

A Brief Overview of Dead-Zone Pattern Matching Algorithms

Miznah Alshammary, Mai Alzamel, Costas Iliopoulos, Richard E. Watson, Bruce W. Watson

▶ **To cite this version:**

Miznah Alshammary, Mai Alzamel, Costas Iliopoulos, Richard E. Watson, Bruce W. Watson. A Brief Overview of Dead-Zone Pattern Matching Algorithms. 15th IFIP International Conference on Artificial Intelligence Applications and Innovations (AIAI), May 2019, Hersonissos, Greece. pp.208-218, 10.1007/978-3-030-19909-8_18 . hal-02363837

HAL Id: hal-02363837
https://inria.hal.science/hal-02363837

Submitted on 14 Nov 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution| 4.0 International License

A brief overview of dead-zone pattern matching algorithms

Miznah Alshammary, Mai Alzamel, Costas Iliopoulos,
Richard E. Watson, Bruce W. Watson

{miznah.alshammary, mai.alzamel, c.iliopoulos}@kcl.ac.uk
bruce@fastar.org

Abstract

Within the last decades, the *dead-zone* algorithms have emerged as being highly performant on certain types of data. Such algorithms solve the keyword exact matching problem over strings, though extensions to trees and two-dimensional data have also been devised. In this short paper, we give an overview of such algorithms.

1 Introduction and related work

In this paper, we present a new family of algorithms solving the single keyword string pattern matching problem. This particular pattern matching problem can be described as follows: given an input string S and a keyword p, find all occurrences of p as a continuous substring of S. The field of string pattern matching is generally well-studied (some thought it to be exhausted by the end of the 1970's), however, it continues to yield new and exciting algorithms, as was seen in annual conferences such as *Combinatorial Pattern Matching* and *Stringology*. In [8] (a dissertation by the last author of this paper), a taxonomy of existing algorithms was presented, along with a number of new algorithms. Any given algorithm may have more than one possible derivation, leading to different classifications of the algorithm in a taxonomy[1]. Many of the new derivations can prove to be more than just an educational curiosity, possibly leading to interesting new families of algorithms. This paper presents one such family, with some new algorithms and also some alternative derivations of existing ones. The algorithms presented in this paper have been extended to handle some more complex pattern matching problems, including multiple keyword pattern matching, regular pattern matching and multi-dimensional pattern matching. For some recent examples of this, see [12, 10, 11, 9].

2 Mathematical preliminaries

While most of the mathematical notation and definitions used in this paper are is described in detail in [4], here we present some more specific notations. Indexing within strings begins at 0, as in the C and C++ programming languages. We use ranges of integers throughout the paper which are defined by (for integers i and j):

$$[i, j) = \{\, k \mid i \leq k < j \,\}$$

$$(i, j] = \{\, k \mid i < k \leq j \,\}$$

$$[i, j] = [i, j) \cup (i, j]$$

$$(i, j) = [i, j) \cap (i, j]$$

In addition, we define a *permutation* of a set of integers to be a bijective mapping of those integers onto themselves.

[1] This is precisely what happened with the Boyer-Moore type algorithms as presented in the dissertation [8].

3 The problem and a first algorithm

Before giving the problem specification (in the form of a postcondition to the algorithms), we define a predicate which will make the postcondition and algorithms easier to read. Keyword p (with the restriction that $p \neq \varepsilon$, where ε is the empty string) is said to *match* at position j in input string S if $p = S_{j \cdots j+|p|-1}$; this is restated in the following predicate:

Definition 3.1 (Predicate *Matches*): We define predicate *Matches* as

$$Matches(S, p, j) \equiv (p = S_{j \cdots j+|p|-1})$$

□

The pattern matching problem requires us to compute the set of all matches of keyword p in input string S. We register the matches as the set O (for "output") of all indices j (in S) such that $Matches(S, p, j)$ holds.

Definition 3.2 (Single keyword pattern matching problem): Given a common alphabet V, input string S, and pattern keyword p, the problem is defined using postcondition PM:

$$O = \{\, j \mid j \in [0, |S|) \wedge Matches(S, p, j) \,\}$$

Note that this postcondition implicitly depends upon S and p, even though we do not make that explicit. □

We can now present a nondeterministic algorithm which keeps track of the set of possible indices (in S) at which a match might still be found (indices at which we have not yet checked for a match). This set is known as the *live zone*. Those indices not in the live zone are said to be in the *dead zone*. This give us our first algorithm (presented in Dijkstra's pseudocode [3, 1, 6]).

Algorithm 3.3:

$live, dead := [0, |S|), \emptyset;$
$O := \emptyset;$
{ invariant: $live \cup dead = [0, |S|) \wedge live \cap dead = \emptyset$
$\wedge\ O = \{\, j \mid j \in dead \wedge Matches(S, p, j) \,\}$ }
do $live \neq \emptyset \rightarrow$
  let $j : j \in live;$
  $live, dead := live \setminus \{j\}, dead \cup \{j\};$
  if $Matches(S, p, j) \rightarrow O := O \cup \{j\}$
  [] $\neg Matches(S, p, j) \rightarrow$ **skip**
  fi
od{ postcondition: PM }

□

The invariant specifies that *live* and *dead* are disjoint and account for all indices in S; additionally, any match at an element of *dead* has already been registered. Thanks to this relationship between *live* and *dead*, we could have written the repetition condition $live \neq \emptyset$ as $dead \neq [0, |S|)$, and the j selection condition $j \in live$ as $j \notin dead$. It should be easy to see that the invariant and the termination condition of the repetition implies the postcondition — yielding a correct algorithm. Note that this algorithm is highly over-specified by keeping both variables *live* and *dead* to represent the live and dead zones, respectively. For efficiency, only one of these sets would normally be kept, as is seen in [10, 11, 9].

Some of the rightmost positions in S cannot possibly accommodate matches — no match can be found at any point $j \in [|S| - |p| + 1, |S|)$ since $|S_{j \cdots |S|-1}| \leq |S_{|S|-|p|+1 \cdots |S|-1}| < |p|$ (the match

attempt begins too close to the end of S for p to fit). For this reason, we safely change the initializations of *live* and *dead* to

$$\mathit{live}, \mathit{dead} := [0, |S| - |p|], [|S| - |p| + 1, |S|)$$

In the next section, give a more deterministic (more realistically implemented) version of the last algorithm.

4 A more deterministic algorithm

In the last algorithm, our comparison of p with $S_{j \cdots j+|p|-1}$ is embedded within the evaluation of predicate *Matches*. In this section, we make this comparison explicit. We begin by noting that $p = S_{j \cdots j+|p|-1}$ is equivalent to comparing the individual symbols p_k of p with the corresponding symbols S_{j+k} of S (for $k \in [0, |p|)$). In fact, we can consider the symbols in any order whatsoever. To determine the order in which they will be considered, we introduce *match orders*:

Definition 4.1 (Match order): We define a match order *mo* as a permutation on $[0, |p|)$. □

Using *mo*, we can restate our match predicate.

Property 4.2 (Predicate *Matches*): Predicate *Matches* is restated as

$$\mathit{Matches}(S, p, j) \equiv (\forall\ i : i \in [0, |p|) : p_{mo(i)} = S_{j+mo(i)})$$

□

This rendition of the predicate will be evaluated by a repetition which uses a new integer variable i to step from 0 to $|p| - 1$, comparing $p_{mo(i)}$ to the corresponding symbol of S. As i increases, the repetition has the following invariant:

$$(\forall\ k : k \in [0, i) : p_{mo(k)} = S_{j+mo(k)})$$

and terminates as early as possible.

In the following algorithm, we use the match order *mo*, the new repetition and our previous optimization to the initializations of *dead* and *live*.

Algorithm 4.3:

$\mathit{live}, \mathit{dead} := [0, |S| - |p|], [|S| - |p| + 1, |S|)$;
$O := \emptyset$;
{ invariant: $\mathit{live} \cup \mathit{dead} = [0, |S|) \wedge \mathit{live} \cap \mathit{dead} = \emptyset$
$\quad \wedge O = \{\, j \mid j \in \mathit{dead} \wedge \mathit{Matches}(S, p, j) \,\}$ }
do $\mathit{live} \neq \emptyset \rightarrow$
$\quad$ **let** $j : j \in \mathit{live}$;
$\quad \mathit{live}, \mathit{dead} := \mathit{live} \setminus \{j\}, \mathit{dead} \cup \{j\}$;
$\quad i := 0$;
$\quad$ { invariant: $(\forall\ k : k \in [0, i) : p_{mo(k)} = S_{j+mo(k)})$ }
$\quad$ **do** $i < |p|$ **cand** $p_{mo(i)} = S_{j+mo(i)} \rightarrow$
$\quad\quad i := i + 1$
$\quad$ **od**;
$\quad$ { postcondition: $(\forall\ k : k \in [0, i) : p_{mo(k)} = S_{j+mo(k)})$
$\quad\quad \wedge (i < |p| \Rightarrow p_{mo(i)} \neq S_{j+mo(i)})$ }
$\quad$ **if** $i = |p| \rightarrow O := O \cup \{j\}$
$\quad [\!]\ \ i < |p| \rightarrow$ **skip**
$\quad$ **fi**
od{ postcondition: PM }

□

The operator P **cand** Q appears in the guard of the inner loop of the above algorithm. This operator is similar to conjunction $P \wedge Q$ except that if the first conjunct evaluates to *false* then the second conjunct is not even evaluated. This proves to be a useful property in cases such as the loop guard since, if the first conjunct ($i < |p|$) is *false* (hence $i >= |p|$, and indeed $i = |p|$), then the term $mo(i)$ appearing in the second conjunct is not even defined. Note that the implication within the second conjunct of the loop postcondition is derived from the loop guard, forcing the implication operator to be conditional as well (that is, if $i < |p|$ is determined to be *false*, then $p_{mo(i)} \neq S_{j+mo(i)}$ is not even evaluated).

5 Reusing match information

On each iteration of the outer repetition, index j is chosen and eliminated from the live zone in the statement:

$$live, dead := live \setminus \{j\}, dead \cup \{j\}$$

The performance of the algorithm can be improved if we remove more than just j in some of the iterations. To do this, we can use some of the match information, such as i, which indicates how far through mo the match attempt proceeded before finding a mismatching symbol. The information most readily available is the postcondition of the inner repetition:

$$(\forall\ k : k \in [0, i) : p_{mo(k)} = S_{j+mo(k)}) \wedge (i < |p| \Rightarrow p_{mo(i)} \neq S_{j+mo(i)})$$

We denote this postcondition by $Result(S, p, i, j)$. Since this postcondition holds, we may be able to deduce that certain indices in S cannot possibly be the site of a match. It is such indices which we could also remove from the live zone. They are formally characterized as:

$$\{\, x \mid x \in [0, |S|) \wedge (Result(S, p, i, j) \Rightarrow \neg Matches(S, p, x)) \,\} \tag{1}$$

Determining this set at pattern matching time is inefficient and not easily implemented. We wish to derive a safe approximation of this set which can be precomputed, tabulated and indexed (at pattern matching time) by i. In order to precompute it, the approximation must be independent of j and S. We wish to find a strengthening of the range predicate since this will allow us to still remove a safe set of elements from set *live*, thanks to the property that, if $P \Rightarrow Q$ (P is a *strengthening* of Q, and Q is a *weakening* of P), then

$$\{\, x \mid P(x) \,\} \subseteq \{\, x \mid Q(x) \,\}$$

As a first step towards this approximation, we can normalize the ideal set (equation (1) above), by subtracting j from each element. The resulting characterization will be more useful for precomputation reasons:

$$\{\, x \mid x \in [-j, |S| - j) \wedge (Result(S, p, i, j) \Rightarrow \neg Matches(S, p, j + x)) \,\}$$

Note that this still depends upon j, however, it will make some of the derivation steps shown shortly in Section 5.1 easier. Because those steps are rather detailed, they are presented in isolation. Condensed, the derivation appears as:

$$\begin{array}{ll}
 & (Result(S, p, i, j) \Rightarrow \neg Matches(S, p, j + x)) \\
\Leftarrow & \quad \{\, \text{Section 5.1} \,\} \\
 & \neg((\forall\ k : k \in [0, i) \wedge mo(k) \in [x, |p| + x) : p_{mo(k)} = p_{mo(k)-x}) \\
 & \wedge (i < |p| \wedge mo(i) \in [x, |p| + x) \Rightarrow p_{mo(i)} \neq p_{mo(i)-x})) \\
\equiv & \quad \{\, \text{De Morgan's Law} \,\} \\
 & \neg(\forall\ k : k \in [0, i) \wedge mo(k) \in [x, |p| + x) : p_{mo(k)} = p_{mo(k)-x}) \\
 & \vee \neg(i < |p| \wedge mo(i) \in [x, |p| + x) \Rightarrow p_{mo(i)} \neq p_{mo(i)-x}) \\
\equiv & \quad \{\, \text{De Morgan's Law on the universal quantification} \,\}
\end{array}$$

$$
\begin{array}{ll}
 & (\exists\ k : k \in [0,i) \wedge mo(k) \in [x, |p|+x) : p_{mo(k)} \neq p_{mo(k)-x}) \\
 & \vee \neg(i < |p| \wedge mo(i) \in [x, |p|+x) \Rightarrow p_{mo(i)} \neq p_{mo(i)-x}) \\
\equiv & \quad \{\text{ De Morgan's Law on the implication, which is conditional }\} \\
 & (\exists\ k : k \in [0,i) \wedge mo(k) \in [x, |p|+x) : p_{mo(k)} \neq p_{mo(k)-x}) \\
 & \vee (i \geq |p| \vee mo(i) \notin [x, |p|+x) \ \mathbf{cor}\ \ p_{mo(i)} \neq p_{mo(i)-x}) \\
\equiv & \quad \{\text{ define the predicate } \mathit{Approximation}(p,i,x)\ \} \\
 & \mathit{Approximation}(p,i,x)
\end{array}
$$

Note that we define the predicate $\mathit{Approximation}(p,i,x)$ which depends only on p and i and hence can be precomputed and tabulated. It should be mentioned that this is one of several possible useful strengthenings which could be derived. We could even have used the strongest predicate, *false*, instead of $\mathit{Approximation}(p,i,x)$. This would yield the empty set, $\emptyset$, to be removed from *live* in addition to j (as in the previous algorithm).

We can derive a smaller range predicate of x for which we have to check if $\mathit{Approximation}(p,i,x)$ holds. Notice that choosing and x such that $[x, |p|+x) \cap [0, |p|) = \emptyset$ has two important consequences:

- The range of the quantification in first conjunct of $\mathit{Approximation}(p,i,x)$ is empty (hence this conjunct is *true*, by the definition of universal quantification with an empty range).
- The range condition of the second conjunct (the 'implicator') is *false* — hence the whole of the second conjunct is *true* since *false* $\Rightarrow P$ for all predicates P.

With this choice of x, we see that predicate $\mathit{Approximation}(p,i,x)$ always evaluates to *false*, in which case we need not even consider values of x such that $[x, |p|+x) \cap [0, |p|) = \emptyset$. As a result, we characterize those x for which $[x, |p|+x) \cap [0, |p|) \neq \emptyset$ as follows:

$$
\begin{array}{ll}
 & [x, |p|+x) \cap [0, |p|) \neq \emptyset \\
\equiv & \quad \{\text{ for integers } u,v,y,z\text{: } [u,v) \cap [y,z) = \emptyset \equiv (v-1 < y \vee u > z-1)\ \} \\
 & \neg(|p| + x - 1 < 0 \vee x > |p| - 1) \\
\equiv & \quad \{\text{ arithmetic }\} \\
 & \neg(x < 1 - |p| \vee x > |p| - 1) \\
\equiv & \quad \{\text{ De Morgan's Law }\} \\
 & x \geq 1 - |p| \wedge x \leq |p| - 1 \\
\equiv & \quad \{\text{ definition of ranges }\} \\
 & x \in [1 - |p|, |p| - 1]
\end{array}
$$

Clearly we can use the restriction $x \in [1-|p|, |p|-1]$. Intuitively (and information theoretically), we know that there must be such a range restriction since we can not possibly know from a current match attempt whether or not we will find a match of p in S more than $|p|$ symbols away.

Finally we have the following algorithm (in which we have added the additional update of *live* and *dead* below the inner repetition). Note that we introduce the set *nogood* to accumulate the indices for which $\mathit{Approximation}(p,i,x)$ holds. Also note that we renormalize the set *nogood* by adding j to each of its members and ensuring that it is within the valid range of indices, $[0, |S|)$.

Algorithm 5.1:

$\mathit{live}, \mathit{dead} := [0, |S| - |p|], [|S| - |p| + 1, |S|)$;
$O := \emptyset$;
{ invariant: $\mathit{live} \cup \mathit{dead} = [0, |S|) \wedge \mathit{live} \cap \mathit{dead} = \emptyset$
$\wedge\ O = \{\ l \mid l \in \mathit{dead} \wedge \mathit{Matches}(S,p,l)\ \}$ }
do $\mathit{live} \neq \emptyset \rightarrow$
let $j : j \in \mathit{live}$;

$live, dead := live \setminus \{j\}, dead \cup \{j\}$;
$i := 0$;
{ invariant: $(\forall\ k : k \in [0, i) : p_{mo(k)} = S_{j+mo(k)})$ }
do $i < |p|$ **cand** $p_{mo(i)} = S_{j+mo(i)} \rightarrow$
  $i := i + 1$
od;
{ postcondition: $Result(S, p, i, j)$ }
if $i = |p| \rightarrow O := O \cup \{j\}$
$[\!]$ $i < |p| \rightarrow$ **skip**
fi;
$nogood := (\{\, x \mid x \in [1 - |p|, |p| - 1] \wedge Approximation(p, i, x) \,\} + j) \cap [0, |S|)$;
$live := live \setminus nogood$;
$dead := dead \cup nogood$
od{ postcondition: PM }

□

5.1 Range predicate strengthening

Here, we present the derivation of a strengthening of the range predicate

$$Result(S, p, i, j) \Rightarrow \neg Matches(S, p, j + x)$$

Being more comfortable with weakening steps, we begin with the negation of part of the above range predicate, and proceed by weakening:

$\neg(Result(S, p, i, j) \Rightarrow \neg Matches(S, p, j + x))$
$\equiv$ { definition of $\Rightarrow$ }
$\neg(\neg Result(S, p, i, j) \vee \neg Matches(S, p, j + x))$
$\equiv$ { De Morgan's Law }
$Result(S, p, i, j) \wedge Matches(S, p, j + x))$
$\equiv$ { definition of *Result* and *Matches* }
$(\forall\ k : k \in [0, i) : p_{mo(k)} = S_{j+mo(k)}) \wedge (i < |p| \Rightarrow p_{mo(i)} \neq S_{j+mo(i)})$
$\wedge\ (\forall\ k : k \in [0, |p|) : p_{mo(k)} = S_{mo(k)+j+x})$
$\equiv$ { change range predicate in second quantification and definition of mo }
$(\forall\ k : k \in [0, i) : p_{mo(k)} = S_{j+mo(k)}) \wedge (i < |p| \Rightarrow p_{mo(i)} \neq S_{j+mo(i)})$
$\wedge\ (\forall\ k : mo(k) \in [0, |p|) : p_{mo(k)} = S_{mo(k)+j+x})$
$\Rightarrow$ { change dummy $(mo(k') = mo(k) + x)$, restrict range }
$(\forall\ k : k \in [0, i) : p_{mo(k)} = S_{j+mo(k)}) \wedge (i < |p| \Rightarrow p_{mo(i)} \neq S_{j+mo(i)})$
$\wedge\ (\forall\ k' : mo(k') - x \in [0, |p|) : p_{mo(k')-x} = S_{mo(k')+j})$
$\equiv$ { simplify range predicate of second quantification }
$(\forall\ k : k \in [0, i) : p_{mo(k)} = S_{j+mo(k)}) \wedge (i < |p| \Rightarrow p_{mo(i)} \neq S_{j+mo(i)})$
$\wedge\ (\forall\ k' : mo(k') \in [x, |p| + x) : p_{mo(k')-x} = S_{mo(k')+j})$
$\Rightarrow$ { one-point rule: second conjunct and second quantification }
$(\forall\ k : k \in [0, i) : p_{mo(k)} = S_{j+mo(k)})$
$\wedge\ ((i < |p| \wedge mo(i) \in [x, |p| + x)) \Rightarrow p_{mo(i)} \neq p_{mo(i)-x})$
$\wedge\ (\forall\ k' : mo(k') \in [x, |p| + x) : p_{mo(k')-x} = S_{mo(k')+j})$
$\Rightarrow$ { combine two quantifications and remove dependency on S and transitivity of $=$ }
$(\forall\ k : k \in [0, i) \wedge mo(k) \in [x, |p| + x) : p_{mo(k)} = p_{mo(k)-x})$
$\wedge\ ((i < |p| \wedge mo(i) \in [x, |p| + x)) \Rightarrow p_{mo(i)} \neq p_{mo(i)-x})$

6 Choosing j from the live zone

In this section, we discuss strategies for choosing the index j (from the live zone) at which to make a match attempt. In the last algorithm, the way in which j is chosen from set *live* is nondeterministic. This leads to the situation that *live* (and, of course, *dead*) is fragmented, meaning that an implementation of the algorithm would have to maintain a set of indices for live. If we can ensure that *live* is contiguous, then an implementation would only need to keep track of the (one or two) boundary points between *live* and *dead*. There are several ways to do this, and we discuss some of them in the following subsections section. Each of these represents a particular *policy* to be used in the selection of j.

6.1 Minimal element — towards the classical Boyer-Moore algorithm

We could use the policy of always taking the minimal element of *live*. In that case, we can make some simplifications to the algorithm (which, in turn, improve the algorithm's performance):

- We need only store the minimal element of *live*, instead of sets *live* and *dead*. We use $\widehat{live}$ to denote the minimal element.
- The dead zone update could be modified as follows: we will have considered all of the positions to the left of j and so we can ignore the negative elements of the update set:

  $$\{\, x \mid x \in [1 - |p|, 0) \wedge \mathit{Approximation}(p, i, x) \,\}$$

  Indeed, we can just add the maximal element (which is still contiguously in the update set and greater than j) of the update set to $\widehat{live}$ for the new version of our new update of *live* and *dead*.

Depending upon the choice of weakening, and the choice of match order, the above policy yields variants of the classical Boyer-Moore algorithm (see [8, 2, 7]):

Algorithm 6.1:

$\widehat{live} := 0$;
$O := \emptyset$;
do $\widehat{live} \leq |S| - |p| \rightarrow$
  $j := \widehat{live}$;
  $\widehat{live} := \widehat{live} + 1$;
  $i := 0$;
  { invariant: $(\forall\ k : k \in [0, i) : p_{mo(k)} = S_{j+mo(k)})$ }
  do $i < |p|$ **cand** $p_{mo(i)} = S_{j+mo(i)} \rightarrow$
    $i := i + 1$
  od;
  { postcondition: $\mathit{Result}(S, p, i, j)$ }
  if $i = |p| \rightarrow O := O \cup \{j\}$
  [] $i < |p| \rightarrow$ **skip**
  fi;
  $\widehat{nogood} := (\mathbf{MAX}\ x : x \in [0, |p| - 1] \wedge (\forall\ h : h \in [0, x] : \mathit{Approximation}(p, i, x)) : x)$;
  $\widehat{live} := \widehat{live} + \widehat{nogood}$
od{ postcondition: PM }

□

6.2 Recursion

We could also devise a recursive version of the algorithm as a procedure. This procedure receives a contiguous range of live indices (*live*) — initially consisting of the range $[0, |S| - |p|]$.

If the set it receives is empty, the procedure immediately returns. If the set is non-empty, j is chosen so that the resulting dead zone would appear reasonably close to the middle of the current live zone[2]. This ensures that we discard as little information as possible from the *nogood* index set. After the match attempt, the procedure recursively invokes itself twice, with the two reduced live zones on either side of the new dead zone. This yields the following procedure:

Algorithm 6.2:

---

proc $mat(S, p, live, dead) \rightarrow$
  { *live* is contiguous }
  if $live = \emptyset \rightarrow$ **skip**
  [] $live \neq \emptyset \rightarrow$
    $live_low := (\mathbf{MIN}\ k : k \in live : k)$;
    $live_high := (\mathbf{MAX}\ k : k \in live : k)$;
    $j := \lfloor (live_low + live_high - |p|)/2 \rfloor$;
    $i := 0$;
    { invariant: $(\forall\ k : k \in [0, i) : p_{mo(k)} = S_{j+mo(k)})$ }
    do $i < |p|$ **cand** $p_{mo(i)} = S_{j+mo(i)} \rightarrow$
      $i := i + 1$
    od;
    { postcondition: $Result(S, p, i, j)$ }
    if $i = |p| \rightarrow O := O \cup \{j\}$
    [] $i < |p| \rightarrow$ **skip**
    fi;
    $new_dead := (\{\ x \mid x \in [1 - |p|, |p| - 1] \wedge Approximation(p, i, x)\ \} + j) \cap [0, |S|)$;
    $dead := dead \cup new_dead$;
    $mat(S, p, [live_low, (\mathbf{MIN}\ k : k \in new_dead : k)), dead)$;
    $mat(S, p, ((\mathbf{MAX}\ k : k \in new_dead : k), live_high], dead)$
  fi
corp

□

This procedure is used in the algorithm:

Algorithm 6.3:

$O := \emptyset$;
$mat(S, p, [0, |S| - |p|], [|S| - |p| + 1, |S|))$
{ postcondition: PM }

□

Naturally, for efficiency reasons, the set *live* can be represented by its minimal and maximal elements (since it is contiguous). Note that the *dead zone* need not be contiguous. This recursive algorithm is presented in [9], and with benchmarking data in [10].

[2]The algorithm given in this section makes a simple approximation by taking the middle of the live zone it receives, and subtracting $\lfloor |p|/2 \rfloor$.

7 Conclusions

We have shown that there are still many interesting algorithms to be derived within the field of single keyword pattern matching. The correctness preserving derivation of am entirely new family of such algorithms demonstrates the use of formal methods and the use of predicates, invariants, postconditions and preconditions. It is unlikely that such a family of algorithms could have be devised without the use of formal methods.

References

[1] Cohen, E. *Programming in the 1990s,* (Springer-Verlag, New York, NY, 1990).

[2] Crochemore, M. and W. Rytter. *Text Algorithms,* (Oxford University Press, Oxford, England, 1994).

[3] Dijkstra, E.W. *A discipline of programming,* (Prentice Hall, Englewood Cliffs, NJ, 1976).

[4] Gries, D. and F.B. Schneider. *A Logical Approach to Discrete Math,* (Springer-Verlag, New York, NY, 1993).

[5] Gries, D. *The Science of Programming,* (Springer-Verlag, New York, NY, 1981).

[6] Kourie, D.G. and Watson, B.W.. *The Correctness-by-Construction Approach to Programming,* (Springer-Verlag, New York, NY, 2012).

[7] Hume, S.C. and Sunday, D.. *Fast string searching*, *Software — Practice & Experience,* **21**(11) pp. 1221–1248.

[8] Watson, B.W. *Taxonomies and Toolkits of Regular Language Algorithms,* Ph.D dissertation, Faculty of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands, September 1995, ISBN 90-386-0396-7.

[9] Watson, B.W., Kourie, D.G. and Strauss, T. *A Sequential Recursive Implementation of Dead-Zone Single Keyword Pattern Matching.* In IWOCA, pp. 236–248. 2012.

[10] Mauch, M., Kourie, D.G., Watson, B.W. and Strauss, T. *Performance assessment of dead-zone single keyword pattern matching,* In Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference, pp. 59–68. 2012.

[11] Watson, B.W., Cleophas, L.G. and Kourie, D.G.. *Using Correctness-by-Construction to Derive Dead-zone Algorithms,* In Prague Stringology Conference, pp. 84–95. 2014.

[12] Daykin, J.W., Groult, R., Guesnet, Y., Lecroq, T, Lefebvre, A., Léonard, M., Mouchard, L., Prieur-Gaston, É. and Watson, B.W. *Three Strategies for the Dead-Zone String Matching Algorithm,* In Prague Stringology Conference, p. 117. 2018.