material, the tagging error rates are determined for $n \in \{2, \ldots, 4\}$ and 25 to 1660 training sentences using the small tag set (v. appendix):

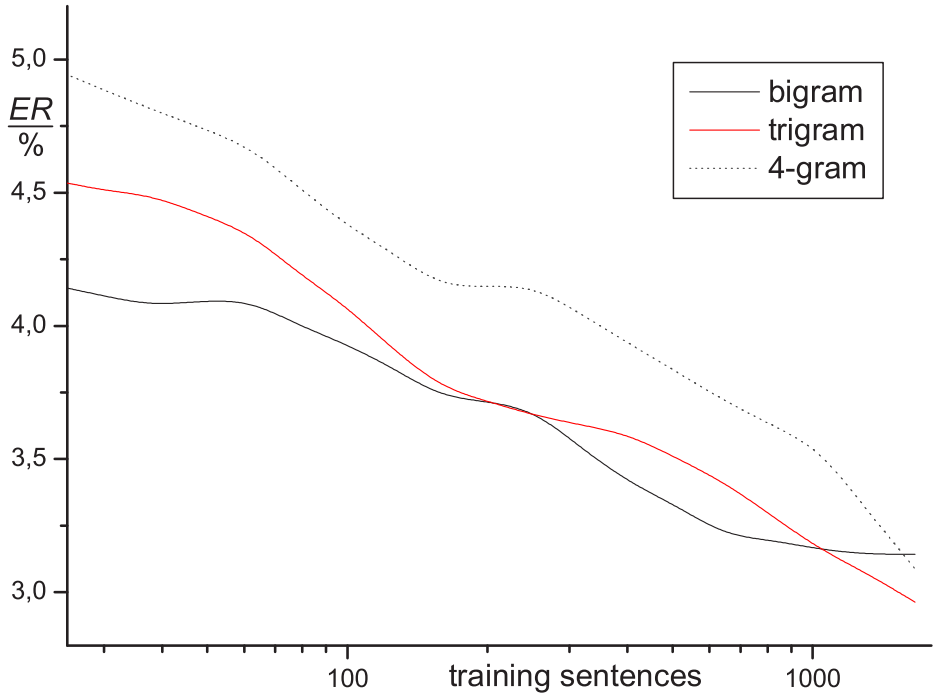


Figure 2.3: Real Tagging Error Rates in Dependence on the Number of Training Sentences

In order to integrate this figure into the previous one the light gray rectangle of the latter marks the former outlines. It is conspicuous that the training with more input data (e.g. the tenfold amount) in all probability would essentially improve the tagging results towards greater orders.

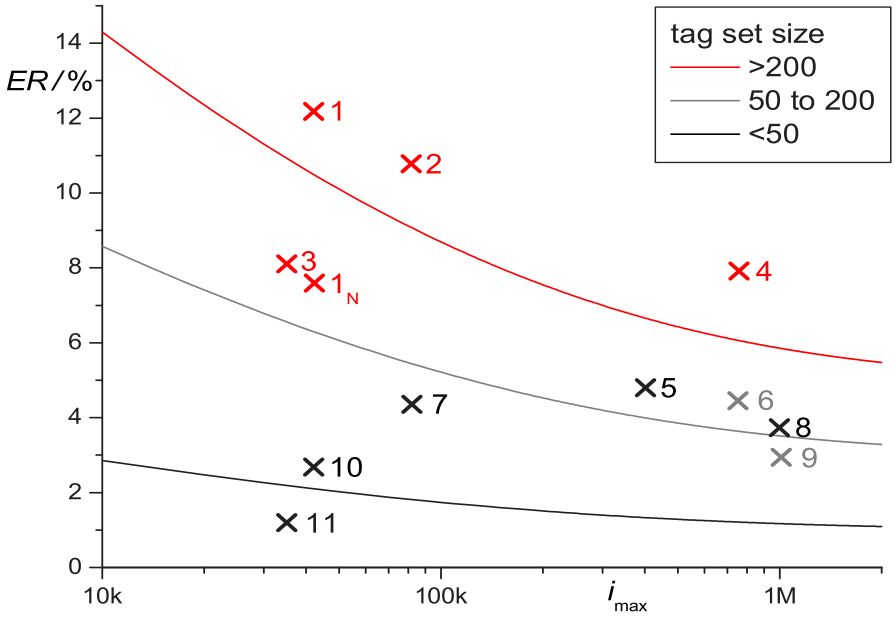
2.2.3 synther by Comparison with Other Taggers

In section 2.2.1 the strong dependence of the average error rate $\overline{ER}$ on the tag set size x_{max} was shown. Therefore to assess as objectively as possible the different taggers are subdivided into three classes regarding x_{max}:

class	x_{max}	$\overline{ER}$	$\overrightarrow{ER}$
small	$x_{max} < 50$	0.1	0.01
middle	$50 \leq x_{max} \leq 200$	0.3	0.03
great	$x_{max} > 200$	0.5	0.05

Table 2.2: Tagger Classes with Regard to x_{max}

The parameters $\overline{ER}$ and $\overrightarrow{ER}$ are utilized to draw guessed error rate curves of these three classes for $n = 4$ as support into the diagram which documents the recognition performance of 11 tagger instances:



#	tagger	$ER/\%$	corpus	i_{max}	x_{max}	source
$1_{[N]}$	synther	12.17[7.59]	FAZ	42 201	215	
2	TnT	10.78	'1984'	81 805	>200	[Dž00]
3	IST	8.1	FAZ	34 912	219	[Wi01]
4	HMM	7.94	Wotan	≈ 750k	233	[Ha01]
5	NN	4.84	MTP	≈ 400k	14	[Ha99]
6	MET	4.44	Wotan	≈ 750k	129	[Ha01]
7	RBT	4.36	'1984'	81 805	12	[Dž00]
8	TBL	3.72	WSJ	≈ 1M	48	[Ha01]
9	MBT	2.94	LOB	≈ 1M	170	[Ha01]
10	synther	2.68	FAZ	42 201	20	
11	IST	1.2	FAZ	34 912	11	[Wi01]

Figure 2.4: Different Taggers' Error Rates in Dependence on Their Class and i_{max}

To explain this figure's abbreviations:

FAZ:	Newspaper corpus v. section 1.2.4,
TnT:	**T**rigram'**n**'**T**ags by Thorsten Brants,
'1984':	George Orwell's novel (cf. [Or49]),
IST:	**I**terative **S**tochastic **T**agging suggested by Petra Witschel,
HMM:	Tagging approach using **H**idden **M**arkov **M**odels,
Wotan:	Tag set for Dutch based on the Eindhoven corpus,
NN:	**N**eural **N**etwork tagger developed by Horst-Udo Hain,
MTP:	**M**ünster **T**agging **P**roject corpus,
MET:	**M**aximum **E**ntropy **T**agger written by Adwait Ratnaparkhi,
RBT:	**R**ule **B**ased **T**agger written by Eric Brill,
TBL:	Error-driven **T**ransformation-**B**ased **L**earning by Eric Brill,
WSJ:	**W**all **S**treet **J**ournal material,
MBT:	**M**emory-**B**ased Learning proposed by Walter Daelemans,
LOB:	**L**ancaster-**O**slo/**B**ergen corpus

In view of the restricted tag set **synther** shows its strength by producing better recognition results (#10) than all the competitors except of the **IST** (#11).

Still it is true that within the great tag set class **synther** causes the worst result (#1), but consulting the related taggers **TnT** (#2) and **HMM** (#4) the main reason seems to be the scarcity of training material. Nevertheless the next section is to draw up an improvement strategy which is modeled on the excellent error rates of the **IST**. Besides it should be mentioned that the development of **synther** particularly aims at the number pronunciation (cf. 1.1 and 2.2.2). Therefore it is a very important fact that the rate of incorrectly recognized numerals turns out to be essentially better ($\#1_{\mathrm{N}}$).

2.2.4 Iterative Tagging

In the previous sections the high importance of possessing sufficient training material was shown. However sometimes we are compelled to deal with a strongly limited corpus, e.g. in connection with very specific tagging domains. Another problematic application is the employment of taggers in an area which differs from the training. Such cases are handled in [Wi01] that especially contains the fundamental description of the **I**terative **S**tochastic **T**agging, stated in 2.2.3, too. The solution idea when changing over a scantily trained A-tagger to an unfamiliar domain B that would cause the error rate ER_0 is to execute a tagging with a set of area-B-sample texts. Now this action's result is basis to generate n-gram statistics which are combined with those of the old domain A. The current error rate ER_1 if any text of B would be tagged in general is less than ER_0. Now we can realize a second iteration tagging the B-sample set and integrating the results into further statistics which are combined with the former, and so on. In this way we step by step adapt the area-A-tagger to the B-samples which should be

representative of the future application field. Understandably the exaggeration of this procedure involves a certain danger, since possibly the final statistics are totally adapted to the iteration samples but hardly recognize deviating substitutes of the same domain.

However because of **synther's** very time-economical algorithms the above discussed idea could be perfected by carrying out the iterations during the operation phase where the text to be tagged would replace the B-samples. This consequently requires I enlargements of the $\ltimes$-statistics by i_{max} symbols and $I+1$ taggings of a sentence with the length i_{max} in case of iterating I times. When we roughly calculate the computation costs for an enlargement

$$\sigma_e \approx n \cdot i_{max} \cdot \mathrm{ld}(\sigma_t) \tag{2.4}$$

and that of the Viterbi tagging (cp. algorithm description in 1.3.2)

$$\sigma_v \approx 2i_{max} \cdot (E(h_{max})^n \cdot (8 + \mathrm{ld}(\sigma_t)) + E(h_{max})^{n-1}) \,, \tag{2.5}$$

we notice the negligibility of the first term. Thus the execution of one iteration means one more tagging. So the value of I is to be defined in dependence on the available computation performance and the requirements of other simultaneously running tasks.

Here is to be calculated exemplarily the operations number σ_i for five iterations of a sentence with $i_{max} = 10$ words assuming a trigram tagger with an $\ltimes$-hash table of the length $\sigma_t = 2^{16}$, the expectation of the ambiguous symbol number is to be $E(h_{max}) = 4$:

$$\sigma_i = I \cdot \sigma_e + (I+1)\sigma_v \approx 188\,640\,\mathrm{ops} \,. \tag{2.6}$$

An instance for the significant recognition improvement is shown here:

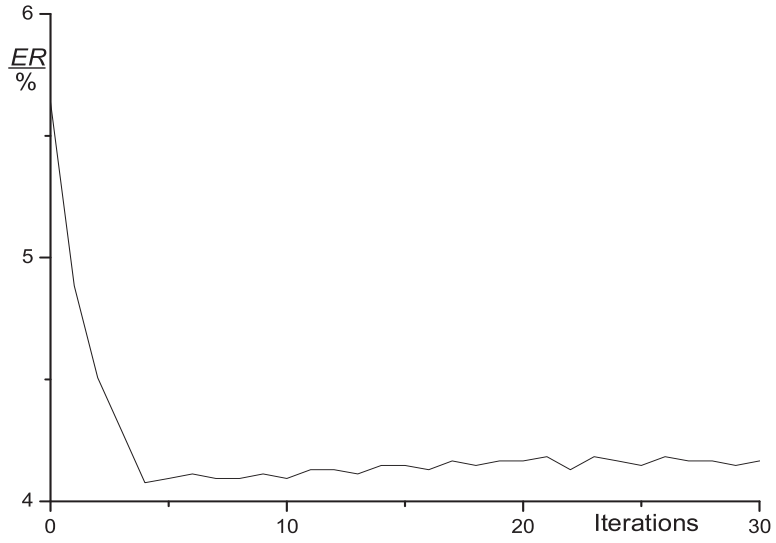


Figure 2.5: Effect of Iterative Tagging on the Error Rate

These example statistics are based on merely 314 sentences of the Münster Tagging Project corpus (mentioned in the previous section) and is iteratively confronted with the same test material as in 2.2.2. After all the error rate was reduced by more than the fourth part. Also the ineffectiveness of taking great iteration numbers is evident; in the above instance the best results already are reached for $I = 4$. Of course this characteristic depends on the specific training and test material and should be inspected when adapting the tagger to another target domain. Likewise the use of greater training and iteration orders than that of the final tagging has proven its worth.

2.2.5 Floor Value Handling in the Light of Numeral Tagging

As announced in 1.3.3 here is to be probed into the important question about the handling of events that have not occurred during the training. One would be inclined to assign the estimated likelihood zero or a small constant value (so-called *floor value*) to them, but because of the structure of the path probability formula (1.50) it is not recommendable: Guessing the conditional probability of a transition with the order ν we would have to execute a division by zero if the denominator transition of the order $\nu - 1$ did not occur, too. Also an adequately small floor value $0 < f_{\mathrm{fl}} < 1$ could not model the matter appropriately, since the determined conditional likelihood would get the value $\frac{f_{\mathrm{fl}}}{f_{\mathrm{fl}}} = 1$ which absolutely does not reflect reality. It is true that a solution is to use floor values $f_{\mathrm{fl},n}$ that depend on the order n, but at any rate better approaches are conceivable:

Definitely Frédéric Bimbot's *multigrams* which represent the likelihood as a weighted sum of all conditional probabilities with the memory $0 \leq \nu < n$ is one of the most intuitive and all-embracing procedures. Through laying down these weights the quality of the floor value modeling is excellently controllable. Unfortunately this method is not consistent with the logarithmic frequency representation, as in this map domain there is no counterpart of the addition that is indispensable. Besides, against the background of measures like iterative tagging the computation expenditure is not acceptable.

Another solution proposal is the punished gradual order decrease until the denominator frequency is unequal to zero, in doing so the punishment is realized with the help of a multiplication by a factor between one and zero; in the logarithmic map domain it is equivalent to an addition with the negative addend $\tilde{f}_{\mathrm{fl}}$. Consequently the converted frequency $\varpi^{\imath}$ (cp. 1.54) is replaced by a new quantity which if necessary reduces the current order utilizing the $\ltimes$-shortening rule (1.77):

$$\begin{gathered}\tilde{\varpi}^{\imath}(\ltimes_{\mathrm{I}}, \ltimes_{\mathrm{II}}) = \varpi^{\imath}(\ltimes_{\mathrm{I}} \bmod \hat{x}^k, \ltimes_{\mathrm{II}} \bmod \hat{x}^k) + (n-k)\tilde{f}_{\mathrm{fl}} \\ \text{with } k = \max_{l=1,\ldots,n} \{l | \ltimes_{\mathrm{I}} \bmod \hat{x}^l \geq 0\}\,.\end{gathered} \tag{2.7}$$

In this connection we are allowed to return to the above zero-likelihood by defining $\tilde{f}(\ltimes) \to -\infty$ if $\ltimes$ did not occur while training.

Now the significant import of the $\tilde{f}_{\mathrm{fl}}$ value is demonstrated by means of the numeral error rate (cf. 1.1 and 2.2.3). To assess objectively the numeral tagging performance, in the first place the maximum and average error rate before training (v. 2.2.1) are calculated. For that purpose the distribution of estimated h_{max}-probabilities exclusively considering numerals in the above test material is consulted:

k	$f(h_{max,\mathrm{N}} = k)$	$p(h_{max,\mathrm{N}} = k)$
1	30	0.095
2	51	0.161
15	1	0.003
24	228	0.722
25	4	0.013
51	2	0.006

Table 2.3: Distribution of Estimated $h_{max,\mathrm{n}}$-Probabilities

Utilizing equations (2.1) and (2.2) the sought rates are computable:

$$ER_{max,\mathrm{N}} \approx 0.905\,, \qquad \overline{ER}_{\mathrm{N}} \approx 0.793\,.$$

Here we notice how indispensable trained knowledge is for the tagging in several application fields. The next diagram demonstrates the course of the numeral error rate in dependence on parameter $\tilde{f}_{\mathrm{fl}}$:

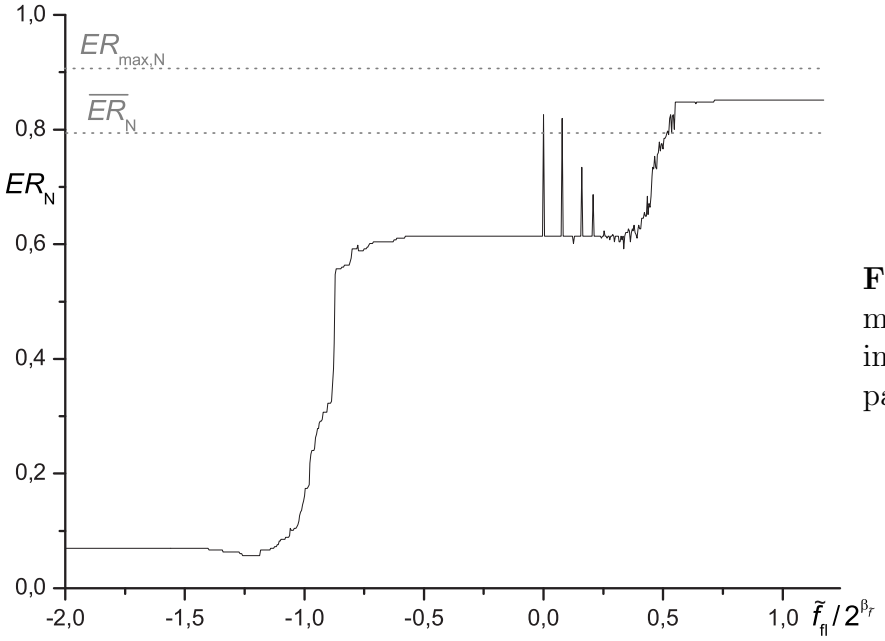


Figure 2.6: Numeral Error Rate in dependence on parameter $\tilde{f}_{\mathrm{fl}}$

In this figure $\tilde{f}_{\text{fl}}$ is normalized concerning the cardinality of the converted frequency's range of values ($2^{\beta_{\tilde{f}}}$); if $\tilde{f}_{\text{fl}}$ is negative its impact agrees with a punishment, positive values correspond to a reward. As expected the latter case raises the error rate to the area of $\overline{ER}_{\text{N}}$ and $ER_{max,\text{N}}$. However also the punishment must be of a minimum size to cause a positive effect: $\tilde{f}_{\text{fl}} \to -\infty$ seems to be a reliable variant ($ER_{\text{N}}(\tilde{f}_{\text{fl}} \to -\infty) \approx 7.6\,\%$, cp. figure 2.4) but there can be a local minimum point which results in a better error rate (e.g. in our example $ER_{\text{N}}(-\frac{5}{4} \cdot 2^{\beta_{\tilde{f}}}) \approx 5.7\,\%$).

2.3 Concluding Annotations and Outlook

In accordance with the demands on tagger development phrased in 1.1 the multilingual tagging system **synther** is robust, efficient concerning running-time as well as memory and adaptable. To improve its recognition results several approaches already have been described. Basically recommendable is a training with an essentially greater corpus to saturate the statistics and make possible the consideration of wider contexts. Still evaluating **synther's** tagging results, two error types coming from exactly demonstrable phenomenons were noticed:

The selection of category-features-combinations which are to shape the tag set must be deliberately executed. An important aspect is the even distribution of new features among all categories. For instance the introduction of the features case, number and gender for determiners but not for pronouns turned out to be a momentous mistake:

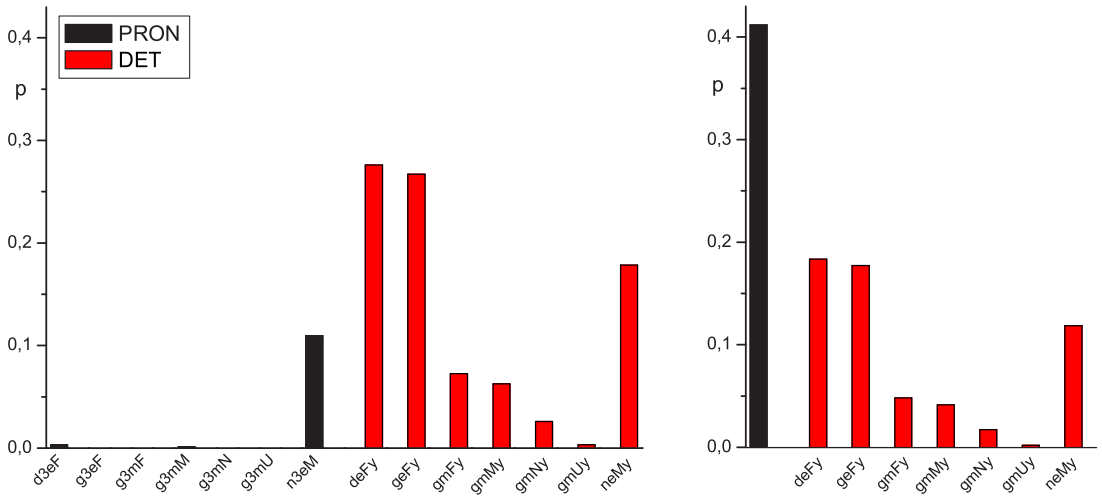


Figure 2.7: The German Word "der": Estimated Occurrence Probabilities of the C.-F.-Combinations with Differentiation of Pronoun Features (Left) and without (Right)

These diagrams show the effect of asymmetrical feature omission in order to restrict the symbol number. With the help of the German word "der" ($\hat{=}$ "the", "that", "who", "which" etc.) it is demonstrated how significantly the estimated probability of a category can increase if its features are ignored. In this special case the Viterbi Algorithm very often falsely settled on pronoun. Even though the symbol set of the left diagram has a greater potency, better tagging results came to light.
Also an occasionally disturbing effect is due to the equal weight of all c.-f.-combinations of a word in the linguistic dictionary. E.g. if the German word "Tor" is neuter it means "gate", "door", "goal" etc. The masculine version stands for "fool" and is an obsolete, seldom utilized notion. Nevertheless it is treated having equal rights because its rareness is considered nowhere. A handsome number of tagging errors is attributable to this insufficient modeling. To remedy things weighted dictionary entries should be introduced, in the course of which an important condition is anew to be provided with a training corpus of representative size to determine the required weights reliably.
As further future extensions are to be mentioned

- **the forced limitation of the path number considered by the Viterbi Algorithm:**
  Indeed the expectation of the included path number is strongly cut down by the Viterbi Algorithm (cp. equation (1.72)), but in case of great orders this restriction may not be enough, too. Therefore a forced dimension limitation of the reduced $\ltimes$-vector (v. (1.66)) with the aid of the most probable elements of the reduced converted frequency vector (v. (1.67)) is conceivable.

- **end-of-sentence detection with synther:**
  This approach is rather simple since no program alterations are required, but the tag set must be slightly modified: Until now we have been utilizing the symbol "SI" to declare a secure end of sentence (cf. appendix). When this information is not given, several punctuation marks in the input text (like ".", "!", "?", ":" and so on) indicate a possible end of sentence expressed by two contradictory symbols, "SI" and "NOSI", which must be learned during the training phase. For instance German ordinals are compounded by the corresponding cardinal plus the character ".". If the ordinal is located at the end of sentence this punctuation mark and the full stop merge. Thus when we find a number followed by the character "." in the text to be tagged, three solutions are possible:

  ADJC PUNCT SI, ADJO PUNCT SI, ADJO PUNCT NOSI.

  By means of the customary Viterbi Algorithm their disambiguation is carried out, at which its limits either are nonambiguous "SI" symbols or they are arranged at fixed positions ignoring the sentence frame.

Appendix A

A.1 Tag Sets

#	symbol	meaning	German example
1	"ACRO"	abbreviation	SPD
2	"ADJ"	adjective	scharfen
3	"ADJC"	cardinal	18000
4	"ADJJ"	year	1984
5	"ADJO"	ordinal	18.
6	"ADV"	adverb	demnach
7	"CONJ"	conjunction	und
8	"DET"	determiner	der
9	"FOREIGN"	foreign word	Foreign
10	"HYPHEN"	hyphenated symbol	wirtschafts-
11	"INTJ"	interjection	Nein
12	"NOMEN"	noun	Film
13	"PDET"	preposition-determiner-merge	im
14	"PREP"	preposition	in
15	"PRON"	pronoun	sich
16	"PUNCT"	punctuation mark	,
17	"SI"	secure end of sentence	
18	"VERB"	verb	müsse
19	"VPART"	verb particle	dar
20	"XINC"	Latin word	absurdum

Table A.1: Small Tag Set, $x_{max} = 20$

#	symbol	case	meaning
1	"ACRO"	"a"	accusative
2	"ADJ"	"d"	dative
3	"ADJ:"+case+number+gender+strongweak	"g"	genitive
4	"ADJC"	"n"	nominative
5	"ADJJ"		
6	"ADJO:"+case+number+gender+strongweak	number	
7	"ADV"	"e"	singular
8	"CONJ"	"m"	plural
9	"DET"		
10	"DET:"+case+number+gender+strongweak	gender	
11	"FOREIGN"	"F"	feminine
12	"HYPHEN"	"M"	masculine
13	"INTJ"	"N"	neuter
14	"NOMEN"	"U"	indefinite
15	"NOMEN:"+case+number+gender		
16	"PDET:"+case+number+gender	strongweak	
17	"PREP"	"x"	strong
18	"PREP:"+case	"y"	weak
19	"PRON"		
20	"PRON:"+case+person+number+gender	person	
21	"PUNCT"	"1"	first person
22	"SI"	"2"	second person
23	"VERB"	"3"	third person
24	"VPART"		
25	"XINC"		

Table A.2: Great Tag Set, $x_{max} = 374$[1]

For example the German word "Hauses" corresponds to the symbol "NOMEN:geN".

Annotation: These Tag Sets are prescribed by the SIEMENS CT IC 5 speech synthesis group and are not subject to the author.

[1]Usually the training material does not contain the whole number of possible symbols, so e.g. the FAZ corpus (cf. section 2.1.4) merely has x_{max}=215.

A.2 synther's Manual

```
synther - Manual
================

Contents
--------

1. Why synther?

2. Training
2.1. The Statistics Input File
2.2. The First Training
2.3. The Symbol Table
2.3.1. The File Identification
2.3.2. Determining the Symbol Number; xMax
2.3.3. The n-gram Order n
2.3.4. Internal and External Symbol Index
2.4. Category and Features
2.5. Print Commands
2.5.1. The Binary Statistics File
2.5.2. Print Status
2.6. Number of Training Sentences

3. Tokenizer Simulation
3.1. What Does the Tokenizer Simulation Do?
3.2. Input File and Dictionary
3.3. The First Tokenizer Simulation
3.4. The Tokenizer Output File

4. Tagging
4.1. Input and Output Files
4.2. The First Tagging
4.3. Comparison and Verification
4.3.1. Print Recognition Results
4.3.2. Consider Features
4.3.2.1. Recognition Errors Considering Features
4.3.2.2. Print Recognition Results Considering Features
4.4. Improve the Tagging Results
4.4.1. Tagging Order
4.4.2. Floor Value
4.4.3. Iterations
4.4.3.1. The First Iteration
4.4.3.2. Two Variants to Raise the Number of Iterations
4.4.3.3. Two Variants for Increasing the Iterative Method's Effectiveness
4.4.3.3.1. Utilizing a Greater Training and Iteration Order than during the
           Test
4.4.3.3.2. Iterating with the Test Set
```

1. Why synther?

synther is an n-gram statistics tool that is used for determining the most probable way through a given symbol network. synther's working method is to be exemplified by a multilingual POS-tagger.

synther.tar.gz should be extracted into a new folder, we want to call it ~/synther/, by means of

```
mkdir ~/synther/
gzip -d synther.tar.gz
mv synther.tar ~/synther/
cd ~/synther/
tar xf synther.tar
rm synther.tar
```

The subfolder lib/ includes the source files which can be compiled on different UNIX derivatives like Sun OS, Linux or Cygwin for Windows using the simple make command in the ~/synther/ path. All following actions are to be executed in this folder; before realizing a command check your location by typing pwd

2. Training

2.1. The Statistics Input File

synther's input data has to be a symbol list in the form of an ASCII-file, in the following called statistics input file. The symbol representation should be alphanumeric (if applicable some special characters can be used aside from ":", whose meaning is explained in 2.4.) and exclusively be separated by word-wrap ("\n") characters. Besides there are two reserved symbols: SI (silence) and NIL (not in list) which have special meanings: Utilizing synther for tagging, speech recognition or similar tasks the silence symbol is used for framing each sentence, so the statistics input file principally looks as follows:

```
SI
DET:neMy  }
...       } 1st sentence
PUNCT     }
SI
SI
PREP:a    }
...       } 2nd sentence
PUNCT     }
SI
...
SI
PRON:n3eM }
...       } last sentence
PUNCT     }
SI
```

An exemplary file is puffer/compareFAZ.txt . The meaning of the NIL-symbol is explained at 3.1.

2.2. The First Training

Now we can realize our first training:

```
synther -c -si puffer/compareFAZ.txt
```

-c means create n-gram statistics and the argument of the -si option is the statistics input file name. For short explanations of synther's arguments v. 5. resp. type synther . Since we occasionally have to pass a long list of arguments synther expects its input and output files in standardized locations if they are not defined explicitly. E.g. the standard statistics input file is user/stat_in.txt . Thus we can shorten the above command by copying our statistics input file to its standard location:

```
cp puffer/compareFAZ.txt user/stat_in.txt
```

Now it is enough typing

```
synther -c
```

2.3. The Symbol Table

During the training two output files are created: the symbol table and the binary statistics file (cf. synther's options -sy and -st). An exemplary structure of the symbol table is

```
!Do not edit this automatically generated list!

file identification:
1013270771

xMax:
1000
```

```
n:
2

index symbol
1     SI
2     DET
3     NOMEN
4     VERB
5     PDET
...
19    INTJ
20    FOREIGN
21    NIL
```

2.3.1. The File Identification
The file identification is needed to check the version equality of the above-mentioned two files - prerequisite for synther's flawless work. This examination is done automatically and throws a warning in case of version conflicts.

2.3.2. Determining the Symbol Number; xMax
xMax is an integer value that can be laid down by means of synther's option -x, otherwise we have xMax = 1000. This parameter always has to be an integer greater than the number of different symbols (in our above instance this demand is met: xMax = 1000 > 21). Example:

```
synther -c -x 22
```

In case we do not know the symbol number we can determine it for the present by assigning a large number to xMax, e.g. 1000000:

```
synther -c -x 1000000
```

Now we can have a look at the index of NIL in the last line of the symbol table, this value is equal to the symbol number.

2.3.3. The n-gram Order n
The integer n is the maximum n-gram order and characterizes the span of the considered transitions; it is defined by -n and has the standard and minimum value n = 2. Between xMax and n the relation xMax^n < 2^64 must apply. Examples:

```
synther -c -x 22 -n 14
```

For POS-Tagging at the most with 21 categories (incl. NIL) the maximum order n = 14 can be taken.

```
synther -c -x 5000 -n 5
```

For a language model on the basis of the Verbmobil training material CD 1 to 5 (with 4936 different words) the maximum order n = 5 can be taken.

2.3.4. Internal and External Symbol Index
The rest of the symbol table contains the correspondence between the ASCII-symbols and their synther-internal index representation. In case of utilizing synther as a module within a system (e.g. for speech recognition or synthesis), the external symbol index representation can differ from the above-mentioned, thus a second correspondence becomes necessary. Modeled on the speech synthesis system Papageno its category list is taken as the external symbol representation standard (cf. puffer/catlist_dt). If we include it into the synther training by

```
synther -c -ca -cl puffer/catlist_dt
```

or

```
cp puffer/catlist_dt user/catlist.txt
synther -c -ca
```

we get a third column of the symbol table which contains the external indices.

2.4. Category and Features
If we compare the contents of the statistics input file puffer/compareFAZ.txt and that of the symbol table we notice that until now only the symbol parts on the left of the ":"-character are taken into consideration. This part is called category, the right one we designate as features. If we want to include the features in the training (= ignore the ":"-character's exceptional effect) we must add the -f option:

```
synther -c -f
```

2.5. Print Commands

2.5.1. The Binary Statistics File
As the Binary Statistics File is not readable with the help of an ASCII-editor, its contents can be printed after the training by the p option:

```
synther -c -p -n 3
```

Of course this information can be detoured into a file:

```
synther -c -p -n 3 > user/stat.txt
```

An exemplary structure of user/stat.txt is

```
line#    alpha    freq.    symbol

1        1        3320     SI
2        2        5909     DET
3        3        10169    NOMEN
4        4        5025     VERB
5        5        735      PDET
...
```

```
19      19      7       INTJ
20      20      5       FOREIGN
21      1001    1659
22      1002    520
23      1003    239
...
70      4003    159
...
1536    20004006 2
1537    20005003 1
1538    20009002 1
```

The front lines show the index assignments like in the symbol table and besides the occurrence frequencies of the respective symbol. The table's second part begins in the line where alpha the first time is greater than xMax. Here we can find the transition frequencies that are coded by means of the alpha values (also designated as addresses). If we want to estimate the frequency of the sequence VERB NOMEN for our above instance, we multiply the index of VERB by xMax and add the index of NOMEN:

```
alpha(VERB NOMEN) = alpha(VERB)*xMax + alpha(NOMEN) = 4*1000 + 3 = 4003
```

In the line where alpha = 4003 we find the searched frequency:

```
freq(VERB NOMEN) = freq(alpha(VERB NOMEN)) = freq(4003) = 159
```

So generally we can seek for a sequence of n symbols by

```
freq(Symbol1 ... Symboln) =
    freq(alpha(Symbol1)*xMax^(n-1) + alpha(Symbol2)*xMax^(n-2) + ...
    + alpha(Symboln-1)*xMax + alpha(Symboln))
```

If alpha(Symbol1 ... Symboln) cannot be found in the table, its frequency is zero.

2.5.2. Print Status

If we want to be informed about the computing status of synther there are two options:

-ps prints detailed information, e.g. how many sentences are trained already.

-pss prints only coarse information, e.g. notifications after finishing the training or tokenizer simulation.

2.6. Number of Training Sentences

When the sentences of the training material are separated by the SI symbol (cf. 2.1.) the number of sentences that are to be considered is controllable by the -s parameter:

```
synther -c -s 10 -ps

only includes the first ten sentences of the statistics input file.


3. Tokenizer Simulation
-----------------------

3.1. What Does the Tokenizer Simulation Do?
Like already described in 1. synther's general task is to find the most
probable way through a symbol network. However dependent on the applica-
tions sometimes this network is not directly available but merely a word
sequence and a symbol dictionary belonging to it. The tokenizer simulation
step by step takes a word out of the sequence, seeks for its representation
in the dictionary and builds with the help of the found symbols the neces-
sary network. If it meets a punctuation mark that indicates an end of sen-
tence (".", "?", "!" or ":"), the silence symbol SI is fitted in twice
(cf. 2.1.).
During this procedure the following exceptions can emerge:

- The dictionary does not include the current word. In this case all sym-
bols contained in the symbol table except for NIL are guessed as represent-
atives.

- The symbol offered by the dictionary has not been seen in the course of
the training. Here the symbol NIL (not in list) is taken.


3.2. Input File and Dictionary
Like already mentioned in 3.1. first we need an input file containing the
word sequence. An instance is puffer/input219.txt:

demnach
müsse
der
Film
in
deutscher
...

The words are only separated by word-wrap ("\n") characters.
An example dictionary is puffer/lex219.txt:

" PUNCT
( PUNCT
) PUNCT
, PUNCT
- PUNCT
...
demnach ADV
...
```

```
Film NOMEN:aeM NOMEN:deM NOMEN:neM
...
müsse VERB
...

The format of each dictionary line has to be

1. word
2. tabulator ("\t") character
3. sequence of symbols separated by blank (" ") characters
4. word-wrap ("\n") character

3.3. The First Tokenizer Simulation
Now we are ready to start our first tokenizer simulation:

synther -token -i puffer/input219.txt -l puffer/lex219.txt -ps

in doing so we use the parameters

-token instructing synther to execute the simulation,
-i to define the input file,
-l to specify the dictionary and
-ps to print the computing status (v. 2.5.2.).

The same effect has the command sequence

cp puffer/input219.txt user/input.txt
cp puffer/lex219.txt user/lex.txt
synther -token -ps

3.4. The Tokenizer Output File
The network constructed by the tokenizer emulation is written into the  to-
kenizer  output file (definable by -to; standard file: user/token.txt) that
is likewise the input file for the tagging algorithm.


4. Tagging
----------

4.1. Input and Output Files
Usually the tagging algorithm needs as  input  the  tokenizer  output  file
(-to; v. 3.4.) as well as its input file (-i; mentioned in 3.2.) since both
the  input  word and its automatically tagged representation are written to
the output file (-o):

input    auto-tagged

         SI
demnach  ADV
müsse    VERB
der      DET
Film     NOMEN
in       PREP
```

Besides the symbol table (-sy; cf. 2.3.) and the binary statistics file (-st; 2.5.1.) are necessary.

4.2. The First Tagging
synther is set to the tagging mode by means of the -t option. To avoid a long argument chain first we move the utilized files (v. 4.1.) to their standard locations and execute

```
synther -t
```

Now we can view the tagging result in the output file.

4.3. Comparison and Verification
If we want to verify synther's tagging quality we should compare its results with a hand-tagged version of the input material. This version must have the same format as demanded for the statistics input file in 2.1. With the help of the verification option -v the tagging results are compared with a comparison file defined by the -comp parameter:

```
synther -c -t -v -comp puffer/compare219.txt -to puffer/token219.txt
(a prepared tokenizer output file is used)

or

cp puffer/token219.txt user/token.txt
cp puffer/compare219.txt user/compare.txt
synther -c -t -v
```

Now the output file user/output.txt has got a third column: the hand-tagged symbol version. If it deviates from synther's assumption, at the end of the line a lozenge ("#") character is inserted.

4.3.1. Print Recognition Results
synther's recognition performance (recognition as notion for the correct determination of the tag belonging to the current word) can be estimated by counting the lozenge ("#") characters of the output file; e.g. by

```
grep -c '#' user/output.txt
```

Taking our above instance we get 174 errors. Now we look after the total symbol number which is the first result of the command

```
wc user/output.txt
```

deducting 2 on account of the file header. We get 5569 taking our example. So the relative error rate is 174/5569 = 0.031 = 3.1% resp. the recognition rate 1-174/5569 = 0.969 = 96.9%
However we can control the recognition performance essentially easier by adding the -pr (print recognition results) option:

```
synther -t -v -pr
```

As results we get
a) the number of words with ambiguous symbols (this value is useful as most of the tagged words permit only a single symbol thus these cases are not significant for the assessment of synther's capacity)
b) the total symbol number
c) the symbol error number
d) the total recognition rate
e) the recognition rate merely considering the number of words with ambiguous symbols; cp. a)

4.3.2. Consider Features
As explained in 2.4. we can include the -f option to consider both categories and features while tagging:

```
synther -c -f
synther -t -f
```

This has consequences in the nature of recognition errors and results:

4.3.2.1. Recognition Errors Considering Features

```
synther -t -v -f
```

generates an output file in the following shape:

```
input         auto-,     hand-tagged
              SI         SI
demnach       ADV        ADV
müsse         VERB       VERB
der           DET:neMy   DET:neMy
...
Staatsbürger  NOMEN:nmM  NOMEN:amM *
als           CONJ       PREP # *
...
3.            ADJO:deMy  ADJO:deMy $
...
3.            ADJO:deFx  ADJO:geFy * %
...
1935          ADJC       ADJJ # * %
...
```

While verifying synther makes use of the four marking characters: "*", "#", "$" and "%" which present themselves in these combinations:

a) "*" means faulty features
b) "# *" marks category errors
c) "$" is a correctly recognized numeral
d) "* %" indicates a numeral's feature error
e) "# * %" shows category mistakes of numerals

4.3.2.2. Print Recognition Results Considering Features

```
synther -t -v -f -pr
```

The now obtained results turn out differently from 4.3.1.:

a) total symbol number
b) number of faulty categories
c) category recognition rate
d) total error number
e) total recognition rate
f) numeral number
g) number of faulty numerals
h) numeral recognition rate

4.4. Improve the Tagging Results

4.4.1. Tagging Order

The generally favorable influence of greater tagging orders on condition of sufficient training material is commonly known and can be exploited by the tagging algorithm. Please note that the training order must be greater than or equal to the tagging order:

```
synther -c -n 3

synther -t -v -pr -n 2
```

results in 174 errors (utilizing the above-mentioned files)

```
synther -t -v -pr -n 3
```

causes only 149 mistakes.

4.4.2. Floor Value

Originally the floor value has been an estimated probability assigned to every transition that had not been seen during the training, since experience shows the favorable effect of this measure. A more successful method is the automatic reduction of the current order if the transition is assessed at the estimated likelihood zero, if necessary until the order 1 (occurrence probability). Each step back is punished by multiplying the found likelihood by a value less than one. Because the internal representation of the probabilities resp. frequencies is logarithmic on account of the computing time advantage of utilizing additions instead of multiplications, the factor less than one is replaced by an addend less than zero. This addend we still designate as floor value, and its standard value -1000 is changeable by means of the -fv option:

```
synther -c -f

synther -t -v -pr -f
```

results in 24 faulty numerals, whereas

```
synther -t -v -pr -f -fv -310
```

causes only 18 mistakes.

4.4.3. Iterations
Heuristic experiments have shown that the iterative method explained below generally leads to recognition improvements: In the first place we execute a training on the basis of hand-tagged material (cf. 2.). If this material is not extensive enough or we want to use the created statistic in a different application domain, we can realize the tagging with the help of an untagged text of the target domain, take the (undoubtedly faulty) tagging result and make a new training, this time being based on the original training set as well as the just automatically created one. This procedure can be iterated; in doing so more passes do not inevitably give rise to a recognition improvement. Therefore the mode of operation always should be heuristically checked.

4.4.3.1. The First Iteration
For the present our experiment is to be prepared by copying the example files to their standard places:

```
cp puffer/compareMTP.txt user/stat_in.txt
(the training set: 384 sentences of the MTP corpus)

cp puffer/inputFAZ219.txt user/input.txt
cp puffer/tokenFAZ219.txt user/token.txt
cp puffer/compareFAZ219.txt user/compare.txt
(the test set: 219 sentences of the FAZ corpus)

cp puffer/tokenFAZ384.txt user/iter_token.txt
(the domain-related iteration tokenizer output file: 384 sentences of the
FAZ corpus different from that above)
```

Now we can make a training pass without iterating

```
synther -c
```

and check the current tagging result

```
synther -t -v -pr
```

We obtain 292 mistakes. Now we can realize our first iteration (-it activates the iterative mode) utilizing the iteration token output file (-to)

```
synther -it -to user/iter_token.txt

synther -t -v -pr
```

shows a little improvement to 288 errors.

4.4.3.2. Two Variants to Raise the Number of Iterations
To determine the recognition result after the second iteration there are two possibilities: Either we extend the statistics of the first one by iterating once more

```
synther -it -to user/iter_token.txt -si user/iter_stat_in.txt
(The file user/iter_stat_in.txt was written during the first iteration.  It
contains  both  the original training set and the results of the first tag-
ging; cp. 4.4.3. This file's location could have been  specified  by  means
of the -io option.)

synther -t -v -pr

causes 287 mistakes;

or we start a new computation and demand two passes with the help of -in:

synther -it -to user/iter_token.txt -in 2 -pss
(-pss merely serves to document the computing status.)

synther -t -v -pr

produces the same result as above.


4.4.3.3. Two Variants for Increasing the Iterative Method's Effectiveness

4.4.3.3.1. Utilizing a Greater Training and Iteration Order than during the
           Test
Repeating the instance of 4.4.1.

synther -c -n 3
synther -t -v -pr -n 2

we  get  292 recognition errors. The raising of the training order does not
have an effect, since the sole demand is that the test order must  be  less
than the former one.
However  now we check the influence of this order combination regarding the
iterative method: Ten iterations using n = 2

synther -it -to user/iter_token.txt -in 10 -n 2 -pss
synther -t -v -pr -n 2

result in 280 mistakes, whereas n = 3 while training and n = 2 for the test

synther -it -to user/iter_token.txt -in 10 -n 3 -pss
synther -t -v -pr -n 2

causes merely 247 errors.

4.4.3.3.2. Iterating with the Test Set
It is true that the logic consequence from the demand for most possible do-
main-similarity  of  iteration  and test set is their identity, but we must
take into account the considerable additional computing expenditure in  the
test phase. Each iteration costs at least one more tagging, its acceptabil-
ity  as for instance in real-time-able speech processing systems depends on
the processor resources.
```

```
cp puffer/compareMTP.txt user/stat_in.txt
(the training set: 384 sentences of the MTP corpus)

cp puffer/token219.txt user/token.txt
cp puffer/input219.txt user/input.txt
cp puffer/compare219.txt user/compare.txt
(the test set: 219 sentences of an internet corpus)

Without iterating

synther -c -n 3
synther -t -v -pr -n 3

we produce 314 mistakes. After five iterations

synther -it -in 5 -n 3 -pss
synther -t -v -pr -n 3

this number is decreased extremely (by 86 to 228 errors).

4.5. Some (Compatibility) Utilities

4.5.1. Block-oriented Tagging
Some special applications of synther (e.g. end-of-sentence-detection) do
not contain the SI symbol or they require constant distances (blocks) in
which the Viterbi Algorithm (the calculation rule to determine the most
probable path through the symbol network) starts its backtracking. This
purpose serves the -b option that forces block-oriented tagging. -bl de-
fines the block length:

synther -t -b -bl 20

4.5.2. Pure Output
If the output file is to contain nothing but the tagging results use the
pure option:

synther -t -pure

4.5.3. Synchronization Conflicts between the Tagging Result and the Compar-
       ison File
Because of the automatically estimated ends of sentence by scanning for
special characters (cf. 3.1.) occasionally there are differences between
the position and number of the SI symbols of the tagging output and the
comparison file. In this case we can prevent the output of the SI symbols,
thus we must merely adapt the comparison file, for instance by

mv user/compare.txt user/compare1.txt
grep -v SI user/compare1.txt > user/compare.txt
rm user/compare1.txt

and can execute the synchronized synther variant:

synther -t -SI -v -pr
```

```
4.5.4. Number of Tagged Sentences
A sometimes helpful tool is the -ts parameter that defines the maximum num-
ber of tagged sentences resp. blocks (comparable to the -s option;
cf. 2.6.):

synther -t -ts 10 -ps

5. Complete Parameter List
--------------------------
-?          print this help
-b          block-oriented instead of sentence-oriented tagging
-bl bl      block length (condition: -b); standard: bl=10
-c          create n-gram statistics
-ca         use category list for adaptation to another category set
-cl cl      category list file for use in case of -ca; standard: cl=
            user/catlist.txt
-comp comp  comparison file for verification of hand-tagged material with
            the synther-generated tagging output; standard: comp=
            user/compare.txt
-f          include features for building statistics, tagging and verifying
-fv fv      floor value; it has to be a negative integer; standard: fv=-1000
-h          print this help
-help       print this help
-i i        tagger input file; standard: i=user/input.txt
-in in      number of iterations (condition: -it); standard: in=1
-io io      training material file utilized by iteration; standard: io=
            user/iter_stat_in.txt
-it         iterates the n-gram training
-l l        dictionary to be used to simulate the tokenizer; standard: l=
            user/lex.txt
-n n        n-gram order for building statistics and tagging; standard: n=2
-o o        tagger output file; standard: o=user/output.txt
-p          print the n-gram statistics results after building
-pr         print recognition results
-ps         print computing status
-pss        print short computing status
-pure       merely write the tagging results into the output file
-s s        number of sentences to be used for building the n-gram statis-
            tics; standard: s=all
-si si      n-gram statistics input file; standard: si=user/stat_in.txt
-SI         do not separate sentences of the tagging output file by the SI
            symbol
-st st      binary n-gram statistics file; standard: si=user/stat.bin
-sy sy      symbol table file; standard: sy=user/symtab.txt
-t          execute tagging
-to to      tokenizer output file; standard: to=user/token.txt
-token      simulate tokenizer
-ts ts      number of sentences resp. blocks to be tagged; standard: ts=all
-v          verification of the tagging results
-x x        maximum number of different symbols; standard: x=1000

written 2002-02-17 by David Sündermann
```

Bibliography

[Ko64] E. Koelwel; H. Ludwig: Gepflegtes Deutsch. Verlag Enzyklopädie, Leipzig 1964

[Or49] G. Orwell: Nineteen Eighty-Four. Martin Secker & Warburg Ltd, London 1949

Mathematical Foundations and Information Technology

[Br96] I. N. Bronstein, K. A. Semendjajew: Taschenbuch der Mathematik. B. G. Teubner Verlag, Leipzig 1996

[Ho98] R. Hoffmann: Signalanalyse und -erkennung. Springer-Verlag, Berlin-Heidelberg 1998

[Le02] E. Leopold: Das Zipfsche Gesetz. Fraunhofer-Institut für Autonome intelligente Systeme, Sankt Augustin 2002

[Ni00] `http://hissa.nist.gov/dads/` (National Institute of Standards and Technology)

[Sü01] D. Sündermann: Design and Development of General Symbol Statistics. Study Work. TU Dresden 2001

[Ve96] K. Vetters: Formeln und Fakten. B. G. Teubner, Leipzig 1996

Speech Processing and Tagging

[Be99] C. Becchetti, L. P. Ricotti: Speech Recognition, Theory and C++ Implementation. John Wiley & Sons Ltd, West Sussex 1999

[Cu92] D. Cutting, J. Kupiec, J. Pedersen, P. Sibun: A practical part-of-speech tagger. In: Proceedings of ANLP, Trento/Italy. 1992.

[De96] S. Deligne, F. Yvon, F. Bimbot: Introducing Statistical Dependencies and Structural Constraints in Variable-Length Sequence Models. ENST - Dept. Signal & Dept. Informatique, Paris 1996

[Dž00] S. Džeroski, T. Erjavec, J. Zavrel: Morphosyntactic Tagging of Slovene: Evaluating Taggers and Tagsets. Proceedings of the Second International Conference on Language Resources and Evaluation, Athens 2000

[Ha99] H.-U. Hain: Datengetriebene Vorgehensweise zur Disambiguierung linguistischer Kategorien und zur Satzendemarkierung. ESSV, Görlitz 1999

[Ha01] H. v. Halteren, J. Zavrel, W. Daelemans: Improving Accuracy in Word Class Tagging through the Combination of Machine Learning Systems. Association for Computational Linguistics, 2001

[Ro00] M. Rojc, J. Stergar, R. Wilhelm, H.-U. Hain, M. Holzapfel, B. Horvat: A Multilingual Text Processing Engine for the Papageno Text-to-Speech Synthesis System. Eurospeech, Budapest 1999

[Wi01] P. Witschel: Optimierte stochastische Sprachmodellierung auf linguistischen Klassen. Centrum für Informations- und Sprachverarbeitung der Universität München, Munich 2001