A taxonomy of sublinear multiple keyword pattern matching algorithms

Eindhoven University of Technology

Department of Mathematics and Computing Science

A taxonomy of sublineair multiple keyword pattern matching algorithms

95/13

ISSN 0926-4515

Computing Science Report 95/13
Eindhoven, April 1995

A taxonomy of sublinear multiple keyword pattern matching algorithms

B.W. Watson & G. Zwaan
Faculty of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB
Eindhoven, The Netherlands
email: `watson@win.tue.nl` or `wsinswan@win.tue.nl`

April 21, 1995

Abstract

This paper presents a taxonomy of sublinear keyword pattern matching algorithms related to the Boyer-Moore algorithm [BM77] and the Commentz-Walter algorithm [CW79a, CW79b]. The taxonomy includes, amongst others, the multiple keyword generalization of the single keyword Boyer-Moore algorithm and an algorithm by Fan and Su [FS93, FS94]. The corresponding precomputation algorithms are presented as well. The taxonomy is based on the idea of ordering algorithms according to their essential problem and algorithm details, and deriving all algorithms from a common starting point by successively adding these details in a correctness preserving way. This way of presentation not only provides a complete correctness argument of each algorithm, but also makes very clear what algorithms have in common (the details of their nearest common ancestor) and where they differ (the details added after their nearest common ancestor). Introduction of the notion of safe shift distances proves to be essential in the derivation and classification of the algorithms. Moreover, the paper provides a common derivation for and a uniform presentation of the precomputation algorithms, not yet found in the literature.

Contents

1 Introduction

The keyword (or string) pattern matching problem can informally be described as the problem of finding all occurrences of keywords (strings) from a given set as substrings in a given (input) string. This problem is encountered in many areas and in several forms. In computing science, for instance, it plays a role in text search/analysis, lexical analysis, and data processing. In biology it is encountered in the analysis of, amongst others, DNA sequences. The problem can also be generalized to the matching of regular expressions, tree patterns, and graph patterns, none of which is treated here.

The keyword pattern matching problem has been extensively studied and a multitude of diverse solutions/algorithms exists. Single keyword algorithms are for instance described by Knuth, Morris, and Pratt [KMP77] and Boyer and Moore [BM77]; multiple keyword algorithms by Aho and Corasick [AC75], by Commentz-Walter [CW79a, CW79b], and by Fan and Su [FS93, FS94]. An overview of keyword pattern matching algorithms can be found in [Aho90].

Due to the diversity of the algorithms and their descriptions—that tend to be rather involved and verbal— it is hard to get a good overview and to make a sound comparison between algorithms. In order to fulfill these needs a taxonomy of keyword pattern matching algorithms was presented by Watson and Zwaan in [WZ92, WZ93]. Here, we focus our attention on a part of that taxonomy containing a family of multiple keyword pattern matching algorithms that have a matching time that may be sublinear in the length of the input string (taking the number of symbol comparisons as a measure of matching time). Amongst others, it comprises the multiple keyword generalization of the single keyword Boyer-Moore algorithm [BM77], the Commentz-Walter algorithm [CW79a, CW79b], and the algorithm by Fan and Su [FS93, FS94] (only after deriving this algorithm we found its description by Fan and Su). Both the Boyer-Moore and Commentz-Walter algorithms provided the inspiration for our derivations and classifying principle.

The main results of this paper are comprised in the taxonomy graph shown in figure 1 (a more detailed description of the graph is found further on in this section). This taxonomy graph can be viewed as an alternative table of contents to this paper. It was obtained in order to meet the following goals:

- the systematic and formal derivation of the algorithms from a common starting point through a series of refinements to either algorithm or problem
- to factor out common portions of (the derivations of) well-known algorithms in order to facilitate the understanding of these algorithms and their comparison
- the presentation of the algorithms in a common framework to permit an easier comprehension of and a better comparison between the algorithms

It is the first concise and systematic presentation of and comparison between the algorithms from the family considered here. Another taxonomy of (single keyword) pattern matching algorithms by Hume and Sunday [HS91] does not meet the goals set in this paper since there any derivations and proofs of algorithms are missing.

Moreover, we show that all functions that need to be precomputed for the pattern matching algorithms of this family can in a simple way be expressed in a small set of base functions. The definitions of the base functions can all be written in forms satisfying one general pattern. From this general pattern a precomputation algorithm scheme has been derived that can be instantiated for each base function to yield a precomputation algorithm for that function (sometimes, the resulting algorithm can be simplified and/or some trivial postprocessing has to be added to it). Hence, the precomputation algorithms can also be derived and presented in a uniform way. This includes formal derivations of the precomputation algorithms for the Boyer-Moore algorithm, the Commentz-Walter algorithm and the algorithm presented by Fan and Su not yet found in the literature.

The taxonomy given here differs from the corresponding part of the taxonomy in [WZ92, WZ93] in that we here present the correct multiple keyword generalization of the Boyer-Moore algorithm

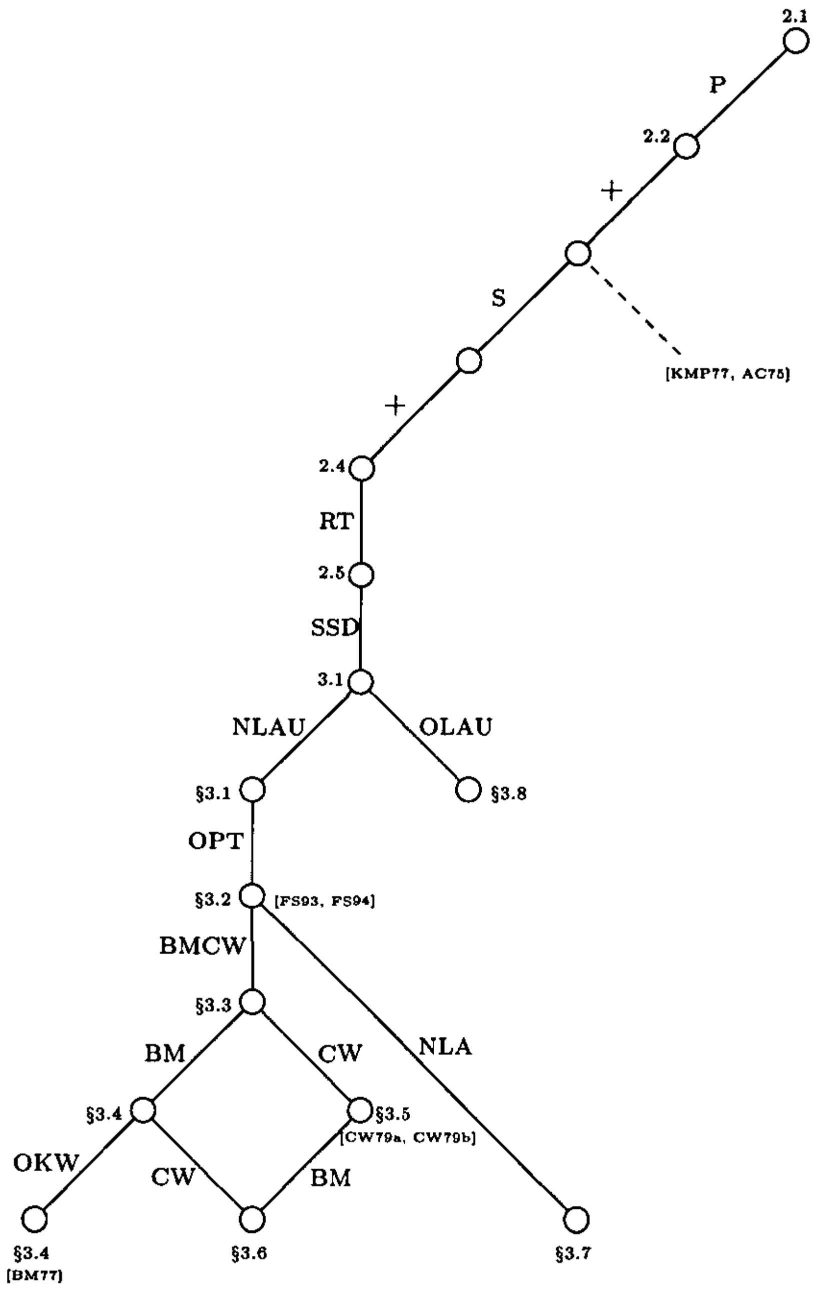

Figure 1: Taxonomy graph of keyword pattern matching algorithms from the Boyer-Moore family. Each vertex corresponds to an algorithm. Vertices are labelled with either an algorithm number or a subsection number preceded by § referring to an algorithm or a subsection in this paper. Some vertices are additionally labelled with literature references. Each edge corresponds to the addition of either a problem or algorithm detail (see appendix B for a complete list of details and their descriptions). The sequence of edge labels that are encountered when going from the top vertex to another vertex forms a characterization of the algorithm corresponding to that vertex. For instance, the algorithm of the vertex labelled §3.5 (the Commentz-Walter algorithm) is characterized by (P_+S_+,RT,SSD,NLAU,OPT,BMCW,CW). The dashed edge leading to [KMP77, AC75] indicates the path to the rest of the taxonomy presented in [WZ92, WZ93] that contains the Knuth-Morris-Pratt and the Aho-Corasick algorithms and is omitted in this paper.

and a common ancestor of this algorithm and the Commentz-Walter algorithm. Because of this the derivations and the structure of the taxonomy have changed. Moreover, we present a completely new derivation of the precomputation algorithms.

An implementation as a C procedural library of almost all algorithms from [WZ92] (called "Eindhoven Pattern Kit") and an analysis of their performance is given in [Wat94] (the implementation is in a somewhat rudimentary form being meant for the benchmarking only). Only the Commentz-Walter algorithm and a common descendant of both the Boyer-Moore and Commentz-Walter algorithm are implemented and discussed there. The performance results for these algorithms conform with the qualitative predictions made here. An implementation of all algorithms from this paper and from [WZ92] as a C++ class library is called "SPARE Parts: A C++ toolkit for String PAttern REcognition" and will be available at URL `ftp://ftp.win.tue.nl /pub/techreports/pi/pattm/spare/`. This implementation is entirely based on the abstract algorithms described in this paper—in fact it is a systematic translation of them (this in contrast to for instance the implementations given in [HS91]).

1.1 Basic algorithm and derivation principles

The algorithms in this family traverse the input string in a direction that is opposite to the direction in which keyword symbols are matched to symbols in the input string. In this paper we choose to inspect suffixes of prefixes of the input string both in order of increasing length. This algorithm will be the starting point of all further derivations. Choosing such a basic algorithm we have the possibility to attain matching times that are sublinear in the length of the input string (i.e. not all symbols of the input string are inspected). It is achieved by taking steps through the input string of which the length is determined by a shift function based on the information of the last matching attempt and possibly on additional information. The example in figure 2 illustrates this principle. Notice that not all symbols of the input string are scanned, although it is possible that some symbols of the input string are scanned more than once. The most simple

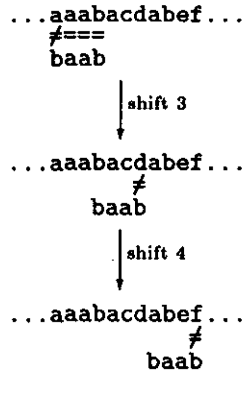

Figure 2: Example of larger shift distances

shift function is the function that always yields 1. However, one can imagine larger shifts being possible. Ideally, such a shift would take us to the next occurrence of a match, but then calculating the value of the shift function is equivalent to the pattern matching problem itself. Therefore, we strive for shift functions that are easier to calculate and that do not exceed the ideal shift (called *safe shift functions*) i.e. we aim at approximations of the ideal shift from below. The ideal shift function is the minimum over a domain characterized by some predicate. We derive various approximations from below by systematically weakening this predicate and derived predicates, by applying rules for minimum and maximum over disjunctive or conjunctive domains, and by enlarging domains. Considerations that play a role in these derivations are, for instance, whether or not to look ahead at symbols of the unscanned part of the input string, what information to use on the last scanned (non-matching) symbol, and the extent to which this information is coupled with the information on the recognized suffix. Thus, we obtain several shift functions that meet the aforementioned requirements leading to an equal number of algorithms amongst which are the well-known Boyer-Moore and Commentz-Walter algorithms. The most simple weakening of the predicate is the weakening to the predicate **true** yielding the shift function that is always equal to 1. The reason to derive ever smaller shift functions is that a smaller shift functions usually takes less precomputation time and less storage space (for instance, a one dimensional table instead of a two dimensional table).

The techniques we use to systematically derive shift functions from the ideal shift function enable us to clearly delineate the relations between the resulting shift functions and algorithms. Furthermore, they may be used in the derivation of yet other members of this family. This truly systematic approach, especially the predicate weakening technique, is not known from literature. Among other things we derive that the Commentz-Walter algorithm [CW79a, CW79b] is not the multiple keyword generalization of the Boyer-Moore algorithm [BM77]. In fact, we show that the algorithms are incomparable (meaning that the shift distance in one algorithm is not always at least the shift distance in the other) and that they have a faster common ancestor that combines the properties of both. As it is, this common ancestor is derived first and subsequently the Boyer-Moore algorithm and the Commentz-Walter algorithm are derived from it. Both algorithms also have a common descendant (the algorithm that was incorrectly identified as the Boyer-Moore algorithm in [WZ92, WZ93]). Furthermore, it is shown that the algorithm described by Fan and Su [FS93, FS94] is an even faster ancestor of the common ancestor of the Boyer-Moore and Commentz-Walter algorithms.

1.2 The taxonomy

The algorithms are derived from a common starting point by successively adding either algorithm or problem details (see appendix B for a complete list of details and their descriptions). The only problem detail considered here is the restriction to the one keyword case. Among the algorithm details considered are restriction of nondeterminacy, introduction of the reverse trie, introduction of a shift function, and choice of a particular shift function. Each addition of an algorithm detail gives a new algorithm satisfying the same specification as the original algorithm; thus, the correctness of the algorithms is preserved. The sequence of details introduced in the derivation of an algorithm can be used to identify its similarities and its differences with other algorithms. These ordered detail sequences are used to identify the algorithms and to give a taxonomy of the algorithms in this family. The taxonomy is depicted as a graph in figure 1. Our method of developing a taxonomy was inspired by the method described by Jonkers [Jon83]. There it is applied to develop a taxonomy of garbage collection algorithms. The method is also applied to attribute evaluation algorithms by Marcelis [Mar90]. Other examples of algorithm taxonomies are found in [Bro83, Dar78].

All algorithms are presented in a somewhat extended version of the guarded command language of Dijkstra [Dij76] in order to avoid the peculiarities of any particular programming language. The algorithms use string type variables in order to abstract from any implementation detail like, for instance, indexing. Derivation of the algorithms is done a calculational way following Dijkstra's style of program derivation.

1.3 Overview

In section 2 we give a formal definition of the pattern matching problem. From a trivial solution to this problem we derive, by addition of a number of algorithm details, an algorithm that is the starting point for the derivation of the algorithms in section 3. In section 3 we start by adding the algorithm detail that states that shifts larger than one may be possible. Addition of this program detail accounts for the possible sublinear matching time of all of the algorithms to be derived in this section. Subsequently, we derive by systematic approximation from below of the maximal safe shift distance the various algorithms of this family (in order of decreasing matching speed). The definitions of all functions introduced in section 3 are rewritten in forms according to a general pattern in section 4. From the general pattern a precomputation algorithm scheme is derived that can be instantiated for any of the functions. Section 5 contains the conclusions. Appendix A contains definitions and properties used throughout this paper. Appendix B contains a complete list of all algorithm and problem details and their descriptions.

2 The problem and some naive solutions

The keyword pattern matching problem is to find all occurrences of keywords from a set as substrings in an input string. Formally, given an alphabet V (a non-empty finite set of symbols), an input string $S \in V^*$, and a finite non-empty pattern set $P \subseteq V^*$, establish

$$R:\ O = \left(\cup\ l,v,r : lvr = S : \{l\} \times (\{v\} \cap P) \times \{r\}\right).$$

Throughout this paper we will adopt the convention that, unless stated otherwise, program variables and bound variables with names from the beginning of the Latin alphabet (i.e. a, b, c) will range over V, while variables with names from the end of the Latin alphabet (i.e. l, q, r, u, v, w) will range over V^*. A trivial (but unrealistic) solution to the problem is

Algorithm 2.1()

```
$O := \left(\cup\ l,v,r : lvr = S : \{l\} \times (\{v\} \cap P) \times \{r\}\right)$
$\{ R \}$
```

The sequence of details describing this algorithm is the empty sequence (sequences of details are introduced in subsection 1.2 and figure 1).

There are two basic directions in which to proceed while developing naive algorithms to solve this problem. Informally, a substring of S can be considered a "suffix of a prefix of S" or a "prefix of a suffix of S". Only the first possibility is considered here, since the second possibility only leads to algorithms that are the mirror images of algorithms obtained by following the first possibility (basically, it amounts to reversing all strings in the problem). Moreover, this is the way that the Boyer-Moore, Commentz-Walter, and Fan and Su algorithms treat substrings of input string S.

Formally, we can consider "suffixes of prefixes of S" as follows:

$$\begin{array}{ll}
 & \left(\cup\ l,v,r : lvr = S : \{l\} \times (\{v\} \cap P) \times \{r\}\right) \\
= & \quad \{\text{introduce } u : u = lv\} \\
 & \left(\cup\ l,v,r,u : ur = S \wedge lv = u : \{l\} \times (\{v\} \cap P) \times \{r\}\right) \\
= & \quad \{l, v \text{ only occur in the latter range conjunct, so restrict their scope}\} \\
 & \left(\cup\ u,r : ur = S : \left(\cup\ l,v : lv = u : \{l\} \times (\{v\} \cap P) \times \{r\}\right)\right)
\end{array}$$

A simple non-deterministic algorithm is obtained by applying "examine prefixes of a given string in any order" (algorithm detail (P)) to input string S. In the following algorithm we use a **for-rof** statement in order to express its non-determinism. Statement **for** $x : P \rightarrow S$ **rof** amounts to executing statement list S once for each value of x that satisfies P initially, assuming only a finite number of such values exist. The order in which the values of x are chosen is arbitrary. It results in

Algorithm 2.2(P)

```
$O := \varnothing$;
**for** $(u, r) : ur = S \rightarrow$
    $O := O \cup \left(\cup\ l,v : lv = u : \{l\} \times (\{v\} \cap P) \times \{r\}\right)$
**rof**$\{ R \}$
```

The update of O (with another quantifier) in the inner repetitions of algorithm (P) can be computed with another non-deterministic repetition. This inner repetition would consider suffixes of u. Thus by applying "examine suffixes of a given string in any order" (algorithm detail (S)) to string u we obtain algorithm

Algorithm 2.3(PS)

```
$O := \varnothing$;
**for** $(u,r) : ur = S \rightarrow$
    **for** $(l,v) : lv = u \rightarrow$
        $O := O \cup \{l\} \times (\{v\} \cap P) \times \{r\}$
    **rof**
**rof**$\{ R \}$
```

Algorithm (PS) consists of two nested non-deterministic repetitions. In each case, the repetition can be made deterministic by considering prefixes (or suffixes as the case is) in increasing (called detail (+)) or decreasing (detail (−)) order of length. This gives two binary choices. Since the Boyer-Moore and Commentz-Walter algorithms examine string S from left to right and the patterns in P from right to left we focus our attention on (the operators $\rceil$, $\rfloor$, $\lceil$, and $\lfloor$ are defined in definition A.1; relation $\leq_p$ is defined in definition A.3):

Algorithm 2.4($\mathrm{P_+S_+}$)

```
$u, r := \varepsilon, S$; $O := \{\varepsilon\} \times (\{\varepsilon\} \cap P) \times \{S\}$;
$\{$ invariant: $O = (\cup\ x,y,z : xyz = S \wedge xy \leq_p u : \{x\} \times (\{y\} \cap P) \times \{z\})$ $\}$
**do** $r \neq \varepsilon \rightarrow$
    $u, r := u(r\rceil 1), r\rfloor 1$; $l, v := u, \varepsilon$; $O := O \cup \{u\} \times (\{\varepsilon\} \cap P) \times \{r\}$;
    **do** $l \neq \varepsilon \rightarrow$
        $l, v := l\lfloor 1, (l\lceil 1)v$;
        $O := O \cup \{l\} \times (\{v\} \cap P) \times \{r\}$
    **od**
**od**$\{ R \}$
```

This algorithm has running time $\mathcal{O}(|S|^2)$, assuming that intersection with P is a $\mathcal{O}(1)$ operation. We will now improve the running time of this algorithm. Consider the set of suffixes of keywords **suff**(P) (the functions **suff** and **pref** are defined in definition A.2). A string w is an element of **suff**(P) if and only if it can be extended on the left to a pattern in P, i.e. $(\exists w' : w' \in V^* : w'w \in P)$. It follows that if $w \notin \mathbf{suff}(P)$ any extension of w on the left is not an element of **suff**(P) either. Consequently, the inner repetition in algorithm 2.4 can terminate as soon as $(l\lceil 1)v \notin \mathbf{suff}(P)$ holds, since then all suffixes of u that are equal to or longer than $(l\lceil 1)v$ are not in **suff**(P) and hence not in P. The inner repetition guard is therefore strengthened to

$$l \neq \varepsilon \ \mathbf{cand}\ (l\lceil 1)v \in \mathbf{suff}(P).$$

Observe that $v \in \mathbf{suff}(P)$ is an now invariant of the inner repetition. This invariant is initially established by the assignment $v := \varepsilon$ since $P \neq \varnothing$ and thus $\varepsilon \in \mathbf{suff}(P)$. Direct evaluation of $(l\lceil 1)v \in \mathbf{suff}(P)$ is expensive. Therefore, it is done using the transition function τ_P of the reverse trie [Fre60] corresponding to P where $\tau_P : \mathbf{suff}(P) \times V \rightarrow \mathbf{suff}(P) \cup \{\perp\}$ is defined by

$$\tau_P(w,a) = \begin{cases} aw & \text{if } aw \in \mathbf{suff}(P) \\ \perp & \text{if } aw \notin \mathbf{suff}(P) \end{cases} \qquad (w \in \mathbf{suff}(P), a \in V).$$

(algorithm detail (RT)). Since we usually refer to the trie corresponding to P we will write τ instead of τ_P. Transition function τ can be computed beforehand. Its precomputation is discussed in subsection 4.1. The guard becomes $l \neq \varepsilon$ **cand** $\tau(v, l\lceil 1) \neq \perp$ yielding algorithm

Algorithm 2.5(P+S+,RT)

```
$u, r := \varepsilon, S$; $O := \{\varepsilon\} \times (\{\varepsilon\} \cap P) \times \{S\}$;
{ invariant: $O = (\cup\, x, y, z : xyz = S \wedge xy \leq_p u : \{x\} \times (\{y\} \cap P) \times \{z\})$ }
do $r \neq \varepsilon \rightarrow$
    $u, r := u(r\upharpoonright 1), r\downharpoonright 1$;
    $l, v := u, \varepsilon$; $O := O \cup \{u\} \times (\{\varepsilon\} \cap P) \times \{r\}$;
    do $l \neq \varepsilon$ cand $\tau(v, l\upharpoonright 1) \neq \perp \rightarrow$
        $l, v := l\lfloor 1, (l\upharpoonright 1)v$;
        $O := O \cup \{l\} \times (\{v\} \cap P) \times \{r\}$
    od
    { $u = lv \wedge v \in \mathbf{suff}(P) \wedge (l = \varepsilon$ cor $(l\upharpoonright 1)v \notin \mathbf{suff}(P))$ }
od{ $R$ }
```

This algorithm has $\mathcal{O}(|S| \cdot (\mathbf{MAX}p : p \in P : |p|))$ running time. It will serve as a starting point for the derivation of all algorithms in the following section.

3 Sublinear pattern matching algorithms

In this section we derive sublinear pattern matching algorithms starting with algorithm 2.5 by exploring the possibility of safely (without missing matches) making shifts of more than one symbol, i.e. replacing assignment $u, r := u(r\upharpoonright 1), r\downharpoonright 1$ by assignment $u, r := u(r\upharpoonright k), r\downharpoonright k$ for some k satisfying

$$1 \leq k \leq (\mathbf{MIN}\, n : 1 \leq n \wedge \mathbf{suff}(u(r\upharpoonright n)) \cap P \neq \varnothing : n)$$

(algorithm detail (SSD) (Safe Shift Distance)). The upperbound is the distance to the next match, the maximal safe shift distance. A number k satisfying this condition is called a *safe shift distance*. Since computing the upperbound on k is essentially the same as the problem we are trying to solve we aim at easier to compute approximations from below of the upperbound. These are derived by systematically weakening the predicate $\mathbf{suff}(u(r\upharpoonright n)) \cap P \neq \varnothing$ in the range of the upperbound, weakening resulting predicates, applying rules for minimum and maximum over a disjunctive or conjunctive domain, and enlarging domains. Considerations that play a role in these derivations are, for instance, whether or not to look ahead at symbols of the unscanned part of the input string, whether to use or not to use information on the last scanned symbol (usually a non-matching symbol), and the extent to which this information is coupled with the information on the recognized suffix. Thus, several algorithms are obtained amongst which the generalized version of the Boyer-Moore algorithm [BM77], the Commentz-Walter algorithm [CW79a, CW79b], and the algorithm by Fan and Su [FS93, FS94]. We derive ever smaller shift functions since the smaller the shift function the less precomputation time and storage space for the functions constituting the shift function is usually needed. For instance, the algorithm by Fan and Su [FS93, FS94] is faster than the Commentz-Walter algorithm [CW79a, CW79b], but it needs a two dimensional table to store one of the functions constituting its shift function whereas the Commentz-Walter algorithm only needs one dimensional tables.

In the derivations we use part of the postcondition of the inner repetition in algorithm 2.5 ($u = lv \wedge v \in \mathbf{suff}(P)$). Adding $l, v := \varepsilon, \varepsilon$ to the initial assignments in algorithm 2.5 turns $u = lv \wedge v \in \mathbf{suff}(P)$ into an invariant of the outer repetition. Due to the dependence of the upperbound on l, v, and r we will aim at shift functions k that depend on l, v, and r and write $k(l,v,r)$. Hence, we arrive at the following algorithm scheme for all algorithms to be derived in this section:

Algorithm 3.1(P$_+$S$_+$,RT,SSD)

```
$u, r := \varepsilon, S$;  $O := \{\varepsilon\} \times (\{\varepsilon\} \cap P) \times \{S\}$;  $l, v := \varepsilon, \varepsilon$;
{ invariant: $O = (\cup\ x, y, z : xyz = S \wedge xy \leq_p u : \{x\} \times (\{y\} \cap P) \times \{z\})$
            $\wedge\ u = lv \wedge v \in \mathbf{suff}(P) \wedge (l = \varepsilon\ \mathbf{cor}\ (l\lceil 1)v \notin \mathbf{suff}(P))$ }
do $r \neq \varepsilon \rightarrow$
    $u, r := u(r\rceil k(l,v,r)), r\rfloor k(l,v,r)$;
    $l, v := u, \varepsilon$;  $O := O \cup \{u\} \times (\{\varepsilon\} \cap P) \times \{r\}$;
    do $l \neq \varepsilon$ cand $\tau(v, l\lceil 1) \neq \bot \rightarrow$
        $l, v := l\lfloor 1, (l\lceil 1)v$;
        $O := O \cup \{l\} \times (\{v\} \cap P) \times \{r\}$
    od
od{ $R$ }
```

Particular algorithms are obtained by substituting their shift functions for $k(l,v,r)$. Such a substitution may not always yield an algorithm that exactly corresponds with its original description in the literature; sometimes an additional transformation of the resulting algorithm is needed (for instance, a phase shift of the repetition; see [WZ92] for a phase shifted version of the algorithm scheme). Conjunct $l = \varepsilon$ **cor** $(l\lceil 1)v \notin \mathbf{suff}(P)$ is added to the invariant in order to stress that provided l is nonempty symbol $l\lceil 1$ is non-matching.

3.1 No lookahead at the unscanned part of the input string

In this subsection we derive an approximation from below of the upperbound on k that does not depend on r and that will be a starting point of most of our further derivations. In terms of algorithms this means that we refrain from looking ahead at the symbols of r, the yet unscanned part of the input string (algorithm detail (NLAU) (No LookAhead at Unscanned part of the input string)). This is in accordance with most of the algorithms we are aiming at. One symbol lookahead at the unscanned part of the input string is discussed in subsection 3.8. We derive

$$
\begin{aligned}
& (\mathbf{MIN}\, n : 1 \leq n \wedge \mathbf{suff}(u(r\rceil n)) \cap P \neq \varnothing : n) \\
= \quad & \quad \{ \text{domain split, } r\rceil n = r \text{ if } n \geq |r| \} \\
& (\mathbf{MIN}\, n : 1 \leq n \leq |r| \wedge \mathbf{suff}(u(r\rceil n)) \cap P \neq \varnothing : n) \\
& \mathbf{min}\, (\mathbf{MIN}\, n : |r| < n \wedge \mathbf{suff}(ur) \cap P \neq \varnothing : n) \\
\geq \quad & \quad \{ 1 \leq n \leq |r| : r\rceil n \in V^n, \text{ monotonicity of } \mathbf{suff} \text{ and } \cap \} \\
& (\mathbf{MIN}\, n : 1 \leq n \leq |r| \wedge \mathbf{suff}(uV^n) \cap P \neq \varnothing : n) \\
& \mathbf{min}\, (\mathbf{MIN}\, n : |r| < n \wedge \mathbf{suff}(ur) \cap P \neq \varnothing : n) \\
= \quad & \quad \{ r \in V^{|r|}, \text{ monotonicity of } \mathbf{suff} \text{ and } \cap,\ \mathbf{suff}(ur) \cap P \neq \varnothing \Rightarrow \mathbf{suff}(uV^{|r|}) \cap P \neq \varnothing \} \\
& (\mathbf{MIN}\, n : 1 \leq n \leq |r| \wedge \mathbf{suff}(uV^n) \cap P \neq \varnothing : n) \\
\geq \quad & \quad \{ \text{enlarging domain} \} \\
& (\mathbf{MIN}\, n : 1 \leq n \wedge \mathbf{suff}(uV^n) \cap P \neq \varnothing : n)
\end{aligned}
$$

Since the last formula is to be the starting point of our further derivations we will from here on aim at shift functions k being dependent only on u, i.e. on l and v (remember $u = lv \wedge v \in \mathbf{suff}(P)$). We will write $k(l,v)$ instead of $k(l,v,r)$.

3.2 Restriction to one symbol lookahead

In all derivations in this subsection and the following subsections we assume

$$u = lv \wedge v \in \mathbf{suff}(P).$$

Restriction to one symbol lookahead ($l{\restriction}1$, the last symbol of u scanned in the inner loop) leads to the algorithm by Fan and Su [FS93, FS94]. It is obtained by weakening the predicate in the approximation of the upperbound in subsection 3.1 in the following way:

$$\begin{array}{ll}
 & \mathbf{suff}(uV^n) \cap P \neq \varnothing \\
= & \quad \{\, u = lv \,\} \\
 & \mathbf{suff}(lvV^n) \cap P \neq \varnothing \\
\Rightarrow & \quad \{\, l = (l{\downharpoonright}1)(l{\restriction}1),\ l{\downharpoonright}1 \in V^*, \text{ monotonicity of } \mathbf{suff} \text{ and } \cap \,\} \\
 & \mathbf{suff}(V^*(l{\restriction}1)vV^n) \cap P \neq \varnothing \\
= & \quad \{\, \text{property A.4} \,\} \\
 & V^*(l{\restriction}1)vV^n \cap V^*P \neq \varnothing \\
= & \quad \{\, l = \varepsilon\text{: property A.5.i; } l \neq \varepsilon\text{: property A.5.ii} \,\} \\
 & V^*(l{\restriction}1)vV^n \cap P \neq \varnothing \vee vV^n \cap V^*P \neq \varnothing
\end{array}$$

Notice that we have obtained a weaker predicate solely by discarding any information on $l{\downharpoonright}1$. The only information on l that is still taken into account is $l{\restriction}1$ being either empty or consisting of one symbol. In the latter case we say to have one symbol lookahead. Observe that the symbol is the last symbol of u scanned in the inner loop and that it is a non-matching symbol. After substituting the weaker predicate we obtain shift distance $k_{opt}(l,v)$ where $k_{opt} \in V^* \times \mathbf{suff}(P) \to \mathbb{N}$ is defined for $x \in V^*$ and $y \in \mathbf{suff}(P)$ by

$$k_{opt}(x,y) = (\mathbf{MIN}\, n : n \geq 1 \wedge (V^*(x{\restriction}1)yV^n \cap P \neq \varnothing \vee yV^n \cap V^*P \neq \varnothing) : n)\,.$$

Function k_{opt} can be expressed as follows

$$k_{opt}(x,y) = \begin{cases} d_{opt}(x{\restriction}1, y) \,\mathbf{min}\, d_{sp}(y) & x \neq \varepsilon \\ d_i(y) \,\mathbf{min}\, d_{sp}(y) & x = \varepsilon \end{cases}$$

where $d_{opt} \in V \times \mathbf{suff}(P) \to \mathbb{N}$ is defined by

$$d_{opt}(a,y) = (\mathbf{MIN}\, n : n \geq 1 \wedge V^*ayV^n \cap P \neq \varnothing : n) \qquad (a \in V, y \in \mathbf{suff}(P)),$$

$d_{sp} \in \mathbf{suff}(P) \to \mathbb{N}$ is defined by

$$d_{sp}(y) = (\mathbf{MIN}\, n : n \geq 1 \wedge yV^n \cap V^*P \neq \varnothing : n) \qquad (y \in \mathbf{suff}(P)),$$

(function d_2 in [CW79a, CW79b]), and $d_i \in \mathbf{suff}(P) \to \mathbb{N}$ is defined by

$$d_i(y) = (\mathbf{MIN}\, n : n \geq 1 \wedge V^*yV^n \cap P \neq \varnothing : n) \qquad (y \in \mathbf{suff}(P)),$$

(function d_1 in [CW79a, CW79b]). Functions d_{opt} and d_i account for occurrences of ay and y, respectively, within some keyword (i.e. as infix of some keyword), whereas function d_{sp} accounts for occurrences of suffixes of y as proper prefixes of some keyword. Precomputation of d_{opt}, d_{sp}, and d_i is discussed in subsection 4.2.

Calculating the shift distance in this way is referred to as algorithm detail (OPT) and results in algorithm (P+S+,RT,SSD,NLAU,OPT). We arrived at this algorithm not knowing it had already been described by Fan and Su in [FS93, FS94]. From their informal description it undoubtedly follows that they describe the same algorithm though their formal treatment of the algorithm and, especially, the precomputation is rather involved. Finally, notice that to store function d_{opt} one needs a two dimensional table, whereas functions d_i and d_{sp} only need one dimensional tables. In the following subsections we derive shift functions smaller than k_{opt} that are expressed solely in functions needing one dimensional tables for storage.

3.3 Lookahead symbol is mismatching

We derive an approximation from below of d_{opt} that yields an algorithm that is the common ancestor of the multiple keyword generalization of the Boyer-Moore algorithm [BM77] and the Commentz-Walter algorithm [CW79a, CW79b]. Essentially, the resulting shift function is not based on the identity of the lookahead symbol $l{\restriction}1$ but only uses the fact that the lookahead symbol is mismatching, as is done in the Boyer-Moore shift function. In this way one might say that the recognized suffix and the (mismatching) lookahead symbol have to some extent been decoupled.

We start by weakening the predicate from d_{opt}. Assume $l \neq \varepsilon$ and $(l{\restriction}1)v \notin \mathbf{suff}(P)$. We derive

$$
\begin{array}{ll}
 & V^*(l{\restriction}1)vV^n \cap P \neq \varnothing \\
= & \{ v \in V^{|v|}, \text{ monotonicity of } \cap \} \\
 & V^*(l{\restriction}1)V^{|v|+n} \cap P \neq \varnothing \wedge V^*(l{\restriction}1)vV^n \cap P \neq \varnothing \\
\Rightarrow & \{ (l{\restriction}1)v \notin \mathbf{suff}(P), \text{ so } l{\restriction}1 \in \{ a \mid a \in V \wedge av \notin \mathbf{suff}(P) \}, \text{ definition } MS \} \\
 & V^*(l{\restriction}1)V^{|v|+n} \cap P \neq \varnothing \wedge V^*(V \setminus MS(v))vV^n \cap P \neq \varnothing
\end{array}
$$

where $MS \in \mathbf{suff}(P) \rightarrow V$ is defined by

$$MS(y) = \{ a \mid a \in V \wedge ay \in \mathbf{suff}(P) \} \qquad (y \in \mathbf{suff}(P)).$$

The first conjunct will lead to a shift component based on the identity of the lookahead symbol that is identical to a component of the Commentz-Walter shift function. The second conjunct will lead to a shift component—based on the recognized suffix and the fact that the lookahead symbol is mismatching—that is identical to a component of the Boyer-Moore shift function. Replacing the range predicate of d_{opt} by the last predicate of the preceding derivation we proceed

$$
\begin{array}{ll}
 & (\mathbf{MIN}\, n : n \geq 1 \wedge V^*(l{\restriction}1)V^{|v|+n} \cap P \neq \varnothing \wedge V^*(V \setminus MS(v))vV^n \cap P \neq \varnothing : n) \\
\geq & \{ \mathbf{MIN} \text{ with conjunctive range} \} \\
 & (\mathbf{MIN}\, n : n \geq 1 \wedge V^*(l{\restriction}1)V^{|v|+n} \cap P \neq \varnothing : n) \\
 & \mathbf{max}\, (\mathbf{MIN}\, n : n \geq 1 \wedge V^*(V \setminus MS(v))vV^n \cap P \neq \varnothing : n) \\
= & \{ \text{change of bound variable: } m = |v| + n; \text{ definition } d_{vi} \text{ (after derivation)} \} \\
 & (\mathbf{MIN}\, m : m \geq |v| + 1 \wedge V^*(l{\restriction}1)V^m \cap P \neq \varnothing : m - |v|)\, \mathbf{max}\, d_{vi}(v) \\
\geq & \{ \text{enlarging domain} \} \\
 & (\mathbf{MIN}\, m : m \geq 1 \wedge V^*(l{\restriction}1)V^m \cap P \neq \varnothing : m - |v|)\, \mathbf{max}\, d_{vi}(v) \\
= & \{ l \neq \varepsilon, \text{ definition of } char_{cw} \text{ (after derivation)} \} \\
 & char_{cw}(l{\restriction}1, |v|)\, \mathbf{max}\, d_{vi}(v)
\end{array}
$$

where $d_{vi} \in \mathbf{suff}(P) \rightarrow \mathbb{N}$ is defined by

$$d_{vi}(y) = (\mathbf{MIN}\, n : n \geq 1 \wedge V^*(V \setminus MS(v))vV^n \cap P \neq \varnothing : n) \qquad (y \in \mathbf{suff}(P))$$

and $char_{cw} \in V \times \mathbb{N} \rightarrow \mathbb{N}$ is defined by

$$char_{cw}(a, z) = (\mathbf{MIN}\, n : n \geq 1 \wedge V^*aV^n \cap P \neq \varnothing : n - z) \qquad (a \in V, z \in \mathbb{N}).$$

This results in shift distance $k_{bmcw}(l,v)$ where $k_{bmcw} \in V^* \times \mathbf{suff}(P) \rightarrow \mathbb{N}$ is defined by

$$
k_{bmcw}(x, y) = \begin{cases} (char_{cw}(x{\restriction}1, |y|)\, \mathbf{max}\, d_{vi}(y))\, \mathbf{min}\, d_{sp}(y) & (x \in V^+, y \in \mathbf{suff}(P)) \\ d_i(y)\, \mathbf{min}\, d_{sp}(y) & (x = \varepsilon, y \in \mathbf{suff}(P)). \end{cases}
$$

Precomputation of $char_{cw}$ is discussed in subsection 4.3 and of d_{vi}, d_{sp}, and d_i in subsection 4.2. Notice that we have

$$k_{opt}(x, y) \geq k_{bmcw}(x, y) \qquad (x \in V^*, y \in \mathbf{suff}(P)).$$

Approximation from below of k_{opt} by k_{bmcw} is referred to as algorithm detail (BMCW). We chose this name to reflect that essential ideas from both the Boyer-Moore and Commentz-Walter algorithms are introduced. In the next two subsections these algorithms are derived from the algorithm presented in this subsection and characterized by detail sequence (P_+S_+,RT,SSD,NLAU,OPT,BMCW).

3.4 The multiple keyword Boyer-Moore algorithm

We proceed by deriving the multiple keyword generalization of the Boyer-Moore algorithm [BM77] from the algorithm in subsection 3.3. It only differs from the algorithm there in the way the lookahead symbol is taken into account. Assuming $l \neq \varepsilon$ we derive

$$\begin{array}{ll}
& char_{cw}(l\upharpoonright 1, |v|) \\
= & \{\text{definition } char_{cw}\} \\
& (\mathbf{MIN}\, n : n \geq 1 \wedge V^*(l\upharpoonright 1)V^n \cap P \neq \varnothing : n - |v|) \\
\geq & \{V^*(l\upharpoonright 1)V^n \cap P \neq \varnothing \Rightarrow V^*(l\upharpoonright 1)V^n \cap V^*P \neq \varnothing\} \\
& (\mathbf{MIN}\, n : n \geq 1 \wedge V^*(l\upharpoonright 1)V^n \cap V^*P \neq \varnothing : n - |v|) \\
= & \{P \neq \varnothing,\ V^*(l\upharpoonright 1)V^{|p|+1} \cap V^*p \neq \varnothing \text{ for all } p \in P, \text{ nonempty domain}\} \\
& (\mathbf{MIN}\, n : n \geq 1 \wedge V^*(l\upharpoonright 1)V^n \cap V^*P \neq \varnothing : n) - |v| \\
= & \{l \neq \varepsilon, \text{ definition of } char_{bm} \text{ (after derivation)}\} \\
& char_{bm}(l\upharpoonright 1) - |v|
\end{array}$$

where $char_{bm} \in V \to \mathbb{N}$ is defined by

$$char_{bm}(a) = (\mathbf{MIN}\, n : n \geq 1 \wedge V^*aV^n \cap V^*P \neq \varnothing : n) \qquad (a \in V).$$

It results in shift distance $k_{bm}(l,v)$ where $k_{bm} \in V^* \times \mathbf{suff}(P) \to \mathbb{N}$ is defined by

$$k_{bm}(x,y) = \begin{cases} ((char_{bm}(x\upharpoonright 1) - |y|) \max d_{vi}(y)) \min d_{sp}(y) & (x \in V^+, y \in \mathbf{suff}(P)) \\ d_i(y) \min d_{sp}(y) & (x = \varepsilon, y \in \mathbf{suff}(P)). \end{cases}$$

Precomputation of $char_{bm}$ is discussed in subsection 4.3 and of d_{vi}, d_{sp}, and d_i is discussed in subsection 4.2. Approximating k_{bmcw} from below by k_{bm} is referred to as algorithm detail (BM). It results in the multiple keyword generalization of the regular Boyer-Moore algorithm [BM77]. The algorithm is characterized by detail sequence (P_+S_+,RT,SSD,NLAU,OPT,BMCW,BM). The regular Boyer-Moore algorithm can be obtained by restricting P to one keyword (problem detail (OKW) (One KeyWord)). Notice that we have

$$k_{bmcw}(x,y) \geq k_{bm}(x,y) \qquad (x \in V^*, y \in \mathbf{suff}(P)).$$

Inequality can only occur if the lookahead symbol does not occur in any keyword except as the last symbol.

The formula for the Boyer-Moore shift function given here differs from the ones given in [BM77] and [Aho90]. We will show that all formulas are equivalent. First we define

$$m_P = \begin{cases} 1 & \text{if } \varepsilon \in P \\ (\mathbf{MIN}\, p : p \in P : |p|) & \text{if } \varepsilon \notin P. \end{cases}$$

We now have for all $a \in V$

$$char_{bm}(a) \leq m_P.$$

Since for all $y \in \mathbf{suff}(P)$ we have $(\forall n : 1 \leq n < m_P - |y| : yV^n \cap V^*P = \varnothing)$ it follows that

$$m_P - |y| \leq d_{sp}(y) \qquad (y \in \mathbf{suff}(P)).$$

Finally, we derive for $x \in V^+$ and $y \in \mathbf{suff}(P)$

$$\begin{array}{ll} & k_{bm}(x,y) \\ = & \quad \{\text{definition } k_{bm}\} \\ & ((char_{bm}(x\lceil 1) - |y|) \mathbf{\,max\,} d_{vi}(y)) \mathbf{\,min\,} d_{sp}(y) \\ = & \quad \{ d_{sp}(y) \geq m_P - |y| \geq char_{bm}(x\lceil 1) - |y| \} \\ & ((char_{bm}(x\lceil 1) - |y|) \mathbf{\,max\,} d_{vi}(y)) \mathbf{\,min\,}((char_{bm}(x\lceil 1) - |y|) \mathbf{\,max\,} d_{sp}(y)) \\ = & \quad \{\text{distributivity}\} \\ & (char_{bm}(x\lceil 1) - |y|) \mathbf{\,max\,}(d_{vi}(y) \mathbf{\,min\,} d_{sp}(y)). \end{array}$$

The last formula in the preceding derivation coincides with the ones in [BM77] and [Aho90].

3.5 The Commentz-Walter algorithm

Instead of approximating $char_{cw}$ in k_{bmcw} from below by $char_{bm}$ we now approximate d_{vi} in k_{bmcw} from below by d_i. This results in the Commentz-Walter algorithm [CW79a, CW79b]. We derive

$$\begin{array}{ll} & d_{vi}(v) \\ = & \quad \{\text{definition } d_{vi}\} \\ & (\mathbf{MIN}\, n : n \geq 1 \wedge V^*(V \setminus MS(v))vV^n \cap P \neq \varnothing : n) \\ \geq & \quad \{ V^*(V \setminus MS(v))vV^n \cap P \neq \varnothing \Rightarrow V^*vV^n \cap P \neq \varnothing \} \\ & (\mathbf{MIN}\, n : n \geq 1 \wedge V^*vV^n \cap P \neq \varnothing : n) \\ = & \quad \{\text{definition } d_i\} \\ & d_i(v). \end{array}$$

This results in shift distance $k_{cw}(l,v)$ where $k_{cw} \in V^* \times \mathbf{suff}(P) \rightarrow \mathbb{N}$ is defined by

$$k_{cw}(x,y) = \begin{cases} (char_{cw}(x\lceil 1, |y|) \mathbf{\,max\,} d_i(y)) \mathbf{\,min\,} d_{sp}(y) & (x \in V^+, y \in \mathbf{suff}(P)) \\ d_i(y) \mathbf{\,min\,} d_{sp}(y) & (x = \varepsilon, y \in \mathbf{suff}(P)). \end{cases}$$

Precomputation of $char_{cw}$ is discussed in subsection 4.3 and of d_i and d_{sp} in subsection 4.2. Approximating k_{bmcw} from below by k_{cw} is referred to as algorithm detail (CW). It results in the Commentz-Walter algorithm [CW79a, CW79b] that is characterized by detail sequence (P$_+$S$_+$,RT,SSD,NLAU,OPT,BMCW,CW). Notice that we have

$$k_{bmcw}(x,y) \geq k_{cw}(x,y) \qquad (x \in V^*, y \in \mathbf{suff}(P)).$$

Such a comparison can not be made between k_{bm} and k_{cw} as the following example shows.

Example 3.1 Let $V \in \{a,b,c,d\}$, $P = \{cababa\}$, and $x \in V^*$. Shift functions k_{bm} and k_{cw} are incomparable since

$$\begin{array}{rclcl} k_{opt}(xd,a) & = & +\infty \mathbf{\,min\,} 6 & = & 6 \\ k_{bmcw}(xd,a) & = & (+\infty \mathbf{\,max\,} 4) \mathbf{\,min\,} 6 & = & 6 \\ k_{bm}(xd,a) & = & ((6-1) \mathbf{\,max\,} 4) \mathbf{\,min\,} 6 & = & 5 \\ k_{cw}(xd,a) & = & (+\infty \mathbf{\,max\,} 2) \mathbf{\,min\,} 6 & = & 6 \end{array}$$

and

$$\begin{array}{rclcl} k_{opt}(xa,a) & = & +\infty \mathbf{\,min\,} 6 & = & 6 \\ k_{bmcw}(xa,a) & = & ((2-1) \mathbf{\,max\,} 4) \mathbf{\,min\,} 6 & = & 4 \\ k_{bm}(xa,a) & = & ((2-1) \mathbf{\,max\,} 4) \mathbf{\,min\,} 6 & = & 4 \\ k_{cw}(xa,a) & = & ((2-1) \mathbf{\,max\,} 2) \mathbf{\,min\,} 6 & = & 2. \end{array}$$

It also follows that in some cases k_{bmcw} is smaller than k_{opt} and that in some cases k_{bm} and k_{cw} are smaller than k_{bmcw}. □

It is not possible that both $k_{bmcw}(x,y) > k_{bm}(x,y)$ and $k_{bmcw}(x,y) > k_{cw}(x,y)$ hold for some $x \in V^+$ and $y \in \mathbf{suff}(P)$ since the first inequality implies $char_{cw}(x\lceil 1, |y|) = +\infty$ and this in its turn implies $k_{bmcw}(x,y) = d_{sp}(y) = k_{cw}(x,y)$.

3.6 Complete decoupling of recognized suffix and lookahead symbol

The derivations in the previous subsections effect an ever stronger decoupling of the recognized suffix v and the lookahead symbol $l\lceil 1$ in the subsequent shift functions. By approximating d_{vi} in k_{bm} from below by d_i or $char_{cw}$ in k_{cw} by $char_{bm}$ (or both in k_{bmcw}) we obtain a complete decoupling. It results in shift distance $k_{dsl}(l,v)$ where $k_{dsl} \in V^* \times \mathbf{suff}(P) \rightarrow \mathbb{N}$ is defined by

$$k_{dsl}(x,y) = \begin{cases} ((char_{bm}(x\lceil 1) - |y|) \max d_i(y)) \min d_{sp}(y) & (x \in V^+, y \in \mathbf{suff}(P)) \\ d_i(y) \min d_{sp}(y) & (x = \varepsilon, y \in \mathbf{suff}(P)). \end{cases}$$

Precomputation of $char_{bm}$ is discussed in subsection 4.3 and of d_i and d_{sp} in subsection 4.2. The algorithm can be characterized by detail sequences (P$_+$S$_+$,RT,SSD,NLAU,OPT,BMCW,BM,CW) and (P$_+$S$_+$,RT,SSD,NLAU,OPT,BMCW,CW,BM).

3.7 Discarding the lookahead symbol

We weaken the predicate in the range of k_{opt} by weakening its first disjunct to $V^*vV^n \cap P \neq \varnothing$ due to $V^*(l\lceil 1) \subseteq V^*$ and the monotonicity of $\cap$. This weakening step is referred to as discarding the lookahead symbol $l\lceil 1$. The shift distance corresponding to this weakening is $k_{nla}(v)$ where $k_{nla} \in \mathbf{suff}(P) \rightarrow \mathbb{N}$ is defined by

$$k_{nla}(y) = d_i(y) \min d_{sp}(y) \qquad (y \in \mathbf{suff}(P)).$$

Notice that this shift function can also be viewed as an approximation from below of k_{dsl}. Precomputation of d_i and d_{sp} is discussed in subsection 4.2. Approximating k_{opt} from below by k_{nla} is referred to as algorithm detail (NLA) (No LookAhead at mismatching symbol) and results in algorithm (P$_+$S$_+$,RT,SSD,NLAU,OPT,NLA).

3.8 One symbol lookahead at the unscanned part of the input string

In this subsection we consider looking ahead at the first symbol of the unscanned part r of the input string. The first symbol of r will be taken into account independently of the other available information. In this way we obtain stronger variants of all of the shift functions derived thus far. Assuming $r \neq \varepsilon$ we derive

$$\begin{array}{ll}
 & (\mathbf{MIN}\, n : 1 \leq n \wedge \mathbf{suff}(u(r\rceil n)) \cap P \neq \varnothing : n) \\
= & \quad \{ \text{domain split, } 1 \leq n \leq |r|\text{: } r\rceil n = (r\rceil 1)((r\rfloor 1)\rceil(n-1)),\ |r| < n\text{: } r\rceil n = r \} \\
 & (\mathbf{MIN}\, n : 1 \leq n \leq |r| \wedge \mathbf{suff}(u(r\rceil 1)((r\rfloor 1)\rceil(n-1))) \cap P \neq \varnothing : n) \\
 & \mathbf{min}\,(\mathbf{MIN}\, n : |r| < n \wedge \mathbf{suff}(ur) \cap P \neq \varnothing : n) \\
\geq & \quad \{ 1 \leq n \leq |r|\text{: } (r\rfloor 1)\rceil(n-1) \in V^{n-1}, \text{ monotonicity of } \mathbf{suff} \text{ and } cap \} \\
 & (\mathbf{MIN}\, n : 1 \leq n \leq |r| \wedge \mathbf{suff}(u(r\rceil 1)V^{n-1}) \cap P \neq \varnothing : n) \\
 & \mathbf{min}\,(\mathbf{MIN}\, n : |r| < n \wedge \mathbf{suff}(ur) \cap P \neq \varnothing : n) \\
= & \quad \{ r \neq \varepsilon,\ r \in (r\rceil 1)V^{|r|-1},\ \mathbf{suff}(ur) \cap P \neq \varnothing \Rightarrow \mathbf{suff}(u(r\rceil 1)V^{|r|-1}) \cap P \neq \varnothing \} \\
 & (\mathbf{MIN}\, n : 1 \leq n \leq |r| \wedge \mathbf{suff}(u(r\rceil 1)V^{n-1}) \cap P \neq \varnothing : n) \\
\geq & \quad \{ u \in V^*, \text{ monotonicity of } \mathbf{suff} \text{ and } \cap \} \\
 & (\mathbf{MIN}\, n : 1 \leq n \leq |r| \wedge \mathbf{suff}(V^*(r\rceil 1)V^{n-1}) \cap P \neq \varnothing : n) \\
\geq & \quad \{ \text{enlarging domain, changing bound variable: } m = n - 1 \}
\end{array}$$

$$
\begin{array}{ll}
 & (\mathbf{MIN}\, m : 0 \le m \wedge \mathbf{suff}(V^*(r\rceil 1)V^m) \cap P \ne \varnothing : m+1) \\
= & \{\text{ property A.4, } P \ne \varnothing,\ V^*(r\rceil 1)V^{|p|} \cap V^*p \ne \varnothing \text{ for all } p \in P \text{, nonempty domain }\} \\
 & (\mathbf{MIN}\, m : 0 \le m \wedge V^*(r\rceil 1)V^m \cap V^*P \ne \varnothing : m) + 1 \\
= & \{\text{ definition } char_{la} \text{ (after derivation) }\} \\
 & char_{la}(r\rceil 1) + 1
\end{array}
$$

where $char_{la} \in V \to \mathbb{N}$ is defined by

$$char_{la}(a) = (\mathbf{MIN}\, n : 0 \le n \wedge V^*aV^n \cap V^*P \ne \varnothing : n) \quad (a \in V).$$

Precomputation of $char_{la}$ is discussed in subsection 4.3.

Let $M(u,r)$ denote the first expression in the preceding derivation as well as the the first expression in the derivation in subsection 3.1, and let $N(u)$ denote the last expression in the derivation in subsection 3.1. We then have

$$
\begin{array}{ll}
 & M(u,r) \\
= & \{\text{ property } \mathbf{max} \} \\
 & M(u,r)\,\mathbf{max}\,M(u,r) \\
\ge & \{\text{ derivation in subsection 3.1, preceding derivation }\} \\
 & N(u)\,\mathbf{max}(char_{la}(r\rceil 1) + 1)
\end{array}
$$

Since all shift functions derived in the previous subsections are approximations from below of $N(u)$ the preceding derivation shows that they all may be extended with $\mathbf{max}(char_{la}(r\rceil 1)+1)$ to form a class of stronger shift functions of signature $k(l,v,r)$ (algorithm detail (OLAU) (One symbol LookAhead at Unscanned part of the input string)). The first derivation in this subsection shows that it is also possible to couple the information on $r\rceil 1$ with the information on l and v $(u = lv)$. We will not pursue that direction any further in this paper.

4 Precomputation

In this section we derive algorithms for the precomputation of the functions used in the pattern matching algorithms in sections 2 and 3. The algorithms are correct due to their formal derivation. This can not always be said about the algorithms found in the literature, mostly due to the absence of any formal derivation (see for instance the single keyword Boyer-Moore precomputation algorithms given in [BM77], [KMP77], and [Ryt80], where each article shows the preceding article to give an incorrect precomputation algorithm). Moreover, we give the first formal derivation of the precomputation algorithms for the Boyer-Moore family of algorithms. They can, amongst others, be specialized to a correct precomputation algorithm for the single keyword Boyer-Moore algorithm. In fact, we show that the definition of all d-functions introduced in section 3 can be rewritten into a form in accordance with one general pattern. Subsequently, a general precomputation algorithm scheme for this general pattern is derived that can be instantiated for every d-function.

4.1 Precomputation of τ_P

The transition function $\tau_P \in \mathbf{suff}(P) \times V \to (\mathbf{suff}(P) \cup \{\bot\})$ of the reverse trie corresponding to P is defined by

$$\tau_P(u,a) = \begin{cases} au & \text{if } au \in \mathbf{suff}(P) \\ \bot & \text{if } au \notin \mathbf{suff}(P) \end{cases} \qquad (u \in \mathbf{suff}(P), a \in V).$$

Since **suff** is idempotent and the definition of τ_P only depends on **suff**(P), we have $\tau_P = \tau_{\mathbf{suff}(P)}$. Set P being nonempty we have $\mathbf{suff}(P) = \{\varepsilon\} \cup \mathbf{suff}(P)$ and $\tau_{\mathbf{suff}(P)} = \tau_{\{\varepsilon\}\cup\mathbf{suff}(P)}$. These observations lead to the following algorithm (cf. [AC75], section 3, algorithm 2) to compute τ_P in which variable *tau* is used to calculate and store τ_P thereby viewing *tau* as a set of ordered pairs (the usual notion of a function) and abbreviate statements like $tau := tau + \{((x,a),y)\}$ to $tau(x,a) := y$:

```
$tau := \varnothing;$
$\{\ tau = \tau_{\varnothing}\ \}$
for $a : a \in V \rightarrow tau(\varepsilon, a) := \bot$ rof;
$\{\ tau = \tau_{\{\varepsilon\}}\ \}$
$P_d, P_r := \varnothing, P;$
$\{$ invariant: $P_d \cup P_r = P \wedge P_d \cap P_r = \varnothing \wedge tau = \tau_{\{\varepsilon\} \cup \mathbf{suff}(P_d)}\ \}$
do $P_r \neq \varnothing \rightarrow$
    $p : p \in P_r;$
    $u, v := p, \varepsilon;$
    $\{$ invariant: $uv = p \wedge tau = \tau_{\{\varepsilon\} \cup \mathbf{suff}(P_d) \cup \mathbf{suff}(v)}\ \}$
    do $u \neq \varepsilon \rightarrow$
        if $tau(v, u\lceil 1) = \bot \rightarrow tau(v, u\lceil 1) := (u\lceil 1)v;$
                                        for $a : a \in V \rightarrow tau((u\lceil 1)v, a) := \bot$ rof
        | $tau(v, u\lceil 1) \neq \bot \rightarrow$ skip
        fi;
        $u, v := u\lfloor 1, (u\lceil 1)v$
    od;
    $P_d, P_r := P_d + \{p\}, P_r - \{p\}$
od$\{\ tau = \tau_P\ \}$
```

In the algorithms we use + for the union of disjoint sets and − for the difference of a set and a subset of it. Notice that the algorithm does a depth first traversal of the reverse trie. Also notice that variable P_d is only needed to formulate an invariant for *tau*, so it may safely be removed from the algorithm. Furthermore, the states of the reverse trie are represented by strings. In practice, one can resort to a more suitable representation, for instance a representation by natural numbers. We will not elaborate this here.

4.2 Precomputation of d-functions

In this subsection we show that all d-functions introduced in section 3 can be written according to a general pattern. For this general pattern a general precomputation algorithm is derived. In order to obtain a precomputation algorithm for a particular d-function one only has to instantiate the general precomputation algorithm and possibly simplify the resulting algorithm.

The general pattern we strive for is a function $d \in V \times \mathbf{suff}(P) \rightarrow \mathbb{N}$ defined for $a \in V$ and $y \in \mathbf{suff}(P)$ by

$$d(a,y) = \big(\mathbf{MIN}\, t : t \in \mathbf{suff}(P) \setminus \{\varepsilon\} \wedge Q(a,t) \wedge R(a,y,t) \wedge y <_p t : |t| - |y|\big)$$

where Q is a predicate on $V \times V^*$ and R a predicate on $V \times V^* \times V^*$. Why both Q and R are introduced will become clear when we derive an algorithm scheme for the computation of d. We will now show that all d-functions introduced in section 3 can be expressed in this pattern. In the following derivations let $a \in V$ and $y \in \mathbf{suff}(P)$.

- We derive

$$\begin{aligned}
& d_{opt}(a,y) \\
= \quad & \{\text{definition } d_{opt}\} \\
& \big(\mathbf{MIN}\, n : n \geq 1 \wedge V^* a y V^n \cap P \neq \varnothing : n\big) \\
= \quad & \{\text{property A.4}\}
\end{aligned}$$

$$\begin{array}{cl}
 & (\mathbf{MIN}\, n : n \geq 1 \wedge ayV^n \cap \mathbf{suff}(P) \neq \varnothing : n) \\
= & \quad \{ \text{change of bound variable: } n = |s| \} \\
 & (\mathbf{MIN}\, s : s \in V^+ \wedge ays \in \mathbf{suff}(P) : |s|) \\
= & \quad \{ \text{change of bound variable: } t = ys \} \\
 & (\mathbf{MIN}\, t : t \in \mathbf{suff}(P) \setminus \{\varepsilon\} \wedge at \in \mathbf{suff}(P) \wedge y <_p t : |t| - |y|)
\end{array}$$

Hence, we have expressed d_{opt} according to the general pattern with $Q(a,t) = at \in \mathbf{suff}(P)$ and $R(a,y,t) = \mathbf{true}$. Notice that $at \in \mathbf{suff}(P) \equiv \tau(t,a) \neq \bot$.

- Assuming $y = \varepsilon$ we derive

$$\begin{array}{cl}
 & d_{sp}(\varepsilon) \\
= & \quad \{ \text{definition } d_{sp} \} \\
 & (\mathbf{MIN}\, n : n \geq 1 \wedge V^n \cap V^*P \neq \varnothing : n) \\
= & \quad \{ \text{calculus} \} \\
 & \Big(\mathbf{MIN}\, t : t \in P : (\mathbf{MIN}\, n : n \geq 1 \wedge V^n \cap V^*t \neq \varnothing : n)\Big) \\
= & \quad \{ \text{domain split: } P = P \setminus \{\varepsilon\} \cup (P \cap \{\varepsilon\}) \} \\
 & (\mathbf{MIN}\, t : t \in P \setminus \{\varepsilon\} : |t|) \,\mathbf{min}\, (\mathbf{MIN}\, t : t \in P \cap \{\varepsilon\} : 1) \\
= & \quad \{ \text{rewriting in order to obtain general pattern} \} \\
 & (\mathbf{MIN}\, t : t \in \mathbf{suff}(P) \setminus \{\varepsilon\} \wedge t \in P \wedge \varepsilon <_p t : |t| - |\varepsilon|) \,\mathbf{min}\, (\mathbf{MIN}\, t : t \in P \cap \{\varepsilon\} : 1)
\end{array}$$

Assuming $y \neq \varepsilon$ we derive

$$\begin{array}{cl}
 & d_{sp}(y) \\
= & \quad \{ \text{definition } d_{sp} \} \\
 & (\mathbf{MIN}\, n : n \geq 1 \wedge yV^n \cap V^*P \neq \varnothing : n) \\
= & \quad \{ \text{property A.4} \} \\
 & (\mathbf{MIN}\, n : n \geq 1 \wedge \mathbf{suff}(yV^n) \cap P \neq \varnothing : n) \\
= & \quad \{ y \neq \varepsilon, \text{ hence } \mathbf{suff}(yV^n) = yV^n \cup \mathbf{suff}((y\rfloor 1)V^n), \text{ set calculus} \} \\
 & (\mathbf{MIN}\, n : n \geq 1 \wedge yV^n \cap P \neq \varnothing : n) \\
 & \mathbf{min}\ (\mathbf{MIN}\, n : n \geq 1 \wedge \mathbf{suff}((y\rfloor 1)V^n) \cap P \neq \varnothing : n) \\
= & \quad \{ \text{change of bound variable: } n = |s|, \text{ property A.4} \} \\
 & (\mathbf{MIN}\, s : s \in V^+ \wedge ys \in P : |s|) \,\mathbf{min}\, (\mathbf{MIN}\, n : n \geq 1 \wedge (y\rfloor 1)V^n \cap V^*P \neq \varnothing : n) \\
= & \quad \{ \text{change of bound variable: } t = ys,\ y \in \mathbf{suff}(P),\ y\rfloor 1 \in \mathbf{suff}(P), \text{ definition } d_{sp} \} \\
 & (\mathbf{MIN}\, t : t \in P \wedge y <_p t : |t| - |y|) \,\mathbf{min}\, d_{sp}(y\rfloor 1) \\
= & \quad \{ \text{rewriting in order to obtain general pattern} \} \\
 & (\mathbf{MIN}\, t : t \in \mathbf{suff}(P) \setminus \{\varepsilon\} \wedge t \in P \wedge y <_p t : |t| - |y|) \,\mathbf{min}\, d_{sp}(y\rfloor 1)
\end{array}$$

Although the derived definition of d_{sp} is recursive and d_{sp} does not have an argument $a \in V$ one can still discern the general pattern with $Q(a,t) = t \in P$ and $R(a,y,t) = \mathbf{true}$. Precomputation of d_{sp} can be done according to the general precomputation algorithm without an iteration over V, followed by a breadth first traversal of the reverse trie.

- We derive

$$
\begin{array}{cl}
 & d_i(y) \\
= & \quad \{\text{definition } d_i\} \\
 & (\mathbf{MIN}\, n : n \geq 1 \wedge V^* y V^n \cap P \neq \varnothing : n) \\
= & \quad \{\text{property A.4}\} \\
 & (\mathbf{MIN}\, n : n \geq 1 \wedge y V^n \cap \mathbf{suff}(P) \neq \varnothing : n) \\
= & \quad \{\text{change of bound variable: } n = |s|\} \\
 & (\mathbf{MIN}\, s : s \in V^+ \wedge ys \in \mathbf{suff}(P) : |s|) \\
= & \quad \{\text{change of bound variable: } t = ys\} \\
 & (\mathbf{MIN}\, t : t \in \mathbf{suff}(P) \setminus \{\varepsilon\} \wedge y <_p t : |t| - |y|)
\end{array}
$$

  Although d_i does not have an argument $a \in V$ its definition still matches the general pattern with $Q(a,t) = \mathbf{true}$ and $R(a,y,t) = \mathbf{true}$.

- We derive

$$
\begin{array}{cl}
 & d_{vi}(y) \\
= & \quad \{\text{definition } d_{vi}\} \\
 & (\mathbf{MIN}\, n : n \geq 1 \wedge V^*(V \setminus MS(y)) y V^n \cap P \neq \varnothing : n) \\
= & \quad \{\text{property A.4}\} \\
 & (\mathbf{MIN}\, n : n \geq 1 \wedge (V \setminus MS(y)) y V^n \cap \mathbf{suff}(P) \neq \varnothing : n) \\
= & \quad \{\text{change of bound variable: } n = |s|\} \\
 & (\mathbf{MIN}\, s : s \in V^+ \wedge (V \setminus MS(y)) ys \cap \mathbf{suff}(P) \neq \varnothing : |s|) \\
= & \quad \{\text{change of bound variable: } t = ys\} \\
 & (\mathbf{MIN}\, t : t \in \mathbf{suff}(P) \setminus \{\varepsilon\} \wedge (V \setminus MS(y)) t \cap \mathbf{suff}(P) \neq \varnothing \wedge y <_p t : |t| - |y|) \\
= & \quad \{\text{definition } MS\} \\
 & (\mathbf{MIN}\, t : t \in \mathbf{suff}(P) \setminus \{\varepsilon\} \wedge MS(t) \cap (V \setminus MS(y)) \neq \varnothing \wedge y <_p t : |t| - |y|)
\end{array}
$$

  Apart from the fact that d_{vi} does not have an argument $a \in V$ its definition matches the general pattern with $Q(a,t) = \mathbf{true}$ and $R(a,y,t) = MS(t) \cap (V \setminus MS(y)) \neq \varnothing$.

Having expressed all d-functions from section 3 in the general pattern we proceed by giving a rather straightforward and nondeterministic algorithm to compute d which will serve as a starting point for further algorithm derivations (notice that program variable *dee* is used to compute and finally store function d):

```
for $y : y \in \mathbf{suff}(P) \rightarrow$
    for $a : a \in V \rightarrow dee(a,y) :=$ +inf rof
rof;
for $t : t \in \mathbf{suff}(P) \setminus \{\varepsilon\} \rightarrow$
    for $a : a \in V \rightarrow$
        if $Q(a,t) \rightarrow$ for $y : y \in \mathbf{suff}(P) \wedge y <_p t \rightarrow$
                        if $R(a,y,t) \rightarrow dee(a,y) := dee(a,y)\,\mathbf{min}(|t| - |y|)$
                        | $\neg R(a,y,t) \rightarrow$ skip
                        fi
                    rof
        | $\neg Q(a,t) \rightarrow$ skip
        fi
    rof
rof{ $dee = d$ } .
```

For d_i and d_{sp} the loop iterating over V can be omitted. For d_{sp} the following additional breadth first traversal of the reverse trie ($\mathbf{suff}(P)$) is needed to complete the computation:

```
$n := 1$;
do $\mathbf{suff}(P) \cap V^n \neq \varnothing \rightarrow$
    for $t : t \in \mathbf{suff}(P) \cap V^n \rightarrow deesp(t) := deesp(t)\,\mathbf{min}\,deesp(t\rfloor 1)$ rof
od{ $deesp = d_{sp}$ } .
```

First, we concentrate on making the innermost repetition deterministic. Define $sp \in \mathbf{suff}(P) \rightarrow \mathcal{P}(\mathbf{suff}(P))$ by

$$sp(t) = \{\, y \mid y \in \mathbf{suff}(P) \wedge y <_p t \,\} \qquad (t \in \mathbf{suff}(P)),$$

the set of all suffixes of keywords that are a proper prefix of t. Notice that for all $t \in \mathbf{suff}(P) \setminus \{\varepsilon\}$ $sp(t)$ is finite, nonempty, and linearly ordered with respect to $\leq_p$. Therefore, we can define $msp \in \mathbf{suff}(P) \setminus \{\varepsilon\} \rightarrow \mathbf{suff}(P)$ by

$$msp(t) = \left(\mathbf{MAX}_{\leq_p}\, y : y \in \mathbf{suff}(P) \wedge y <_p t : y\right) \qquad (t \in \mathbf{suff}(P) \setminus \{\varepsilon\})$$

being the maximal element of $sp(t)$. In literature [AC75, BM77, CW79a, KMP77] function msp is known as the *failure function* corresponding to the reverse trie. For $t \in \mathbf{suff}(P) \setminus \{\varepsilon\}$ we derive a recursive definition of $sp(t)$ in terms of function msp:

$$\begin{aligned}
& sp(t) \\
= \quad & \{\text{ definition } sp \,\} \\
& \{\, y \mid y \in \mathbf{suff}(P) \wedge y <_p t \,\} \\
= \quad & \{\, t \in \mathbf{suff}(P) \setminus \{\varepsilon\}, \text{ property A.6} \,\} \\
& \{\, y \mid y \in \mathbf{suff}(P) \wedge \left(y = msp(t) \vee y <_p msp(t)\right) \} \\
= \quad & \{\text{ definition } sp \text{ and } msp \,\} \\
& \{msp(t)\} \cup sp(msp(t))
\end{aligned}$$

Provided msp is already computed (precomputation of msp is discussed in subsection 4.4) the innermost repetition traversing the set $sp(t)$ can be replaced by the following deterministic repetition (variable $\overline{v}$ is a ghost variable needed to express the invariant):

```
$v := t$; { $\overline{v} := \varnothing$; }
{ invariant: $v \in sp(t) \cup \{t\} \wedge sp(t) = \overline{v} \cup sp(v) \wedge \overline{v} \cap sp(v) = \varnothing$ }
do $v \neq \varepsilon \rightarrow$ { $sp(v) = \{msp(v)\} \cup sp(msp(v))$ }
            $v := msp(v)$; { $\overline{v} := \overline{v} + \{v\}$; }
            if $R(a,v,t) \rightarrow dee(a,v) := dee(a,v) \min(|t| - |v|)$
            | $\neg R(a,v,t) \rightarrow$ skip
            fi
od{ $v = \varepsilon$, so $sp(v) = \varnothing$ and $sp(t) = \overline{v}$ }
```

The invariant expresses that we have a bipartition of $sp(t)$ in $\overline{v}$ (elements of $sp(t)$ that have already contributed to the computation of d) and $sp(v)$ (the other elements of $sp(t)$).

In case $R(a,y,t) = \mathtt{true}$ for all a, y, and t the inner repetition can be made more efficient. Notice that this can be done for all presented d-functions except d_{vi}. In the following assume that $R(a,y,t) = \mathtt{true}$ for all a, y, and t. Suppose that for some $v \in sp(t)$ in the above repetition we have $|t| - |v| \geq dee(a,v)$. From the structure of the algorithms we infer that $dee(a,v) = |t_0| - |v|$ for some $t_0 \in \mathbf{suff}(P) \setminus \{\varepsilon\}$ with $|t_0| \leq |t|$ that has already contributed to the computation of d. Therefore, for all $s \in sp(v)$ we have $dee(a,s) \leq |t_0| - |s|$ leading to

$$
\begin{array}{ll}
 & dee(a,s) \\
\leq & \qquad \{\,\} \\
 & |t_0| - |s| \\
= & \qquad \{\, dee(a,v) = |t_0| - |v| \,\} \\
 & dee(a,v) + |v| - |s| \\
\leq & \qquad \{\, dee(a,v) \leq |t| - |v| \,\} \\
 & |t| - |s|
\end{array}
$$

Hence, the contribution of t will not change the already computed value of $dee(a,s)$ for $s \in sp(v)$ and the inner repetition can be terminated. This yields the following repetition using an additional boolean variable *contributes* (notice that ghost variable $\overline{v}$ is omitted):

```
$v := t$; $contributes := \mathtt{true}$;
do $v \neq \varepsilon \wedge contributes \rightarrow$
    $v := msp(v)$;
    if $|t| - |v| < dee(a,v) \rightarrow dee(a,v) := |t| - |v|$
    | $|t| - |v| \geq dee(a,v) \rightarrow contributes := \mathtt{false}$
    fi
od.
```

Variable *contributes* can be removed resulting in the following repetition

```
$v := t$;
do $v \neq \varepsilon \rightarrow$
    $v := msp(v)$;
    if $|t| - |v| < dee(a,v) \rightarrow dee(a,v) := |t| - |v|$
    | $|t| - |v| \geq dee(a,v) \rightarrow v := \varepsilon$
    fi
od.
```

In order to further exploit this phenomenon the elements of $\mathbf{suff}(P) \setminus \{\varepsilon\}$ are dealt with in order of increasing length, i.e. the outermost repetition of the general precomputation algorithm does a breadth first traversal of the reverse trie. This results in the following algorithm:

```
for $y : y \in \mathbf{suff}(P) \rightarrow$
    for $a : a \in V \rightarrow dee(a,y) := \mathtt{+inf}$ rof
rof;
$n := 1$;
do $\mathbf{suff}(P) \cap V^n \neq \varnothing \rightarrow$
    for $t : t \in \mathbf{suff}(P) \cap V^n \rightarrow$
        for $a : a \in V \rightarrow$
            if $Q(a,t) \rightarrow v := t$;
                    do $v \neq \varepsilon \rightarrow$
                        $v := msp(v)$;
                        if $|t| - |v| < dee(a,v) \rightarrow dee(a,v) := |t| - |v|$
                        | $|t| - |v| \geq dee(a,v) \rightarrow v := \varepsilon$
                        fi
                    od
            | $\neg Q(a,t) \rightarrow$ skip
            fi
        rof
    rof;
    $n := n + 1$
od.
```

In this optimized breadth first precomputation algorithm for each node v in the reverse trie and each symbol from V the step from v to $msp(v)$ is done at most two times. Therefore, the precomputation time is $\mathcal{O}(|\mathbf{suff}(P)| \cdot |V|)$. If the traversal of V can be omitted (as is the case for the precomputation of d_i and d_{sp}) it is $\mathcal{O}(|\mathbf{suff}(P)|)$.

The breadth first precomputation algorithm for d_i can be simplified further by observing that since $Q(a,t) = \mathbf{true}$ the steps taken from t are always preceded by the steps taken from $msp(t)$ (provided $msp(t) \neq \varepsilon$) since $|msp(t)| < |t|$. So, only the contribution of t to $d_i(msp(t))$ has to be considered. This results in the following precomputation algorithm for d_i (program variable *deei* is used to compute and finally store d_i):

```
for $y : y \in \mathbf{suff}(P) \rightarrow deei(y) := \mathtt{+inf}$ rof;
$n := 1$;
do $\mathbf{suff}(P) \cap V^n \neq \varnothing \rightarrow$
    for $t : t \in \mathbf{suff}(P) \cap V^n \rightarrow$
        $deei(msp(t)) := deei(msp(t))\ \mathbf{min}(|t| - |msp(t)|)$
    rof;
    $n := n + 1$
od.
```

Notice that the breadth first traversal in this algorithm may be replaced by an arbitrary traversal of the reverse trie.

4.3 Precomputation of $char_{cw}$, $char_{bm}$, and $char_{la}$

Function $char_{cw}$ can be expressed as

$$char_{cw}(a,z) = \begin{cases} \mathtt{+inf} & \text{if } \overline{char}_{cw}(a) = \mathtt{+inf} \\ \overline{char}_{cw}(a) - z & \text{if } \overline{char}_{cw}(a) \neq \mathtt{+inf} \end{cases} \qquad (a \in V, z \in \mathbb{N})$$

where function $\overline{char}_{cw} \in V \rightarrow \mathbb{N}$ is defined by

$$\overline{char}_{cw}(a) = (\mathbf{MIN}\, n : n \geq 1 \wedge V^*aV^n \cap P \neq \varnothing : n) \qquad (a \in V).$$

From the definition $\overline{char}_{cw}$ it immediately follows that that its computation can be interwoven with the precomputation of the reverse trie τ.

Next, we derive for $a \in V$

$$\begin{array}{cl}
 & char_{bm}(a) \\
= & \quad \{\text{definition } char_{bm}\} \\
 & (\mathbf{MIN}\, n : n \geq 1 \wedge V^*aV^n \cap V^*P \neq \varnothing : n) \\
= & \quad \{\text{property A.5.iii}\} \\
 & (\mathbf{MIN}\, n : n \geq 1 \wedge (V^*aV^n \cap P \neq \varnothing \vee aV^n \cap V^+P \neq \varnothing) : n) \\
= & \quad \{\text{disjunctive range, definition } \overline{char}_{cw}\} \\
 & \overline{char}_{cw}(a) \,\mathbf{min}\, (\mathbf{MIN}\, n : n \geq 1 \wedge aV^n \cap V^+P \neq \varnothing : n) \\
= & \quad \{aV^n \cap V^+P \neq \varnothing \equiv V^n \cap V^*P \neq \varnothing\} \\
 & \overline{char}_{cw}(a) \,\mathbf{min}\, (\mathbf{MIN}\, n : n \geq 1 \wedge V^n \cap V^*P \neq \varnothing : n) \\
= & \quad \{\text{definition } m_P \text{ (subsection 3.4)}\} \\
 & \overline{char}_{cw}(a) \,\mathbf{min}\, m_P.
\end{array}$$

Finally, we derive for $a \in V$

$$\begin{array}{cl}
 & char_{la}(a) \\
= & \quad \{\text{definition } char_{la}\} \\
 & (\mathbf{MIN}\, n : 0 \leq n \wedge V^*aV^n \cap V^*P \neq \varnothing : n) \\
= & \quad \{\text{domain split: } n \geq 1 \vee n = 0, \text{ definition } char_{bm}\} \\
 & char_{bm}(a) \,\mathbf{min}\, (\mathbf{MIN}\, n : n = 0 \wedge V^*a \cap V^*P \neq \varnothing : 0) \\
= & \quad \{\text{property A.5.ii}\} \\
 & char_{bm}(a) \,\mathbf{min}\, (\mathbf{MIN}\, n : n = 0 \wedge (a \in \mathbf{suff}(P) \vee \varepsilon \in P) : 0)
\end{array}$$

We conclude that $char_{cw}$, $char_{bm}$, and $char_{la}$ can be computed from $\overline{char}_{cw}$.

4.4 Precomputation of *msp*

We conclude this section with the derivation of an algorithm computing *msp*. We start by deriving

- for $a \in \mathbf{suff}(P) \cap V$

$$\begin{array}{cl}
 & msp(a) \\
= & \quad \{\text{definition } msp\} \\
 & (\mathbf{MAX}_{\leq_p}\, y : y \in \mathbf{suff}(P) \wedge y <_p a : y) \\
= & \quad \{y <_p a \equiv y = \varepsilon,\ \varepsilon \in \mathbf{suff}(P)\} \\
 & \varepsilon
\end{array}$$

- and for $ax \in \mathbf{suff}(P)$, $a \in V$, and $x \in \mathbf{suff}(P) \setminus \{\varepsilon\}$

$$\begin{array}{cl}
 & msp(ax) \\
= & \quad \{\text{definition } msp\} \\
 & (\mathbf{MAX}_{\leq_p}\, y : y \in \mathbf{suff}(P) \wedge y <_p ax : y) \\
= & \quad \{\text{domain split, } \varepsilon <_p ax,\ \varepsilon \in \mathbf{suff}(P)\}
\end{array}$$

$$
\begin{array}{ll}
 & (\mathbf{MAX}_{\leq_p}\, y : y \in \mathbf{suff}(P) \land y <_p ax \land y \neq \varepsilon : y)\ \mathbf{max}_{\leq_p}\, \varepsilon \\
= & \quad \{\text{change of bound variable: } y = ay'\} \\
 & (\mathbf{MAX}_{\leq_p}\, y' : ay' \in \mathbf{suff}(P) \land y' <_p x : ay')\ \mathbf{max}_{\leq_p}\, \varepsilon \\
= & \quad \{ay' \in \mathbf{suff}(P) \Rightarrow y' \in \mathbf{suff}(P),\ x \in \mathbf{suff}(P) \setminus \{\varepsilon\}, \text{definition } sp\} \\
 & (\mathbf{MAX}_{\leq_p}\, y' : ay' \in \mathbf{suff}(P) \land y' \in sp(x) : ay')\ \mathbf{max}_{\leq_p}\, \varepsilon
\end{array}
$$

From this it follows that $msp(ax)$ can be computed by a linear search in downward order over $sp(x)$ (remember that $sp(x)$ is linearly ordered with respect to $\leq_p$) starting with $msp(x)$. Provided the computation of msp is done using a breadth first traversal of the reverse trie (**suff**(P)) the value of msp is already computed for all all elements of $sp(x) \cup \{x\}$ and can therefore be used to implement the linear search over $sp(x)$. This results in the following algorithm (variable $emsp$ is used to compute and finally store msp):

```
for $a : a \in \mathbf{suff}(P) \cap V \rightarrow emsp(a) := \varepsilon$ rof;
$n := 1$;
{ invariant: $1 \leq n \leq (\mathbf{MAX}\, p : p \in P : |p|) + 1$
            $\land\ (\forall y : y \in \mathbf{suff}(P) \setminus \{\varepsilon\} \land |y| \leq n : emsp(y) = msp(y))$ }
do $\mathbf{suff}(P) \cap V^n \neq \varnothing \rightarrow$
   for $t : t \in \mathbf{suff}(P) \cap V^n \rightarrow$
      for $a : a \in V \rightarrow$
         if $at \in \mathbf{suff}(P) \rightarrow v := emsp(t)$;
                              { linear search }
                              do $av \notin \mathbf{suff}(P) \land v \neq \varepsilon \rightarrow v := emsp(v)$ od;
                              if $av \in \mathbf{suff}(P) \rightarrow emsp(at) := av$
                              | $av \notin \mathbf{suff}(P) \land v = \varepsilon \rightarrow emsp(at) := \varepsilon$
                              fi
         | $at \notin \mathbf{suff}(P) \rightarrow$ skip
         fi
      rof
   rof;
   $n := n + 1$
od.
```

This breadth first algorithm computing msp can be combined with the breadth first algorithm computing the d-functions. The precomputation time now is $\mathcal{O}(|\mathbf{suff}(P)|^2 \cdot |V|)$. This can be reduced to $\mathcal{O}(|\mathbf{suff}(P)| \cdot |V|)$ at the expense of $\mathcal{O}(|\mathbf{suff}(P)| \cdot |V|)$ additional storage space by also computing and storing the transition function γ_r of the reverse trie where $\gamma_r \in V \times \mathbf{suff}(P) \rightarrow \mathbf{suff}(P)$ is defined by

$$\gamma_r(a, x) = (\mathbf{MAX}_{\leq_p}\, y : y \in \mathbf{suff}(P) \land y \leq_p ax : y) \qquad (a \in V, x \in \mathbf{suff}(P)).$$

The details of this approach can be found in [WZ92].

5 Conclusions

In this paper we derived and presented a taxonomy of sublinear keyword pattern matching algorithms closely related to the Boyer-Moore algorithm [BM77] and the Commentz-Walter algorithm [CW79a, CW79b]. It includes, amongst others, the multiple keyword generalization of the single keyword Boyer-Moore algorithm and the algorithm presented by Fan and Su [FS93, FS94]. We presented the algorithms within a common framework permitting an easier comprehension of and a better comparison between the algorithms. This was achieved by the systematic and formal derivation of the algorithms from a common starting point and by factoring out of common portions of the derivations. The derivations were done through series of refinements to either algorithm or problem. A refinement to the algorithm/problem is referred to as the introduction of an algorithm/problem detail. The sequence of details that are subsequently introduced in a derivation characterizes the algorithm obtained by that derivation. Detail sequences can therefore be used to classify the algorithms in the taxonomy. Algorithms can now be compared by looking at their detail sequences. The taxonomy graph in figure 1 constitutes a concise presentation and classification of the pattern matching algorithms discussed, vertices representing algorithms and edges representing the addition of an algorithm or problem detail. It can be viewed as an alternative table of contents to this paper. Our results show how fruitful the applied method of developing a taxonomy is (it was inspired by the method described by Jonkers [Jon83]).

Introduction of the notion of safe shift distances proved to be essential for the derivation of the various algorithms. All algorithms are characterized by a—systematically derived and more or less easy to compute—approximation from below of the maximal safe shift distance, computation of the latter being equivalent to the keyword pattern matching problem itself. The systematic derivation provided a means to compare the algorithms and their matching speeds, and to get a better understanding of the algorithms and their interrelations. Perhaps this better understanding will help further the use of the algorithms from this family. Our derivations show the Commentz-Walter algorithm not to be the multiple keyword generalization of the Boyer-Moore algorithm (as was the original intention of Commentz-Walter) and that such a generalization can indeed be obtained. Of the algorithms presented the algorithm by Fan and Su [FS93, FS94] is the fastest (at the expense of additional precomputation time and additional storage requirements), followed by the common ancestor of the Boyer-Moore and Commentz-Walter algorithms, and then by both the multiple keyword generalization of the Boyer-Moore algorithm [BM77] and the Commentz-Walter algorithm [CW79a, CW79b]. The latter two are incomparable in matching speed. It is clear that we have not derived and presented all algorithms of the Boyer-Moore family. Our derivation method, however, clearly indicates how yet other members of this family of algorithms may be derived.

Apart from giving a taxonomy of pattern matching algorithms from the Boyer-Moore family we presented the first formally derived and therefore correct precomputation algorithms (this can not always be said about the algorithms found in the literature, mostly due to the absence of any formal derivation; see for instance the many solutions for the Boyer-Moore precomputation that have been published, corrected and republished). In fact, we showed that most of the precomputation algorithms can be obtained as instantiations of a general precomputation algorithm scheme derived for a general function pattern in which most components of the various shift functions can be expressed. Thus, we provided a common framework for the precomputation algorithms as well.

6 Acknowledgements

We thank Kees Hemerik and Frans Kruseman Aretz for their careful reading of earlier drafts of this paper and their constructive criticisms. We also thank Anne Kaldewaij and Lex Bijlsma for reading the final version.

A Definitions and properties

This section provides a series of definitions and properties which are used throughout this paper. In the following let V be an alphabet.

For a string $w \in V^*$ w^R denotes the reversal of w. For any language $L \subseteq V^*$ we define $L^R = \{w^R \mid w \in L\}$ (the reversal of language L).

Definition A.1 *The infix operators* $\upharpoonleft, \downharpoonleft, \upharpoonright, \downharpoonright : V^* \times \mathbf{N} \to V^*$ *are defined by*

$$\begin{array}{rcll} v\upharpoonleft 0 & = & \varepsilon & (v \in V^*) \\ \varepsilon\upharpoonleft(k+1) & = & \varepsilon & (k \geq 0) \\ (aw)\upharpoonleft(k+1) & = & a(w\upharpoonleft k) & (k \geq 0, a \in V, w \in V^*) \\ v\downharpoonleft 0 & = & v & (v \in V^*) \\ \varepsilon\downharpoonleft(k+1) & = & \varepsilon & (k \geq 0) \\ (aw)\downharpoonleft(k+1) & = & w\downharpoonleft k & (k \geq 0, a \in W, w \in V^*) \end{array}$$

Define $\upharpoonright$ *as* $v\upharpoonright k = (v^R\upharpoonleft k)^R$ *and* $\downharpoonright$ *as* $v\downharpoonright k = (v^R\downharpoonleft k)^R$. *The operators* $\upharpoonleft, \downharpoonleft, \upharpoonright$, *and* $\downharpoonright$ *are called "left take," "left drop," "right take," and "right drop" respectively.* □

Definition A.2 *Functions* $\mathbf{pref} : \mathcal{P}(V^*) \to \mathcal{P}(V^*)$ *and* $\mathbf{suff} : \mathcal{P}(V^*) \to \mathcal{P}(V^*)$ *are defined by*

$$\mathbf{pref}(L) = \{w \mid w \in V^* \wedge (\exists x : x \in V^* : wx \in L)\}$$

and

$$\mathbf{suff}(L) = (\mathbf{pref}(L^R))^R = \{w \mid w \in V^* \wedge (\exists x : x \in V^* : xw \in L)\}.$$

□

For $w \in V^*$ we will write $\mathbf{pref}(w)$ ($\mathbf{suff}(w)$) instead of $\mathbf{pref}(\{w\})$ ($\mathbf{suff}(\{w\})$).

Definition A.3 *The relations* $\leq_p$ *and* $\leq_s$ *over* $V^* \times V^*$ *are defined by* $u \leq_p v \equiv u \in \mathbf{pref}(v)$ *and* $u \leq_s v \equiv u \in \mathbf{suff}(v)$. □

The following two properties are used in the derivation of the Commentz-Walter precomputation algorithm.

Property A.4 *Let* $A, B \subseteq V^*$. *Then* $\mathbf{pref}(A) \cap B \neq \varnothing \equiv A \cap BV^* \neq \varnothing$ *and* $\mathbf{suff}(A) \cap B \neq \varnothing \equiv A \cap V^*B \neq \varnothing$. □

Property A.5 *Let* $A, B \subseteq V^*$ *and* $a \in V$. *Then*

i. $V^*A \cap V^*B \neq \varnothing \equiv V^*A \cap B \neq \varnothing \vee A \cap V^*B \neq \varnothing$

ii. $V^*aA \cap V^*B \neq \varnothing \equiv V^*aA \cap B \neq \varnothing \vee A \cap V^*B \neq \varnothing$

iii. $V^*A \cap V^*B \neq \varnothing \equiv V^*A \cap B \neq \varnothing \vee A \cap V^+B \neq \varnothing$

□

Property A.6 *For* $x, y \in \mathbf{suff}(P)$ *and* $y \neq \varepsilon$ *we have*

$$x <_p y \equiv x \leq_p msp(y).$$

Proof
Let $x, y \in \mathbf{suff}(P)$ and $y \neq \varepsilon$. We derive

$x <_p y$

$=$ $\{$ definition of $<_p$ and **pref**, $x \in \mathbf{suff}(P)$ $\}$

$x \in \mathbf{pref}(y) \setminus \{y\} \cap \mathbf{suff}(P)$

$\Rightarrow$ $\{$ $\mathbf{pref}(y) \setminus \{y\} \cap \mathbf{suff}(P)$ is nonempty ($y \neq \varepsilon$), finite and linearly ordered w.r.t. $\leq_p$ $\}$

$x \leq_p (\mathbf{MAX}_{\leq_p}\, w : w \in \mathbf{pref}(y) \setminus \{y\} \cap \mathbf{suff}(P) : w)$

$=$ $\{$ $y \neq \varepsilon$, definition of msp $\}$

$x \leq_p msp(y)$

$\Rightarrow$ $\{$ $y \neq \varepsilon$, $msp(y) <_p y$ (by definition of msp), transitivity of $<_p$ $\}$

$x <_p y$

□

B Algorithm and problem details

In this appendix we list the algorithm and problem details introduced in this paper with a short description.

P Examine prefixes of a given string in any order.

S Examine suffixes of a given string in any order.

+ Examine the strings from a given set in order of increasing length (this program detail can only be applied after, for instance, program details P and S).

RT Usage of the transition function of the reverse trie corresponding to the set of keywords to check whether a string, that is a suffix of some keyword, preceded by a symbol is again a suffix of some keyword.

SSD Allow any shift distance at least one that is safe, i.e. that does not cause the omission of any matches.

NLAU No lookahead at the symbols of the unscanned part of the input string when computing a safe shift distance.

OLAU One symbol lookahead at the unscanned part of the input string when computing a safe shift distance.

OPT When computing a safe shift distance use the recognized suffix and only the immediately preceding (mismatching) symbol, strictly coupled.

NLA When computing a safe shift distance do not look at the symbols preceding the recognized suffix.

BMCW When computing a safe shift distance on the one hand use the recognized suffix and the fact that the symbol preceding it is mismatching, and on the other hand, but strictly independent, the identity of that symbol.

BM Lessen the contribution of the symbol preceding the recognized suffix to the shift distance in case it does not occur in any keyword.

CW When computing a safe shift distance do not use the fact that the symbol preceding the recognized suffix is mismatching (use the recognized suffix and the symbol preceding it independently).

OKW The set of keywords contains only one keyword (in contrast to the preceding program details this is a problem detail).

References

[AC75] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, June 1975.

[Aho90] Alfred V. Aho. Algorithms for finding patterns in strings. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 5, pages 255–300. North-Holland, Amsterdam, 1990.

[BM77] Robert S. Boyer and J. Strother Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, October 1977.

[Bro83] Manfred Broy. Program construction by transformations: a family tree of sorting programs. In A. W. Biermann and G. Guiho, editors, *Computer Program Synthesis Methodologies*, volume 95 of *NATO Advanced Study Institutes Series, Series C: Mathematical and Physical Sciences*, pages 1–49. Reidel, 1983.

[CW79a] Beate Commentz-Walter. A string matching algorithm fast on the average. In H.A. Maurer, editor, *Proceedings 6th International Colloquium on Automata, Languages and Programming*, volume 71 of *Lecture Notes in Computer Science*, pages 118–132. Springer, July 1979.

[CW79b] Beate Commentz-Walter. A string matching algorithm fast on the average. Technical Report TR 79.09.007, IBM-Germany, Scientific Center Heidelberg, September 1979.

[Dar78] J. Darlington. A synthesis of several sorting algorithms. *Acta Informatica*, 11:1–30, 1978.

[Dij76] E.W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[Fre60] E. Fredkin. Trie memory. *Communications of the ACM*, 3(9):490–499, 1960.

[FS93] Jang-Jong Fan and Key-Yih Su. An efficient algorithm for matching multiple patterns. *IEEE Transactions on Knowledge and Data Engineering*, 5(2):339–351, April 1993.

[FS94] Jang-Jong Fan and Key-Yih Su. An efficient algorithm for matching multiple patterns. In Jun-ichi Aoe, editor, *Computer Algorithms: String Pattern Matching Strategies.* IEEE Computer Society Press, 1994.

[HS91] Andrew Hume and Daniel M. Sunday. Fast string searching. *Software—Practice and Experience*, 21(11):1221–1248, November 1991.

[Jon83] H.B.M. Jonkers. *Abstraction, specification and implementation techniques, with an application to garbage collection.* Number 166 in Mathematical Centre Tracts. Mathematisch Centrum, Amsterdam, 1983.

[KMP77] Donald E. Knuth, James H. Morris, Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, June 1977.

[Mar90] A.J.J.M. Marcelis. On the classification of attribute evaluation algorithms. *Science of Computer Programming*, 14:1–24, 1990.

[Ryt80] Wojciech Rytter. A correct preprocessing algorithm for Boyer-Moore string-searching. *SIAM Journal on Computing*, 9(3):509–512, August 1980.

[Wat94] Bruce W. Watson. The performance of single-keyword and multiple-keyword pattern matching algorithms. Computing Science Notes 94/19, Eindhoven University of Technology, Eindhoven, the Netherlands, April 1994. The report and implementations are available at URL `ftp://ftp.win.tue.nl/pub/techreports/pi/pattm/bench/`.

[WZ92] Bruce W. Watson and Gerard Zwaan. A taxonomy of keyword pattern matching algorithms. Computing Science Notes 92/27, Eindhoven University of Technology, Eindhoven, the Netherlands, December 1992. The report is available at URL `ftp://ftp.win.tue.nl/pub/techreports/pi/pattm/taxonomy/1st.edition/pattm.ps.gz`.

[WZ93] Bruce W. Watson and Gerard Zwaan. A taxonomy of keyword pattern matching algorithms. In H.A. Wijshoff, editor, *Proceedings Computing Science in the Netherlands 93*, pages 25–39. SION, Stichting Mathematisch Centrum, November 1993.

Computing Science Reports **Department of Mathematics and Computing Science**
Eindhoven University of Technology

In this series appeared:

93/01	R. van Geldrop	Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.
93/02	T. Verhoeff	A continuous version of the Prisoner's Dilemma, p. 17
93/03	T. Verhoeff	Quicksort for linked lists, p. 8.
93/04	E.H.L. Aarts J.H.M. Korst P.J. Zwietering	Deterministic and randomized local search, p. 78.
93/05	J.C.M. Baeten C. Verhoef	A congruence theorem for structured operational semantics with predicates, p. 18.
93/06	J.P. Veltkamp	On the unavoidability of metastable behaviour, p. 29
93/07	P.D. Moerland	Exercises in Multiprogramming, p. 97
93/08	J. Verhoosel	A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.
93/09	K.M. van Hee	Systems Engineering: a Formal Approach Part I: System Concepts, p. 72.
93/10	K.M. van Hee	Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.
93/11	K.M. van Hee	Systems Engineering: a Formal Approach Part III: Modeling Methods, p. 101.
93/12	K.M. van Hee	Systems Engineering: a Formal Approach Part IV: Analysis Methods, p. 63.
93/13	K.M. van Hee	Systems Engineering: a Formal Approach Part V: Specification Language, p. 89.
93/14	J.C.M. Baeten J.A. Bergstra	On Sequential Composition, Action Prefixes and Process Prefix, p. 21.
93/15	J.C.M. Baeten J.A. Bergstra R.N. Bol	A Real-Time Process Logic, p. 31.
93/16	H. Schepers J. Hooman	A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27
93/17	D. Alstein P. van der Stok	Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.
93/18	C. Verhoef	A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.
93/19	G-J. Houben	The Design of an Online Help Facility for ExSpect, p.21.
93/20	F.S. de Boer	A Process Algebra of Concurrent Constraint Programming, p. 15.
93/21	M. Codish D. Dams G. Filé M. Bruynooghe	Freeness Analysis for Logic Programs - And Correctness, p. 24
93/22	E. Poll	A Typechecker for Bijective Pure Type Systems, p. 28.
93/23	E. de Kogel	Relational Algebra and Equational Proofs, p. 23.
93/24	E. Poll and Paula Severi	Pure Type Systems with Definitions, p. 38.
93/25	H. Schepers and R. Gerth	A Compositional Proof Theory for Fault Tolerant Real-Time Distributed Systems, p. 31.
93/26	W.M.P. van der Aalst	Multi-dimensional Petri nets, p. 25.
93/27	T. Kloks and D. Kratsch	Finding all minimal separators of a graph, p. 11.
93/28	F. Kamareddine and R. Nederpelt	A Semantics for a fine λ-calculus with de Bruijn indices, p. 49.
93/29	R. Post and P. De Bra	GOLD, a Graph Oriented Language for Databases, p. 42.
93/30	J. Deogun T. Kloks D. Kratsch H. Müller	On Vertex Ranking for Permutation and Other Graphs, p. 11.

93/31	W. Körver	Derivation of delay insensitive and speed independent CMOS circuits, using directed commands and production rule sets, p. 40.
93/32	H. ten Eikelder and H. van Geldrop	On the Correctness of some Algorithms to generate Finite Automata for Regular Expressions, p. 17.
93/33	L. Loyens and J. Moonen	ILIAS, a sequential language for parallel matrix computations, p. 20.
93/34	J.C.M. Baeten and J.A. Bergstra	Real Time Process Algebra with Infinitesimals, p.39.
93/35	W. Ferrer and P. Severi	Abstract Reduction and Topology, p. 28.
93/36	J.C.M. Baeten and J.A. Bergstra	Non Interleaving Process Algebra, p. 17.
93/37	J. Brunekreef, J-P. Katoen, R. Koymans, S. Mauw	Design and Analysis of Dynamic Leader Election Protocols in Broadcast Networks, p. 73.
93/38	C. Verhoef	A general conservative extension theorem in process algebra, p. 17.
93/39	W.P.M. Nuijten, E.H.L. Aarts, D.A.A. van Erp Taalman Kip, K.M. van Hee	Job Shop Scheduling by Constraint Satisfaction, p. 22.
93/40	P.D.V. van der Stok, M.M.M.P.J. Claessen, D. Alstein	A Hierarchical Membership Protocol for Synchronous Distributed Systems, p. 43.
93/41	A. Bijlsma	Temporal operators viewed as predicate transformers, p. 11.
93/42	P.M.P. Rambags	Automatic Verification of Regular Protocols in P/T Nets, p. 23.
93/43	B.W. Watson	A taxomomy of finite automata construction algorithms, p. 87.
93/44	B.W. Watson	A taxonomy of finite automata minimization algorithms, p. 23.
93/45	E.J. Luit, J.M.M. Martin	A precise clock synchronization protocol,p.
93/46	T. Kloks, D. Kratsch, J. Spinrad	Treewidth and Patwidth of Cocomparability graphs of Bounded Dimension, p. 14.
93/47	W. v.d. Aalst, P. De Bra, G.J. Houben, Y. Komatzky	Browsing Semantics in the "Tower" Model, p. 19.
93/48	R. Gerth	Verifying Sequentially Consistent Memory using Interface Refinement, p. 20.
94/01	P. America, M. van der Kammen, R.P. Nederpelt, O.S. van Roosmalen, H.C.M. de Swart	The object-oriented paradigm, p. 28.
94/02	F. Kamareddine, R.P. Nederpelt	Canonical typing and Π-conversion, p. 51.
94/03	L.B. Hartman, K.M. van Hee	Application of Marcov Decision Processe to Search Problems, p. 21.
94/04	J.C.M. Baeten, J.A. Bergstra	Graph Isomorphism Models for Non Interleaving Process Algebra, p. 18.
94/05	P. Zhou, J. Hooman	Formal Specification and Compositional Verification of an Atomic Broadcast Protocol, p. 22.
94/06	T. Basten, T. Kunz, J. Black, M. Coffin, D. Taylor	Time and the Order of Abstract Events in Distributed Computations, p. 29.
94/07	K.R. Apt, R. Bol	Logic Programming and Negation: A Survey, p. 62.
94/08	O.S. van Roosmalen	A Hierarchical Diagrammatic Representation of Class Structure, p. 22.
94/09	J.C.M. Baeten, J.A. Bergstra	Process Algebra with Partial Choice, p. 16.

94/10	T. verhoeff	The testing Paradigm Applied to Network Structure. p. 31.
94/11	J. Peleska C. Huizing C. Petersohn	A Comparison of Ward & Mellor's Transformation Schema with State- & Activitycharts, p. 30.
94/12	T. Kloks D. Kratsch H. Müller	Dominoes, p. 14.
94/13	R. Seljée	A New Method for Integrity Constraint checking in Deductive Databases, p. 34.
94/14	W. Peremans	Ups and Downs of Type Theory, p. 9.
94/15	R.J.M. Vaessens E.H.L. Aarts J.K. Lenstra	Job Shop Scheduling by Local Search, p. 21.
94/16	R.C. Backhouse H. Doornbos	Mathematical Induction Made Calculational, p. 36.
94/17	S. Mauw M.A. Reniers	An Algebraic Semantics of Basic Message Sequence Charts, p. 9.
94/18	F. Kamareddine R. Nederpelt	Refining Reduction in the Lambda Calculus, p. 15.
94/19	B.W. Watson	The performance of single-keyword and multiple-keyword pattern matching algorithms, p. 46.
94/20	R. Bloo F. Kamareddine R. Nederpelt	Beyond β-Reduction in Church's $\lambda\rightarrow$, p. 22.
94/21	B.W. Watson	An introduction to the Fire engine: A C++ toolkit for Finite automata and Regular Expressions.
94/22	B.W. Watson	The design and implementation of the FIRE engine: A C++ toolkit for Finite automata and regular Expressions.
94/23	S. Mauw and M.A. Reniers	An algebraic semantics of Message Sequence Charts, p. 43.
94/24	D. Dams O. Grumberg R. Gerth	Abstract Interpretation of Reactive Systems: Abstractions Preserving $\forall$CTL*, $\exists$CTL* and CTL*, p. 28.
94/25	T. Kloks	$K_{1,3}$-free and W_4-free graphs, p. 10.
94/26	R.R. Hoogerwoord	On the foundations of functional programming: a programmer's point of view, p. 54.
94/27	S. Mauw and H. Mulder	Regularity of BPA-Systems is Decidable, p. 14.
94/28	C.W.A.M. van Overveld M. Verhoeven	Stars or Stripes: a comparative study of finite and transfinite techniques for surface modelling, p. 20.
94/29	J. Hooman	Correctness of Real Time Systems by Construction, p. 22.
94/30	J.C.M. Baeten J.A. Bergstra Gh. Ştefanescu	Process Algebra with Feedback, p. 22.
94/31	B.W. Watson R.E. Watson	A Boyer-Moore type algorithm for regular expression pattern matching, p. 22.
94/32	J.J. Vereijken	Fischer's Protocol in Timed Process Algebra, p. 38.
94/33	T. Laan	A formalization of the Ramified Type Theory, p.40.
94/34	R. Bloo F. Kamareddine R. Nederpelt	The Barendregt Cube with Definitions and Generalised Reduction, p. 37.
94/35	J.C.M. Baeten S. Mauw	Delayed choice: an operator for joining Message Sequence Charts, p. 15.
94/36	F. Kamareddine R. Nederpelt	Canonical typing and Π-conversion in the Barendregt Cube, p. 19.
94/37	T. Basten R. Bol M. Voorhoeve	Simulating and Analyzing Railway Interlockings in ExSpect, p. 30.
94/38	A. Bijlsma C.S. Scholten	Point-free substitution, p. 10.

94/39	A. Blokhuis, T. Kloks	On the equivalence covering number of splitgraphs, p. 4.
94/40	D. Alstein	Distributed Consensus and Hard Real-Time Systems, p. 34.
94/41	T. Kloks, D. Kratsch	Computing a perfect edge without vertex elimination ordering of a chordal bipartite graph, p. 6.
94/42	J. Engelfriet, J.J. Vereijken	Concatenation of Graphs, p. 7.
94/43	R.C. Backhouse, M. Bijsterveld	Category Theory as Coherently Constructive Lattice Theory: An Illustration, p. 35.
94/44	E. Brinksma, J. Davies, R. Gerth, S. Graf, W. Janssen, B. Jonsson, S. Katz, G. Lowe, M. Poel, A. Pnueli, C. Rump, J. Zwiers	Verifying Sequentially Consistent Memory, p. 160
94/45	G.J. Houben	Tutorial voor de ExSpect-bibliotheek voor "Administratieve Logistiek", p. 43.
94/46	R. Bloo, F. Kamareddine, R. Nederpelt	The λ-cube with classes of terms modulo conversion, p. 16.
94/47	R. Bloo, F. Kamareddine, R. Nederpelt	On Π-conversion in Type Theory, p. 12.
94/48	Mathematics of Program Construction Group	Fixed-Point Calculus, p. 11.
94/49	J.C.M. Baeten, J.A. Bergstra	Process Algebra with Propositional Signals, p. 25.
94/50	H. Geuvers	A short and flexible proof of Strong Normalazation for the Calculus of Constructions, p. 27.
94/51	T. Kloks, D. Kratsch, H. Müller	Listing simplicial vertices and recognizing diamond-free graphs, p. 4.
94/52	W. Penczek, R. Kuiper	Traces and Logic, p. 81
94/53	R. Gerth, R. Kuiper, D. Peled, W. Penczek	A Partial Order Approach to Branching Time Logic Model Checking, p. 20.
95/01	J.J. Lukkien	The Construction of a small CommunicationLibrary, p.16.
95/02	M. Bezem, R. Bol, J.F. Groote	Formalizing Process Algebraic Verifications in the Calculus of Constructions, p.49.
95/03	J.C.M. Baeten, C. Verhoef	Concrete process algebra, p. 134.
95/04	J. Hidders	An Isotopic Invariant for Planar Drawings of Connected Planar Graphs, p. 9.
95/05	P. Severi	A Type Inference Algorithm for Pure Type Systems, p.20.
95/06	T.W.M. Vossen, M.G.A. Verhoeven, H.M.M. ten Eikelder, E.H.L. Aarts	A Quantitative Analysis of Iterated Local Search, p.23.
95/07	G.A.M. de Bruyn, O.S. van Roosmalen	Drawing Execution Graphs by Parsing, p. 10.
95/08	R. Bloo	Preservation of Strong Normalisation for Explicit Substitution, p. 12.
95/09	J.C.M. Baeten, J.A. Bergstra	Discrete Time Process Algebra, p. 20
95/10	R.C. Backhouse, R. Verhoeven, O. Weber	Math∫pad: A System for On-Line Prepararation of Mathematical Documents, p. 15

95/11	R. Seljée	Deductive Database Systems and integrity constraint checking, p. 36.
95/12	S. Mauw and M. Reniers	Empty Interworkings and Refinement Semantics of Interworkings Revised, p. 19.