

TECHNISCHE UNIVERSITEIT EINDHOVEN  
Department of Mathematics and Computer Science

MASTER'S THESIS  
Two-dimensional pattern matching

by  
M.G.W.H. van de Rijdt

Supervisors:      dr. ir. G. Zwaan  
                         prof. dr. B.W. Watson

Eindhoven, August 2005



## **Abstract**

This thesis contains formal derivations of several two-dimensional pattern matching algorithms.

The two-dimensional pattern matching problem is to find all exact occurrences of a given two-dimensional pattern matrix within a larger matrix. Two-dimensional pattern matching is mostly applied in image processing (and image recognition in particular), although there are other applications as well.

We give a formal derivation (and correctness proof) for several known algorithms in this field, as well as a few improvements to some of these algorithms.

## **Samenvatting**

Dit afstudeerverslag bevat formele afleidingen voor enige algoritmen voor twee-dimensionale patroonherkenning.

Het probleem van twee-dimensionale patroonherkenning bestaat uit het vinden van de voorkomens van een gegeven twee-dimensionale patroonmatrix binnen een grotere matrix. Twee-dimensionale patroonherkenning wordt vooral toegepast binnen de beeldverwerking (en in het bijzonder beeldherkenning), maar er zijn ook andere toepassingen.

We geven een formele afleiding (tevens correctheidsbewijs) van verschillende bekende algoritmen die dit probleem oplossen. Daarnaast introduceren we enige verbeteringen op sommige van deze algoritmen.



## Preface

This document is my Master's Thesis, written to complete my education in "Technische Informatica" (Technical Computer Science). It is the result of my research for the Software Construction group at the Department of Mathematics and Computer Science at the Eindhoven University of Technology (TU/e), under the supervision of dr. ir. Gerard Zwaan and prof. dr. Bruce Watson.

I thank Bruce Watson, for initially suggesting the topic of two-dimensional pattern matching and involving me in the FASTAR (Finite Automata Systems – Theoretical and Applied Research) group. I would also like to thank Gerard Zwaan, for his reviews of countless draft versions of this thesis and offering many corrections and suggestions to improve this document greatly. I thank Loek Cleophas for his reviews of several early versions of this document, as well as some pointers to relevant articles. Many thanks go to my good friend Remi Bosman, for all the brainstorming and his reviews of this document's near-final drafts. I thank LaQuSo (the Laboratory for Quality Software at the TU/e), for providing me with a workspace for many months. Finally, I would like to thank my friends and family, particularly my parents, for all their support during the time of this research.

– Martijn van de Rijdt, August 2005.



# Contents

<b>0</b>	<b>Introduction</b>	<b>10</b>
<b>1</b>	<b>Preliminaries</b>	<b>12</b>
1.0	Two-dimensional arrays . . . . .	12
1.1	Composed matrices . . . . .	13
1.2	One-dimensional pattern matching . . . . .	14
<b>2</b>	<b>Problem</b>	<b>16</b>
<b>3</b>	<b>Naive algorithm</b>	<b>18</b>
3.0	Algorithm structure . . . . .	18
3.1	Match function . . . . .	18
<b>4</b>	<b>Filter-based approach</b>	<b>20</b>
4.0	The filter function . . . . .	20
4.1	One-dimensional pattern matching in one direction . . . . .	21
4.2	Efficient computation and storage of the reduced text . . . . .	22
4.3	Baker and Bird . . . . .	23
4.4	Takaoka and Zhu . . . . .	26
4.5	Generalisations . . . . .	30
<b>5</b>	<b>Baeza-Yates and Régnier</b>	<b>32</b>
5.0	Algorithm structure . . . . .	32
5.1	Baeza-Yates and Régnier's <i>CheckMatch</i> approach . . . . .	33
5.2	Inspecting fewer pattern rows . . . . .	36
5.3	Inspecting only matching pattern rows . . . . .	39
5.4	Computation of unique row indices . . . . .	44
5.5	Generalisations . . . . .	44
<b>6</b>	<b>Polcar</b>	<b>48</b>
6.0	Introduction . . . . .	48
6.1	Derivation . . . . .	48
6.2	Algorithm structure . . . . .	54

6.3	Precomputation . . . . .	56
6.3.0	Representing sets of matrices by lists of maximal elements . . . . .	56
6.3.1	Precomputation algorithm structure . . . . .	61
6.3.2	Case analysis . . . . .	63
6.3.3	Entire precomputation algorithm . . . . .	79
6.3.4	Computation of the “failure function” . . . . .	81
6.4	Entire algorithm . . . . .	85
6.5	Remarks . . . . .	86
<b>7</b>	<b>Conclusions and future work</b>	<b>88</b>
7.0	Conclusions . . . . .	88
7.1	Future work . . . . .	88
<b>A</b>	<b>Properties of <code>div</code> and <code>mod</code></b>	<b>90</b>
<b>B</b>	<b>Properties of <code>pref</code> and <code>suff</code> (for strings)</b>	<b>92</b>
<b>C</b>	<b>Properties of <code>pref</code> and <code>suff</code> (for matrices)</b>	<b>94</b>
<b>D</b>	<b>Lists</b>	<b>96</b>

## List of Figures

0	A two-dimensional array . . . . .	12
1	Every match intersects with a row $i * m_1 - 1$ . . . . .	32
2	Submatrix of the text, inspected when a match occurs in row $i * m_1 - 1$ . . . . .	35
3	The Baeza-Yates algorithm idea, applied in three dimensions . . . . .	45
4	A pattern occurrence as a suffix of a prefix of the text . . . . .	48



## 0 Introduction

In this text we will formally derive a number of known two-dimensional pattern matching algorithms. The two-dimensional pattern matching problem consists of finding all occurrences of a given two-dimensional matrix, the so-called *pattern*, in a larger two-dimensional matrix, the *text*.

We provide formal derivations of these algorithms for a number of reasons. First of all, a formal derivation is also a correctness proof. This method also ensures that all algorithms are presented in a (more-or-less) uniform way, which is independent of implementation details and choice of programming language. And finally, this presentation also highlights the major design decisions during the algorithm's construction. Variations on these decisions may give rise to new solutions to the two-dimensional pattern matching problem.

Part of the original goal of this research was to construct a *taxonomy* of algorithms. A taxonomy is a structured classification of algorithms – see for examples [Wat95, WZ92, WZ93, WZ95, WZ96]. However, as we will see, the differences between most of the algorithms discussed here are so great, that the corresponding taxonomy would have a very coarse structure and therefore would not provide much additional value.

Section 1 introduces some definitions and notations used in the rest of the thesis. In section 2, we formally define the two-dimensional pattern matching problem. Section 3 contains a description of a very straightforward, but inefficient, solution to the problem: the naive algorithm. In section 4 the so-called filter-based approaches are discussed; most notably: Baker and Bird's algorithm ([Bak78, Bir77]) and Takaoka and Zhu's algorithm ([TZ89, TZ94]). Section 5 contains the description of Baeza-Yates and Régnier's algorithm ([BYR93]) and in section 6 we will derive Polcar's algorithm ([Pol04, MP04]). Section 7 will contain the conclusions and suggestions for future work. Finally, in the appendices we list some definitions, notations and properties that are useful in the derivations in the main text, but not an essential part of the derivations themselves.



# 1 Preliminaries

## 1.0 Two-dimensional arrays

Say we have a two-dimensional array, or matrix,  $M$ . Such a matrix can be visualised as shown in figure 0. The size of  $M$  is determined by its number of rows, denoted by  $\text{row}(M)$ , and its number of columns,  $\text{col}(M)$ .

Let  $\text{row}(M) = l_1$  and  $\text{col}(M) = l_2$ . Rows are numbered from 0 to  $l_1 - 1$ , columns from 0 to  $l_2 - 1$ . We call  $M[i][0 .. l_2)$ , or simply  $M[i]$ , the  $(i + 1)$ th row of  $M$ . Similarly,  $M[0 .. l_1][i]$  is the  $(i + 1)$ th column of  $M$ . The set of all two-dimensional matrices over  $\Sigma$  is denoted by  $\mathcal{M}_2(\Sigma)$ .

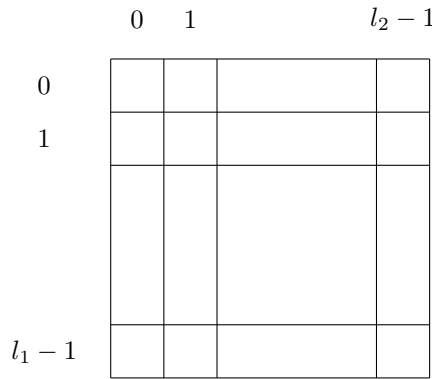


Figure 0: A two-dimensional array

We call a matrix for which the number of rows or the number of columns (or both) is equal to 0 an *empty matrix*. This is a special kind of matrix because, since it has no elements, it is completely defined by its size. We denote the empty matrix of size  $k_1 \times k_2$  by  $\mathcal{E}_{k_1, k_2}$  (where  $k_1 = 0 \vee k_2 = 0$ ). We call the set of all empty matrices  $\mathcal{ES}$ :

$$\mathcal{ES} = \langle \text{set } j_1, j_2 : 0 \leq j_1 \wedge 0 \leq j_2 \wedge (j_1 = 0 \vee j_2 = 0) : \mathcal{E}_{j_1, j_2} \rangle$$

We can also give the following alternate definition:

$$\mathcal{ES} = \langle \text{set } j : 0 \leq j : \mathcal{E}_{j, 0} \rangle \cup \langle \text{set } j : 0 \leq j : \mathcal{E}_{0, j} \rangle$$

We introduce the following notation for a set of empty matrices of a certain size, for  $0 \leq i_1, 0 \leq i_2$ :

$$\mathcal{ES}_{i_1, i_2} = \langle \text{set } j_1, j_2 : 0 \leq j_1 \leq i_1 \wedge 0 \leq j_2 \leq i_2 \wedge (j_1 = 0 \vee j_2 = 0) : \mathcal{E}_{j_1, j_2} \rangle$$

$$\mathcal{ES}_{i_1, i_2} = \langle \text{set } j : 0 \leq j \leq i_1 : \mathcal{E}_{j, 0} \rangle \cup \langle \text{set } j : 0 \leq j \leq i_2 : \mathcal{E}_{0, j} \rangle$$

Furthermore, for  $0 \leq i_1 \leq l_1 - k_1$  and  $0 \leq i_2 \leq l_2 - k_2$ , the following is a  $k_1 \times k_2$  *submatrix* of  $M$ :

$$M[i_1 .. i_1 + k_1][i_2 .. i_2 + k_2)$$

For  $0 \leq j_1 < k_1$  and  $0 \leq j_2 < k_2$ :

$$(M[i_1 .. i_1 + k_1][i_2 .. i_2 + k_2])[j_1, j_2] = M[i_1 + j_1, i_2 + j_2]$$

If  $i_1$  and  $i_2$  are both equal to 0, we call the submatrix a *prefix* of  $M$ . If  $i_1 + k_1 = l_1$  and  $i_2 + k_2 = l_2$ , we call the submatrix a *suffix* of  $M$ . More formally, a prefix of  $M$  is an element of the set  $\text{pref}(M)$ . A suffix is an element of  $\text{suff}(M)$ . We define the sets  $\text{pref}(M)$  and  $\text{suff}(M)$  similarly to the one-dimensional case:

$$\begin{aligned} \text{pref}(M) &= \langle \text{set } i_1, i_2 : 0 \leq i_1 \leq l_1 \wedge 0 \leq i_2 \leq l_2 : M[0 .. i_1][0 .. i_2] \rangle \\ \text{suff}(M) &= \langle \text{set } i_1, i_2 : 0 \leq i_1 \leq l_1 \wedge 0 \leq i_2 \leq l_2 : M[i_1 .. l_1][i_2 .. l_2] \rangle \end{aligned}$$

Note that both  $\text{pref}(M)$  and  $\text{suff}(M)$  include the following empty matrices:  $\mathcal{E}S_{l_1, l_2}$ .

Two matrices  $M$  and  $N$  are equal if they have the same size, say  $k_1 \times k_2$ , and:

$$\langle \forall h_1, h_2 : 0 \leq h_1 < k_1 \wedge 0 \leq h_2 < k_2 : M[h_1, h_2] = N[h_1, h_2] \rangle$$

## 1.1 Composed matrices

Suppose we have  $l_1 * l_2$  matrices, called  $M_{i_1, i_2}$  ( $0 \leq i_1 < l_1$  and  $0 \leq i_2 < l_2$ ), for which the following holds:

$$\langle \forall i_1, i_2 : 0 \leq i_1 < l_1 \wedge 0 \leq i_2 < l_2 : \text{row}(M_{i_1, i_2}) = \text{row}(M_{i_1, 0}) \wedge \text{col}(M_{i_1, i_2}) = \text{col}(M_{0, i_2}) \rangle$$

Then we can introduce the following *composed matrix*:

$$\left[ \begin{array}{c|c|c|c} M_{0,0} & M_{0,1} & \cdots & M_{0,l_2-1} \\ \hline M_{1,0} & M_{1,1} & \cdots & M_{1,l_2-1} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline M_{l_1-1,0} & M_{l_1-1,1} & \cdots & M_{l_1-1,l_2-1} \end{array} \right]$$

The meaning of this notation should be intuitively obvious. For a formal definition, we first introduce an auxiliary definition. For  $0 \leq i_1 \leq l_1$  and  $0 \leq i_2 \leq l_2$ :

$$\begin{aligned} r(i_1) &= \langle \Sigma j : 0 \leq j < i_1 : \text{row}(M_{j,0}) \rangle \\ c(i_2) &= \langle \Sigma j : 0 \leq j < i_2 : \text{col}(M_{0,j}) \rangle \end{aligned}$$

Let us call the composed matrix  $N$ . It is the matrix for which the following holds:

$$\begin{aligned} \text{row}(N) &= r(l_1) \\ \text{col}(N) &= c(l_2) \\ \langle \forall j_1, j_2 : 0 \leq j_1 < l_1 \wedge 0 \leq j_2 < l_2 : N[r(j_1) .. r(j_1 + 1)][c(j_2) .. c(j_2 + 1)] &= M_{j_1, j_2} \rangle \end{aligned}$$

We will discuss two special cases of the matrix composition. First we have  $[ A \mid B ]$  (which is only defined if  $\text{row}(A) = \text{row}(B)$ ). This is known as the *column concatenation* and sometimes

written as  $A \oplus B$  or  $[A \ B]$ . The other special case is  $\left[ \begin{array}{c} A \\ B \end{array} \right]$  (only defined if  $\text{col}(A) = \text{col}(B)$ ): the *row concatenation*, which is sometimes denoted in the literature by  $A \ominus B$  or  $[A; B]$ . For a more extensive description of row and column concatenation and how these two operators can be used to define two-dimensional regular expressions and two-dimensional languages, we refer to [RS97].

Using the matrix composition, we can give an alternate definition for **pref** and **suff**, which is equivalent to the definitions given in section 1.0, but expressed in terms of the matrix composition, as opposed to indices:

$$\begin{aligned} \text{pref}(M) &= \langle \text{set } A, B, C, D : M = \left[ \begin{array}{c|c} A & B \\ \hline C & D \end{array} \right] : A \rangle \\ \text{suff}(M) &= \langle \text{set } A, B, C, D : M = \left[ \begin{array}{c|c} A & B \\ \hline C & D \end{array} \right] : D \rangle \end{aligned}$$

## 1.2 One-dimensional pattern matching

In some of the two-dimensional pattern matching algorithms to be discussed, we will use a one-dimensional pattern matching algorithm (for example, on rows or columns of the text). When it is not relevant which one-dimensional pattern matching algorithm is used, we will refer to function  $PM_1$ , with the following specification (for strings  $p$  and  $t$ ):

$$PM_1(p, t) = \langle \text{set } l, r : t = lpr : |l| \rangle$$

In the same spirit we introduce the multipattern matching function  $MPM_1$ , with the following specification (for set of strings  $PS$  and string  $t$ ):

$$MPM_1(PS, t) = \langle \text{set } l, p, r : p \in PS \wedge t = lpr : (|l|, p) \rangle$$



## 2 Problem

In ‘normal’ (one-dimensional) pattern matching, the problem is to find all occurrences of a pattern  $p$  in a text  $t$ , where both  $p$  and  $t$  are strings over an alphabet  $\Sigma$ . In the two-dimensional case, instead of matching strings, we are matching two-dimensional arrays.

Our pattern  $P$  and text  $T$  are matrices over  $\Sigma$ , with  $\text{row}(P) = m_1$ ,  $\text{col}(P) = m_2$ ,  $\text{row}(T) = n_1$  and  $\text{col}(T) = n_2$ . In practical applications, the text is often a picture. However, we continue using the term ‘text’ in this overview to emphasise the similarities to one-dimensional pattern matching.

The problem is to find all exact matches of pattern  $P$  in text  $T$ . More formally, our postcondition is

$$R : O = \langle \text{set } i_1, i_2 : \quad 0 \leq i_1 \leq n_1 - m_1 \wedge 0 \leq i_2 \leq n_2 - m_2 \wedge \\ T[i_1 .. i_1 + m_1][i_2 .. i_2 + m_2] = P : (i_1, i_2) \rangle$$

Note that we identify occurrences of the pattern by their ‘upper-left corner’. Since the problem we are focussing on is *exact* two-dimensional pattern matching, the size of each occurrence is the same and equal to the size of the pattern. Therefore, any point can be used to represent an occurrence. We have chosen the ‘upper left corner’ because it is convenient for most of the algorithms we will discuss, but we could have just as easily reported the ‘lower-right corner’, or any other point, instead.



### 3 Naive algorithm

#### 3.0 Algorithm structure

The simplest way of establishing  $R$  is by checking, for all positions  $(i_1, i_2)$  in the text, whether there is a match starting at that position.

We introduce a set of index pairs  $D : \mathbb{N} \times \mathbb{N}$ . We will maintain the following invariant:

$$P0 : O = \langle \text{set } i_1, i_2 : (i_1, i_2) \in D \wedge T[i_1 .. i_1 + m_1][i_2 .. i_2 + m_2] = P : (i_1, i_2) \rangle$$

Invariant  $P0$  is trivially established by the assignment  $O, D := \emptyset, \emptyset$ . We have established  $R$  when:

$$D = [0, n_1 - m_1] \times [0, n_2 - m_2]$$

Ad  $P0(D := D \cup \{(j_1, j_2)\})$ :

$$\begin{aligned} & \langle \text{set } i_1, i_2 : (i_1, i_2) \in D \cup \{(j_1, j_2)\} \wedge T[i_1 .. i_1 + m_1][i_2 .. i_2 + m_2] = P : (i_1, i_2) \rangle \\ = & \{ \text{split off } (i_1, i_2) = (j_1, j_2), P0 \} \\ & \begin{cases} O \cup \{(j_1, j_2)\} & \text{if } T[j_1 .. j_1 + m_1][j_2 .. j_2 + m_2] = P \\ O & \text{if } T[j_1 .. j_1 + m_1][j_2 .. j_2 + m_2] \neq P \end{cases} \\ = & \{ \bullet \text{match}_{T,P}(i_1, i_2) \equiv T[i_1 .. i_1 + m_1][i_2 .. i_2 + m_2] = P \} \\ & \begin{cases} O \cup \{(j_1, j_2)\} & \text{if } \text{match}_{T,P}(j_1, j_2) \\ O & \text{if } \neg \text{match}_{T,P}(j_1, j_2) \end{cases} \end{aligned}$$

Now we can give the so-called “naive algorithm”:

```

O, D := ∅, ∅;
for j1, j2 : 0 ≤ j1 ≤ n1 - m1 ∧ 0 ≤ j2 ≤ n2 - m2 → { inv. P0 }
  if matchT,P(j1, j2) → O := O ∪ {(j1, j2)}
  || ¬matchT,P(j1, j2) → skip
  fi;
  D := D ∪ {(j1, j2)}
rof

```

Note that the assignments to set  $D$  can be removed from the algorithm without harming its correctness;  $D$  is never inspected.

#### 3.1 Match function

Recall our specification of  $\text{match}_{T,P}$ :

$$\text{match}_{T,P}(i_1, i_2) \equiv T[i_1 .. i_1 + m_1][i_2 .. i_2 + m_2] = P$$

Again, we introduce a set of index pairs,  $C : \mathbb{N} \times \mathbb{N}$ . We will maintain the following invariant:

$$Q0 : \text{res} \equiv \langle \forall j_1, j_2 : (j_1, j_2) \in C : T[i_1 + j_1, i_2 + j_2] = P[j_1, j_2] \rangle$$

Invariant  $Q0$  is initially trivially established by  $res, C := true, \emptyset$ . When the following holds, we have established  $res = match_{T,P}(i_1, i_2)$ :

$$C = [0, m_1) \times [0, m_2)$$

Ad  $Q0(C := C \cup \{(k_1, k_2)\})$ :

$$\begin{aligned} & \langle \forall j_1, j_2 : (j_1, j_2) \in C \cup \{(k_1, k_2)\} : T[i_1 + j_1, i_2 + j_2] = P[j_1, j_2] \rangle \\ \equiv & \quad \{ \text{split off } (j_1, j_2) = (k_1, k_2), Q0 \} \\ & res \wedge T[i_1 + k_1, i_2 + k_2] = P[k_1, k_2] \end{aligned}$$

The complete implementation becomes the following:

```

func matchT,P(i1, i2 :integer) :boolean
{ pre: 0 ≤ i1 ≤ n1 − m1 ∧ 0 ≤ i2 ≤ n2 − m2 }
{ result: T[i1 .. i1 + m1][i2 .. i2 + m2] = P }
||
  res, C := true, ∅;
  for k1, k2 : 0 ≤ k1 < m1 ∧ 0 ≤ k2 < m2 →
    res := res ∧ T[i1 + k1, i2 + k2] = P[k1, k2];
    C := C ∪ {(k1, k2)}
  rof;
  return(res)
||

```

Note that we can improve on this algorithm: we can stop the computation as soon as  $res$  becomes false. We can also omit set  $C$ , since it is never inspected.

## 4 Filter-based approach

### 4.0 The filter function

The idea is to reduce the two-dimensional pattern matching problem to “normal” (one-dimensional) pattern matching, by means of a “filter function”. More specifically, we want to reduce the problem to matching occurrences of a pattern string  $p$  in the columns of a matrix  $t$ , where a detected match corresponds to a (possible) occurrence of our pattern  $P$  in text  $T$ . In order to do this, we reduce each row of  $P$  to a single value, using our filter function. Then  $p$  is simply the concatenation of these values.

Note that the two-dimensional pattern matching problem is symmetrical in both dimensions. We could just as well reduce *columns* of the pattern to a single value and then search for their occurrence in the *rows* of the text. (This incidentally corresponds to applying our method to the transposed pattern and text.)

We introduce column vector  $p$  of length  $m_1$  over  $X$  (we will later implicitly interpret  $p$  as a string), matrix  $t$  over  $X$ , of size  $n_1 \times n_2 - m_2$ , and function  $f_P : \Sigma^{m_2} \rightarrow X$  (where  $X$  is some still unspecified set), with the following relationship.

$$\begin{aligned} p[i] &= f_P(P[i]) & (0 \leq i < m_1) \\ t[i_1, i_2] &= f_P(T[i_1][i_2 .. i_2 + m_2]) & (0 \leq i_1 < n_1, 0 \leq i_2 \leq n_2 - m_2) \end{aligned}$$

We write the subscript  $P$  in  $f_P$ , because in some (but not all) of the algorithms that we will describe, the value of  $f_P$  will depend on  $P$ .

Now we have:

$$\begin{aligned} &T[i_1][i_2 .. i_2 + m_1] \neq P[i] \\ \Leftarrow &\{ f_P \text{ is a function} \} \\ &f_P(T[i_1][i_2 .. i_2 + m_2]) \neq f_P(P[i]) \\ \equiv &\{ \text{spec. } p, t \} \\ &t[i_1, i_2] \neq p[i] \end{aligned}$$

Therefore, we can conclude:

$$T[i_1 .. i_1 + m_1][i_2 .. i_2 + m_2] \neq P \Leftarrow t[i_1 .. i_1 + m_1][i_2] \neq p \tag{0}$$

We can now use one-dimensional pattern matching techniques for matching  $p$  against the columns of  $t$ . As we have seen, there can only be a match of  $P$  in  $T$  on those positions, where  $p$  and  $t$  match. On the other hand, in the general case, when we detect a match of  $p$  in  $t$ , we still need to check whether there is an actual match of  $P$  on that position in  $T$ . To do this, we can use the function “ $\text{match}_{T,P}$ ”, as described in section 3.1.

We introduce the following invariant:

$$P0 : O = \langle \text{set } i_1, i_2 : (i_1, i_2) \in D \wedge T[i_1 .. i_1 + m_1][i_2 .. i_2 + m_2] = P : (i_1, i_2) \rangle$$

This is the same invariant we used in the Naive Algorithm (see section 3.0). Termination: when  $D = [0, n_1 - m_1] \times [0, n_2 - m_2]$ . However, our update of  $D$  will be slightly different; instead of adding one pair of indices to  $D$  at a time, we will add  $[0, n_1 - m_1] \times \{j\}$ .

$$\begin{aligned}
& \langle \mathbf{set} \ i_1, i_2 : (i_1, i_2) \in D \cup ([0, n_1 - m_1] \times \{j\}) \wedge \\
& \quad T[i_1 .. i_1 + m_1][i_2 .. i_2 + m_2] = P : (i_1, i_2) \rangle \\
= & \quad \{ \text{split off: } [0, n_1 - m_1] \times \{j\}, \text{ use: } P0 \} \\
& O \cup \langle \mathbf{set} \ i_1 : 0 \leq i_1 \leq n_1 - m_1 \wedge T[i_1 .. i_1 + m_1][j .. j + m_2] = P : (i_1, j) \rangle \\
= & \quad \{ (0) \} \\
& O \cup \langle \mathbf{set} \ i_1 : 0 \leq i_1 \leq n_1 - m_1 \wedge t[i_1 .. i_1 + m_1][j] = p \wedge \\
& \quad T[i_1 .. i_1 + m_1][j .. j + m_2] = P : (i_1, j) \rangle \\
= & \quad \{ \text{spec. } PM_1, \text{ spec. } \text{match}_{T,P} \} \\
& O \cup \langle \mathbf{set} \ i_1 : i_1 \in PM_1(p, t[0 .. n_1][j]) \wedge \text{match}_{T,P}(i_1, j) : (i_1, j) \rangle
\end{aligned}$$

Our algorithm now becomes:

```

“construct  $p$ ”;
“construct  $t$ ”;
 $O, D := \emptyset, \emptyset$ ;
for  $j : 0 \leq j \leq n_2 - m_2 \rightarrow \{ \text{inv. } P0 \}$ 
  for  $i : i \in PM_1(p, t[0 .. n_1][j]) \rightarrow$ 
    if  $\text{match}_{T,P}(i, j) \rightarrow O := O \cup \{(i, j)\}$ 
    ||  $\neg \text{match}_{T,P}(i, j) \rightarrow \mathbf{skip}$ 
  fi
rof;
 $D := D \cup ([0, n_1 - m_1] \times \{j\})$ 
rof

```

Again, set  $D$  can be omitted without loss of correctness. Note that, depending on the choice of a filter function,  $i \in PM_1(p, t[0 .. n_1][j])$  may imply that certain elements of  $P$  and  $T$  are equal, rendering some or all of the comparisons in  $\text{match}_{T,P}(i, j)$  unnecessary.

Now we only need to decide how to construct  $p$  and  $t$ , which depends on our choice for  $f_P$ . Depending on this choice of  $f_P$ , we can get several different algorithms. The most simple filter function is, for  $x \in \Sigma^{m_2}$  and any arbitrary value  $a$ :

$$f_P(x) = a$$

This essentially results in the naive algorithm, as described in section 3. In the remainder of this section we will investigate several other filter functions, which will give rise to different algorithms.

#### 4.1 One-dimensional pattern matching in one direction

Another very simple choice for  $f_P$  is the following, for all  $x \in \Sigma^{m_2}$  and some  $i, 0 \leq i < m_2$ :

$$f_P(x) = x[i]$$

Here, the codomain of function  $f_P$  is  $\Sigma$ . Obviously this is only a valid choice if we assume  $0 < m_2$ ; that is, our pattern  $P$  is not the empty matrix  $\mathcal{E}$ .

What this boils down to is searching for one column of the pattern in the text's columns using a one-dimensional pattern matching technique. If such a column is found, we use a brute force check to match the other columns.

Note that this filter function's values do not depend on the first  $i$  and the last  $m_2 - 1 - i$  columns of both pattern  $P$  and text  $T$ .

## 4.2 Efficient computation and storage of the reduced text

As we have seen, the columns of matrix  $t$  are inspected one by one in our filter-based algorithms. The order in which they are inspected has been left unspecified so far. In the remaining filter-based algorithms, it is possible to efficiently compute the values of column  $i_2 + 1$  from those of column  $i_2$ . So we will decide to inspect the columns of  $t$  in increasing order. In this case, it is not even necessary to precompute and store the entire matrix  $t$ ; we can simply precompute the first column and then compute the next column on the fly.

Recall our relationship between  $t$  and  $f_P$ :

$$t[i_1, i_2] = f_P(T[i_1][i_2 .. i_2 + m_2]) \quad (0 \leq i_1 < n_1, 0 \leq i_2 \leq n_2 - m_2)$$

We can make this improvement whenever the value of  $f_P(xb)$  (for  $x \in \Sigma^{m_2-1}$  and  $b \in \Sigma$ ) can be expressed in terms of  $f_P(ax)$ ,  $a$  and  $b$  (for any  $a: a \in \Sigma$ ). In other words, when we can find a function  $g_P : X \times \Sigma \times \Sigma \rightarrow X$ , with the following property:

$$f_P(xb) = g_P(f_P(ax), a, b)$$

To replace  $t$ , we introduce  $s[0 .. n_1]$ , which has the following relationship with  $t$  (for  $0 \leq i < n_1$ ):

$$s[i] = t[i, j] \tag{1}$$

That is, we have the following invariant:

$$s[i] = f_P(T[i][j .. j + m_2])$$

Then our algorithm becomes:

```

“construct  $p$ ”;
“construct initial value of  $s$ ”;
 $O := \emptyset$ ;
 $j := 0$ ;
do  $j \neq n_2 - m_2 \rightarrow$ 
  for  $i : i \in PM_1(p, s) \rightarrow$ 
    if  $\text{match}_{T,P}(i, j) \rightarrow O := O \cup \{(i, j)\}$ 
    ||  $\neg \text{match}_{T,P}(i, j) \rightarrow \text{skip}$ 
    fi
  rof;
  for  $i : 0 \leq i < n_1 \rightarrow$ 
     $s[i] := g_P(s[i], T[i_1, i_2], T[i_1, i_2 + m_2])$ 
  rof;
   $j := j + 1$ 
od;
for  $i : i \in PM_1(p, s) \rightarrow$ 

```

```

if matchT,P(i, j) → O := O ∪ {(i, j)}
  || ¬matchT,P(i, j) → skip
fi
rof

```

The initial value of  $s[i]$  is  $f_P(T[i][0 .. m_2])$ , for  $0 \leq i < n_1$ .

To avoid inspection of the elements of the (nonexisting) column  $n_2$  of the text, here we have “peeled off” the last layer of the main repetition. This is only possible if  $m_2 < n_2$  (“the text is larger than the pattern”).

This approach, using function  $g_P$ , allows us to exploit the possibility of efficiently computing the next required values of  $f_P$ . It also provides a space improvement: we replaced matrix  $t$  of size  $n_1 \times n_2 - m_2$  by vector  $s$  of size  $n_1$ .

This improvement was included in the original description ([TZ89, TZ94]) of the Takaoka-Zhu algorithm (which is the filter-based algorithm we will discuss in section 4.4). In a recently published paper, [MŽ05], Bořivoj Melichar and Jan Žďárek propose the same space improvement for the Baker and Bird algorithm (discussed in section 4.3). We had already generalised the improvement to be applied to Baker and Bird’s algorithm, independently of Melichar and Žďárek.

### 4.3 Baker and Bird

The idea is to construct the optimal Aho-Corasick automaton (introduced in [AC75]), where the pattern set contains the rows of  $P$ , and to use this automaton’s states for the result values of function  $f_P$ . More formally, our pattern set is the set  $PR$  (“pattern rows”), defined by:

$$PR = \langle \text{set } i : 0 \leq i < m_1 : P[i] \rangle$$

The optimal Aho-Corasick automaton based on pattern row set  $PR$  is a deterministic finite automaton (DFA):  $(Q, \Sigma, \delta, q_0, F)$ . We will not go into the details of constructing such an automaton; instead we refer to [WZ92] (pages 11 – 16), or to [WZ93]<sup>0</sup>. Our filter function becomes:

$$f_P(x) = \delta^*(q_0, x)$$

In this case, the codomain of  $f_P$  is state set  $Q$ . For the computation of  $\delta^*(q_0, x)$  we can use the following program fragment:

```

j, r := 0, q0;
do j ≠ |x| →
  r := δ(r, x[j]);
  j := j + 1
od
{ r = δ*(q0, x) }

```

We introduce the following abbreviation for this program fragment:

---

<sup>0</sup>In these articles, a Moore machine is presented. However, we do not need the properties of a Moore machine here; a DFA suffices for our purposes. In fact, we do not even need the set of final states  $F$ .

$$r := \delta^*(q_0, x)$$

$$\{ r = \delta^*(q_0, x) \}$$

**Vector**  $p$

$$p[i]$$

$$= \{ \text{spec. } p, \text{ def. } f_P \}$$

$$\delta^*(q_0, P[i])$$

The program for “construct  $p$ ” simply becomes the following:

**for**  $i : 0 \leq i < m_1 \rightarrow$   
 $p[i] := \delta^*(q_0, P[i])$   
**rof**

**Vector**  $s$

The initial computation of  $s$  is very similar to that of  $p$ , since our specification gives:

$$s[i] = \delta^*(q_0, T[i][0 .. m_2])$$

Now, for our update of  $s$ , we will try to find a function  $g_P$ , satisfying the following property, for  $x \in \Sigma^{m_2-1}$  and  $a, b \in \Sigma$  (see section 4.2):

$$f_P(xb) = g_P(f_P(ax), a, b)$$

$$f_P(xb)$$

$$= \{ \text{def. } f_P \}$$

$$\delta^*(q_0, xb)$$

$$= \{ \bullet m_2 \leq |xb|, \delta^*(q, axb) = \delta^*(q, xb) \}$$

$$\delta^*(q_0, axb)$$

$$= \{ \delta^* \}$$

$$\delta(\delta^*(q_0, ax), b)$$

$$= \{ \text{def. } f_P \}$$

$$\delta(f_P(ax), b)$$

So we define:

$$g_P(q, a, b) = \delta(q, b)$$

Note that the value of  $g$  is independent of parameter  $a$  in this case.

In order to prove that the preceding derivation is correct, we still need to prove:

$$\delta^*(q, ax) = \delta^*(q, x), \text{ for } m_2 \leq |x|$$

Here we will use that our state set  $Q$  is in fact  $\mathcal{P}(\text{pref}(PR))$  and the following definition for state  $q$ :

$$q = \text{suff}(w_q) \cap \text{pref}(PR)$$

In this definition,  $w_q$  is the string that consists of the labels on the shortest path from  $q_0$  to  $q$ .

$$\begin{aligned}
& \delta^*(q, ax) \\
= & \{ \delta^* \} \\
& \text{suff}(w_q ax) \cap \text{pref}(PR) \\
= & \{ \text{theorem B.1 (on page 92)} \} \\
& (\text{suff}(w_q)ax \cup \text{suff}(x)) \cap \text{pref}(PR) \\
= & \{ m_2 < |ax|, \text{ therefore: } \text{suff}(w_q)ax \cap \text{pref}(PR) = \emptyset; \text{ distributivity} \} \\
& \text{suff}(x) \cap \text{pref}(PR) \\
= & \{ x \in \text{suff}(x) \} \\
& (\{x\} \cup \text{suff}(x)) \cap \text{pref}(PR) \\
= & \{ m_2 \leq |x|, \text{ therefore: } \text{suff}(w_q)x \cap \text{pref}(PR) = \{x\} \cap \text{pref}(PR); \text{ distributivity} \} \\
& (\text{suff}(w_q)x \cup \text{suff}(x)) \cap \text{pref}(PR) \\
= & \{ \text{theorem B.0 (on page 92)} \} \\
& \text{suff}(w_q x) \cap \text{pref}(PR) \\
= & \{ \delta^* \} \\
& \delta^*(q, x)
\end{aligned}$$

### Remarks

We know, from (0) and (1):

$$T[i .. i + m_1][j .. j + m_2] \neq P \Leftrightarrow s[i .. i + m_1] \neq p$$

However, because of our particular choice of  $f_P$  (the states in an Aho-Corasick automaton), in this case we have:

$$\begin{aligned}
& s[i .. i + m_1] \neq p \\
\equiv & \{ \text{string equality} \} \\
& \langle \forall k : 0 \leq k < m_1 : s[i + k] \neq p[k] \rangle \\
\equiv & \{ \text{specification } s, p \} \\
& \langle \forall k : 0 \leq k < m_1 : f_P(T[i + k][j .. j + m_2]) \neq f_P(P[k]) \rangle \\
\equiv & \{ \text{definition } f_P \} \\
& \langle \forall k : 0 \leq k < m_1 : \delta^*(q_0, T[i + k][j .. j + m_2]) \neq \delta^*(q_0, P[k]) \rangle \\
\equiv & \{ \text{property of Aho-Corasick automata} \} \\
& \langle \forall k : 0 \leq k < m_1 : T[i + k][j .. j + m_2] \neq P[k] \rangle \\
\equiv & \{ \text{equality of matrices} \} \\
& T[i .. i + m_1][j .. j + m_2] \neq P
\end{aligned}$$

Therefore, the call to  $\text{match}_{T,P}$  in our main algorithm is not necessary; when we detect a match of  $p$  in  $s$  we can immediately conclude that there is a match of  $P$  in  $T$ .

We will give the complete algorithm one more time.

```

for  $i : 0 \leq i < m_1 \rightarrow$ 
     $p[i] := \delta^*(q_0, P[i])$ 
rof;
for  $i : 0 \leq i < n_1 \rightarrow$ 
     $s[i] := \delta^*(q_0, T[i][0 .. m_2])$ 
rof;
 $O := \emptyset;$ 
 $j := 0;$ 
do  $j \neq n_2 - m_2 \rightarrow$ 
    for  $i : i \in PM_1(p, s) \rightarrow$ 
         $O := O \cup \{(i, j)\}$ 
    rof;
    for  $i : 0 \leq i < n_1 \rightarrow$ 
         $s[i] := \delta(s[i], T[i, j + m_2])$ 
    rof;
     $j := j + 1$ 
od;
for  $i : i \in PM_1(p, s) \rightarrow$ 
     $O := O \cup \{(i, j)\}$ 
rof

```

This algorithm was first discovered, independently, by Bird ([Bir77]) and Baker ([Bak78]). Another presentation is given in [CR02]. In the original presentations, the method for one-dimensional pattern matching was explicitly selected: the Knuth-Morris-Pratt algorithm ([KMP77]), which is the single-string version of the Aho-Corasick algorithm ([AC75]). I have not explicitly chosen an algorithm here, instead using  $PM_1$ .

#### 4.4 Takaoka and Zhu

Here we choose for  $f_P$  the hash function, as used in Karp and Rabin's one-dimensional pattern matching algorithm ([KR87]). For  $x \in \Sigma^{m_2}$  we define:

$$f_P(x) = \langle \Sigma^j : 0 \leq j < m_2 : \text{ord}(x[j]) * |\Sigma|^{m_2-1-j} \rangle \mathbf{mod} \ q \quad (2)$$

Here,  $q$  is some large prime number. The function  $\text{ord}$  is a bijective function  $\Sigma \rightarrow [0 .. |\Sigma|)$ . That is, a function for which the following holds for all  $a, b \in \Sigma$ :

$$\text{ord}(a) = \text{ord}(b) \equiv a = b$$

An interesting special case of this choice of  $f_P$  is the one where  $\Sigma$  is equal to  $\{0, 1\}$  (and therefore,  $|\Sigma| = 2$ ) and  $\text{ord}$  is simply the identity function. In this case, our text and alphabet are matrices over bits. The filter function  $f_P$  will then look like this:

$$f_P(x) = \langle \Sigma^j : 0 \leq j < m_2 : x[j] * 2^{m_2-1-j} \rangle \mathbf{mod} \ q$$

Because we are multiplying by (powers of) 2, this may lead to efficient implementations of the algorithm (possibly using bit parallelism).

In most practical applications, converting any given matrix into such a form is rather easy. Each element in the original matrix is represented by a subrow (or subcolumn) in our new bitmatrix. Using this technique, the size of our matrix increases by the number of bits necessary to represent all values of the original alphabet. In practice we do not need to duplicate the entire matrix; we can use its internal bit representation.

In [KR87], Karp and Rabin examine the same special case of  $\Sigma = \{0, 1\}$  for their one-dimensional pattern matching algorithm as well. For the remainder of this section, we will consider the more general definition of  $f_P$ , as given in (2).

### Vector $p$

We will need to compute  $p[i]$ , for all  $i$  ( $0 \leq i < m_1$ ). The postcondition for “ $p[i] := f_P(P[i])$ ” is the following:

$$p[i] = \langle \Sigma^j : 0 \leq j < m_2 : \text{ord}(P[i, j]) * |\Sigma|^{m_2-1-j} \rangle \mathbf{mod} q$$

We introduce the following invariants:

$$Q0 : 0 \leq k \leq m_2$$

$$Q1 : p[i] = \langle \Sigma^j : 0 \leq j < k : \text{ord}(P[i, j]) * |\Sigma|^{k-1-j} \rangle \mathbf{mod} q$$

Ad  $Q1(k := k + 1)$ :

$$\begin{aligned} & \langle \Sigma^j : 0 \leq j < k + 1 : \text{ord}(P[i, j]) * |\Sigma|^{k+1-1-j} \rangle \mathbf{mod} q \\ = & \{ \text{split off: } j = k \} \\ & (\langle \Sigma^j : 0 \leq j < k : \text{ord}(P[i, j]) * |\Sigma|^{k+1-1-j} \rangle + \text{ord}(P[i, k]) * |\Sigma|^0) \mathbf{mod} q \\ = & \{ \text{math} \} \\ & (\langle \Sigma^j : 0 \leq j < k : \text{ord}(P[i, j]) * |\Sigma|^{k-1-j} \rangle * |\Sigma| + \text{ord}(P[i, k])) \mathbf{mod} q \\ = & \{ (a * b + c) \mathbf{mod} q = ((a \mathbf{mod} q) * b + c) \mathbf{mod} q \text{ (theorem A.2 on page 90), } Q1 \} \\ & (p[i] * |\Sigma| + \text{ord}(P[i, k])) \mathbf{mod} q \end{aligned}$$

Our algorithm “construct  $p$ ” becomes:

```

for  $i : 0 \leq i < m_1 \rightarrow$ 
   $k, p[i] := 0, 0;$ 
  do  $k \neq m_2 \rightarrow$ 
     $p[i] := (p[i] * |\Sigma| + \text{ord}(P[i, k])) \mathbf{mod} q;$ 
     $k := k + 1$ 
  od
rof

```

Because our update of  $p[i]$  is always computed modulo  $q$ ,  $p[i] < q$  is an invariant of the computation. Therefore we know that all intermediate results in the computation will be at most  $q * |\Sigma|$ .

### Vector $s$

The initial value of  $s$  can be computed using an algorithm, similar to “construct  $p$ ”. For the update of  $s$ , we have the following derivation (for  $x \in \Sigma^{m_2-1}$  and  $a, b \in \Sigma$ ):

$$\begin{aligned}
& f_P(xb) \\
= & \{ \text{def. } f_P \} \\
& \langle \Sigma^j : 0 \leq j < m_2 : \text{ord}((xb)[j]) * |\Sigma|^{m_2-1-j} \bmod q \\
= & \{ \text{split off: } j = m_2 - 1, \text{ use: } |xb| = m_2 \} \\
& (\langle \Sigma^j : 0 \leq j < m_2 - 1 : \text{ord}(x[j]) * |\Sigma|^{m_2-1-j} + \text{ord}(b) \rangle \bmod q \\
= & \{ \text{dummy transformation: } j := j - 1 \} \\
& (\langle \Sigma^j : 1 \leq j < m_2 : \text{ord}(x[j-1]) * |\Sigma|^{m_2-j} + \text{ord}(b) \rangle \bmod q \\
= & \{ \text{domain expansion: } j = 0, \text{ use: } |ax| = m_2 \} \\
& (\langle \Sigma^j : 0 \leq j < m_2 : \text{ord}((ax)[j]) * |\Sigma|^{m_2-j} - \text{ord}(a) * |\Sigma|^{m_2} + \text{ord}(b) \rangle \bmod q \\
= & \{ \text{math} \} \\
& (\langle \Sigma^j : 0 \leq j < m_2 : \text{ord}((ax)[j]) * |\Sigma|^{m_2-1-j} * |\Sigma| - \text{ord}(a) * |\Sigma|^{m_2} + \text{ord}(b) \rangle \bmod q \\
= & \{ \text{theorem A.2 (on page 90)} \} \\
& ((\langle \Sigma^j : 0 \leq j < m_2 : \text{ord}((ax)[j]) * |\Sigma|^{m_2-1-j} \rangle \bmod q) * |\Sigma| - \text{ord}(a) * |\Sigma|^{m_2} + \text{ord}(b)) \bmod q \\
= & \{ \text{def. } f_P \} \\
& (f_P(ax) * |\Sigma| - \text{ord}(a) * |\Sigma|^{m_2} + \text{ord}(b)) \bmod q
\end{aligned}$$

So we get the following definition for  $g_P$ :

$$g_P(q, a, b) = (q * |\Sigma| - \text{ord}(a) * |\Sigma|^{m_2} + \text{ord}(b)) \bmod q$$

This corresponds to the *rehash* function from the original article by Takaoka and Zhu ([TZ89, TZ94]). Note that  $|\Sigma|^{m_2}$  is a constant and can therefore be precomputed.

### Remarks

We present the complete algorithm.

```

for  $i : 0 \leq i < m_1 \rightarrow$ 
   $k, p[i] := 0, 0;$ 
  do  $k \neq m_2 \rightarrow$ 
     $p[i] := (p[i] * |\Sigma| + \text{ord}(P[i, k])) \bmod q;$ 
     $k := k + 1$ 
  od
rof;
for  $i : 0 \leq i < n_1 \rightarrow$ 
   $k, s[i] := 0, 0;$ 
  do  $k \neq m_2 \rightarrow$ 
     $s[i] := (s[i] * |\Sigma| + \text{ord}(T[i, k])) \bmod q;$ 
     $k := k + 1$ 
  od
rof;
 $O := \emptyset;$ 

```

```

j := 0;
do j ≠ n2 - m2 →
  for i : i ∈ PM1(p, s) →
    if matchT,P(i, j) → O := O ∪ {(i, j)}
    || ¬matchT,P(i, j) → skip
  fi
  rof;
  for i : 0 ≤ i < n1 →
    s[i] := (s[i] * |Σ| - ord(T[i, j]) * |Σ|m2 + ord(T[i, j + m2])) mod q
  rof;
  j := j + 1
od;
for i : i ∈ PM1(p, s) →
  if matchT,P(i, j) → O := O ∪ {(i, j)}
  || ¬matchT,P(i, j) → skip
fi
rof

```

This algorithm is presented in [TZ89, TZ94]. In the original article, the authors wrote the following update of  $s[i]$ , which differs slightly from the one presented above:<sup>1</sup>

$$s[i] := \left( \left( s[i] + |\Sigma| * q - \text{ord}(T[i, j]) * (|\Sigma|^{m_2-1} \bmod q) \right) * |\Sigma| + \text{ord}(T[i, j + m_2]) \right) \bmod q$$

However, we can prove that the two expressions are equal, using theorem A.2.

$$\begin{aligned}
& ((s[i] + |\Sigma| * q - \text{ord}(T[i, j]) * (|\Sigma|^{m_2-1} \bmod q)) * |\Sigma| + \text{ord}(T[i, j + m_2])) \bmod q \\
= & \{ * \text{ over } + \} \\
& (s[i] * |\Sigma| + |\Sigma|^2 * q - \text{ord}(T[i, j]) * (|\Sigma|^{m_2-1} \bmod q) * |\Sigma| + \text{ord}(T[i, j + m_2])) \bmod q \\
= & \{ (|\Sigma|^2 * q) \bmod q = 0 \} \\
& (s[i] * |\Sigma| - \text{ord}(T[i, j]) * (|\Sigma|^{m_2-1} \bmod q) * |\Sigma| + \text{ord}(T[i, j + m_2])) \bmod q \\
= & \{ \text{theorem A.2, math} \} \\
& (s[i] * |\Sigma| - \text{ord}(T[i, j]) * |\Sigma|^{m_2} + \text{ord}(T[i, j + m_2])) \bmod q
\end{aligned}$$

The differences between the two expressions are the following:

- Where we suggested that the value of  $|\Sigma|^{m_2}$  can be precomputed, Takaoka and Zhu chose to precompute  $|\Sigma|^{m_2-1} \bmod q$  and then multiply it by  $|\Sigma|$ . The advantage of storing the constant modulo  $q$  is that its value is at most  $q$ . In an implementation, this can be useful for preventing an overflow. By replacing the constant  $|\Sigma|^{m_2}$  with  $|\Sigma|^{m_2} \bmod q$  in the algorithm text we presented, we can achieve the same advantage.
- Takaoka and Zhu included an extra term  $|\Sigma| * q$  (or, more accurately,  $|\Sigma|^2 * q$ ). It is possible that this term was included to ensure that the computation only has nonnegative intermediate results. (The only negative term in the expression is  $-\text{ord}(T[i, j]) * (|\Sigma|^{m_2-1} \bmod q)$ . We know:  $\text{ord}(T[i, j]) < |\Sigma|$  and  $|\Sigma|^{m_2-1} \bmod q < q$ , so  $\text{ord}(T[i, j]) * (|\Sigma|^{m_2-1} \bmod q) < |\Sigma| * q$ .)

Another difference with the original article is that the authors, like Baker and Bird, explicitly chose the Knuth-Morris-Pratt algorithm ([KMP77]) for one-dimensional pattern matching, where we referred to  $PM_1$ .

<sup>1</sup>In an attempt to improve readability, we have used brackets of different sizes in the expression.

## 4.5 Generalisations

We can use these filter-based techniques easily for two-dimensional multipattern matching, where we have  $k$  pattern matrices:  $P_0, \dots, P_{k-1}$ . We can use our filter function  $f_P$  to reduce each of these patterns to a column vector  $p_i$ . Then we use one-dimensional multipattern matching to match these column vectors with the reduced text.

To apply  $f_P$  to all rows of all patterns, the pattern matrices need to have the same width (number of columns). However, their lengths (that is: number of rows) may vary; one-dimensional multipattern matching strategies do not depend on the patterns all being of the same length.

Pattern matching in more than two dimensions is also possible. For matching in  $n + 1$  dimensions ( $1 \leq n$ ), we reduce our pattern to an  $n$ -dimensional matrix using our filter function  $f_P$  and then apply an  $n$ -dimensional pattern matching technique for matching the reduced pattern and the reduced text.



## 5 Baeza-Yates and Régnier

### 5.0 Algorithm structure

In section 4 we described algorithms where we reduce an entire row of the pattern to a single value, in order to be able to use one-dimensional pattern matching techniques on the resulting column vector of values. Here we will present a different kind of filter algorithm. It is based on the knowledge that all matches of pattern  $P$  in text  $T$  intersect with exactly one row of  $T$  with a number of the form  $i * m_1 - 1$ , since our pattern has exactly  $m_1$  rows. See figure 1.

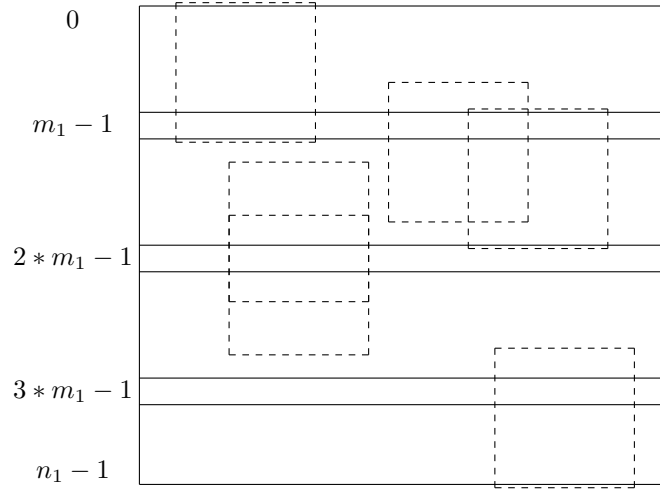


Figure 1: Every match intersects with a row  $i * m_1 - 1$

In this algorithm description, we will use the set  $PR$  (as in Baker and Bird, in section 4.3), which is the set of rows of the pattern  $P$ .

$$PR = \langle \text{set } i : 0 \leq i < m_1 : P[i] \rangle$$

Recall our postcondition  $R$ :

$$O = \langle \text{set } i_1, i_2 : 0 \leq i_1 \leq n_1 - m_1 \wedge 0 \leq i_2 \leq n_2 - m_2 \wedge T[i_1 .. i_1 + m_1][i_2 .. i_2 + m_2] = P : (i_1, i_2) \rangle$$

We start our derivation with the predicate “a match of  $P$  in  $T$  exists”:

$$\begin{aligned} & \langle \exists i_1, i_2 : 0 \leq i_1 \leq n_1 - m_1 \wedge 0 \leq i_2 \leq n_2 - m_2 : \\ & \quad T[i_1 .. i_1 + m_1][i_2 .. i_2 + m_2] = P \rangle \\ \Rightarrow & \quad \{ i_1 \leq (i_1 \text{ div } m_1 + 1) * m_1 - 1 < i_1 + m_1 \text{ (see theorem A.3 on page 90); } \\ & \quad PR \text{ contains all rows of } P \\ & \quad \} \\ & \langle \exists i_1, i_2 : 0 \leq i_1 \leq n_1 - m_1 \wedge 0 \leq i_2 \leq n_2 - m_2 : \\ & \quad T[(i_1 \text{ div } m_1 + 1) * m_1 - 1][i_2 .. i_2 + m_2] \in PR \rangle \\ \equiv & \quad \{ \text{specification } MPM_1 \} \end{aligned}$$

$$\begin{aligned}
& \langle \exists i_1 : 0 \leq i_1 \leq n_1 - m_1 : MPM_1(PR, T[(i_1 \mathbf{div} m_1 + 1) * m_1 - 1]) \neq \emptyset \rangle \\
\equiv & \quad \{ \text{property } \exists; \text{ dummy transformation: } i = i_1 \mathbf{div} m_1 + 1 \} \\
& \langle \exists i : 1 \leq i \leq n_1 \mathbf{div} m_1 : MPM_1(PR, T[i * m_1 - 1]) \neq \emptyset \rangle
\end{aligned}$$

For  $0 \leq k < m_1$ ,  $1 \leq i \leq n_1 \mathbf{div} m_1$  and  $0 \leq j \leq n_2 - m_2$ :

$$\begin{aligned}
& (j, P[k]) \notin MPM_1(PR, T[i * m_1 - 1]) \\
\equiv & \quad \{ \text{specification } MPM_1 \} \\
& (j, P[k]) \notin \langle \mathbf{set} \ l, p, r : p \in PR \wedge T[i * m_1 - 1] = lpr : (|l|, p) \rangle \\
\equiv & \quad \{ \text{set calculus, } |P[k]| = m_2, P[k] \in PR \} \\
& T[i * m_1 - 1][j .. j + m_2] \neq P[k] \\
\Rightarrow & \quad \{ \bullet (i + 1) * m_1 - 1 - k \leq n_1 \} \\
& T[i * m_1 - 1 - k .. (i + 1) * m_1 - 1 - k][j .. j + m_2] \neq P
\end{aligned}$$

If the above assumption,  $(i + 1) * m_1 - 1 - k \leq n_1$ , does not hold, then there can also be no match starting at row  $i * m_1 - 1 - k$ , because it would not “fit” within the bounds of the text. So our algorithm will be of the following form:

```

for  $i : 1 \leq i \leq n_1 \mathbf{div} m_1 \rightarrow$ 
  for  $j, p : (j, p) \in MPM_1(PR, T[i * m_1 - 1]) \rightarrow$ 
     $O := O \cup \langle \mathbf{set} \ k : 0 \leq k < m_1 \wedge (i + 1) * m_1 - 1 - k \leq n_1 \wedge P[k] = p \wedge$ 
       $T[i * m_1 - 1 - k .. (i + 1) * m_1 - 1 - k][j .. j + m_2] = P :$ 
       $(i * m_1 - 1 - k, j) \rangle$ 
  rof
rof

```

Note that the multipattern matching function  $MPM_1$  is always called with pattern set  $PR$  as its first parameter. Any precomputation, that depends on  $PR$ , necessary for the multipattern matching algorithm needs to occur only once.

A simple implementation of this algorithm is the following:

```

for  $i : 1 \leq i \leq n_1 \mathbf{div} m_1 \rightarrow$ 
  for  $j, p : (j, p) \in MPM_1(PR, T[i * m_1 - 1]) \rightarrow$ 
    for  $k : 0 \leq k < m_1 \wedge (i + 1) * m_1 - 1 - k \leq n_1 \rightarrow$ 
      if  $P[k] = p \rightarrow$ 
        if  $\text{match}_{T,P}(i * m_1 - 1 - k, j) \rightarrow O := O \cup \{(i * m_1 - 1 - k, j)\}$ 
        ||  $\neg \text{match}_{T,P}(i * m_1 - 1 - k, j) \rightarrow \mathbf{skip}$ 
        fi
      ||  $P[k] \neq p \rightarrow \mathbf{skip}$ 
      fi
    rof
  rof
rof

```

## 5.1 Baeza-Yates and Régner’s *CheckMatch* approach

While the algorithm we presented is correct, the call to  $\text{match}_{T,P}$  can lead to the same comparisons being repeated several times. For instance, from our call to  $MPM_1$  and the guard  $P[k] = p$  we

can conclude that  $T[i * m_1 - 1][j .. j + m_1) = P[k]$ , yet each call to  $\text{match}_{T,P}$  will repeat the computation. Besides that, if more than one row of  $P$  is equal to string  $p$ , the  $\text{match}_{T,P}$  function is called multiple times and there is some overlap in the rows of  $T$  that are being matched each time. This should be obvious if we look at the inner repetition of the algorithm, with an implementation of  $\text{match}_{T,P}$  filled in.

```

for  $k : 0 \leq k < m_1 \wedge (i + 1) * m_1 - 1 - k \leq n_1 \rightarrow$ 
  if  $P[k] = p \rightarrow$ 
     $h, res := 0, \text{true};$ 
    do  $h \neq m_1 \wedge res \rightarrow$ 
      {  $\text{inv.} : res \equiv T[i * m_1 - 1 - k .. i * m_1 - 1 - k + h][j .. j + m_2) = P[0 .. h) \}$ 
       $res := T[i * m_1 - 1 - k + h][j .. j + m_2) = P[h];$ 
       $h := h + 1$ 
    od;
    if  $res \rightarrow O := O \cup \{(i * m_1 - 1 - k, j)\}$ 
    ||  $\neg res \rightarrow \text{skip}$ 
    fi
  ||  $P[k] \neq p \rightarrow \text{skip}$ 
fi
rof

```

This  $\text{match}_{T,P}$  implementation differs slightly from the one presented in section 3.1. Here we consider entire rows in a fixed order, instead of single elements in an unspecified order. We also terminate the computation as soon as a mismatch is discovered, as suggested at the end of section 3.1.

To avoid the aforementioned unnecessary comparisons, we will keep a record of the rows of  $T$  that have already been matched against the rows of  $P$ . To do this, we first introduce a unique index for each row of the pattern  $P$ . We introduce function  $g : \Sigma^* \rightarrow \mathbb{N}$ :

$$\begin{cases} g(x) = \langle \downarrow l : 0 \leq l < m_1 \wedge x = P[l] : l \rangle & \text{if } x \in PR \\ g(x) = m_1 & \text{if } x \notin PR \end{cases} \quad (3)$$

For strings that occur as a row in the pattern  $P$ , the  $g$ -value is equal to the row number of their first occurrence. This is a unique numbering for these strings; that is, for all  $x, y \in PR$ :

$$x = y \equiv g(x) = g(y)$$

In fact, we have the following property for  $g$ :

**Theorem 5.0** *For strings  $x$  and  $y$ , with  $g(y) \neq m_1$ , we have:*

$$x = y \equiv g(x) = g(y)$$

**Proof** Property  $x = y \Rightarrow g(x) = g(y)$  follows directly from the fact that  $g$  is a function. We only need to prove  $g(x) = g(y) \Rightarrow x = y$ , assuming  $g(y) \neq m_1$ .

- Case:  $g(x) = m_1$ .

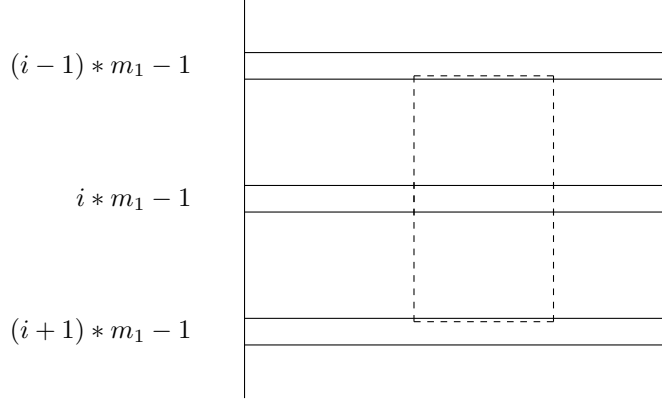


Figure 2: Submatrix of the text, inspected when a match occurs in row  $i * m_1 - 1$

$$\begin{aligned}
& g(x) = g(y) \\
\equiv & \{ g(x) = m_1, g(y) \neq m_1 \} \\
& \text{false} \\
\Rightarrow & \{ \text{predicate calculus} \} \\
& x = y
\end{aligned}$$

- Case:  $g(x) \neq m_1$ .

$$\begin{aligned}
& g(x) = g(y) \\
\Rightarrow & \{ g(x) \neq m_1, g(y) \neq m_1, \text{spec. } g \} \\
& \langle \exists l : 0 \leq l < m_1 : x = P[l] \wedge y = P[l] \rangle \\
\Rightarrow & \{ \text{distributivity, transitivity} = \} \\
& x = y
\end{aligned}$$

□

We can precompute the  $g$ -values for the rows of the pattern. We introduce auxiliary array  $r[0 .. m_1)$  for this purpose, with (for  $0 \leq h < m_1$ ):  $r[h] = g(P[h])$ . We get:

$$r[h] = \langle \downarrow l : 0 \leq l < m_1 \wedge P[h] = P[l] : l \rangle \quad (4)$$

Now, we note that the computation in the “**for**  $k$ ” loop attempts matches in the following submatrix of the text:  $T[(i-1)*m_1 .. (i+1)*m_1 - 1][j .. j + m_2)$  (a band surrounding row  $T[i*m_1 - 1]$ ; see figure 2). This submatrix has  $2 * m_1 - 1$  rows. We introduce array  $f[0 .. 2 * m_1 - 1)$  to store the results of the row comparisons that have already been made. We will maintain the following invariant:

$$\langle \forall h : 0 \leq h < 2 * m_1 - 1 : f[h] = \perp \vee f[h] = g(T[(i-1)*m_1 + h][j .. j + m_2]) \rangle$$

If  $f[h] = \perp$ , that means that the corresponding value of  $g$  has not been computed yet. Initially  $f[h] = \perp$ , for all  $h$  ( $0 \leq h < 2 * m_1 - 1$ ).

Now we have, for  $0 \leq h < 2 * m_1 - 1$  and  $0 \leq l < m_1$ , assuming  $f[h] \neq \perp$ :

$$\begin{aligned}
& T[(i-1) * m_1 + h][j .. j + m_2) = P[l] \\
\equiv & \quad \{ \text{theorem 5.0 (page 34)} \} \\
& g(T[(i-1) * m_1 + h][j .. j + m_2)) = g(P[l]) \\
\equiv & \quad \{ f[h] \neq \perp, \text{invariant, spec. } r \} \\
& f[h] = r[l]
\end{aligned}$$

The algorithm now becomes:

```

for  $h : 0 \leq h < 2 * m_1 - 1 \rightarrow$ 
   $f[h] := \perp$ 
rof;
 $\{ T[i * m_1 - 1][j .. j + m_2) = p \}$ 
 $f[m_1 - 1] := g(p)$ ;
for  $k : 0 \leq k < m_1 \wedge (i + 1) * m_1 - 1 - k \leq n_1 \rightarrow$ 
   $\{ \text{inv. } \langle \forall h : 0 \leq h < 2 * m_1 - 1 : f[h] = \perp \vee f[h] = g(T[(i-1) * m_1 + h][j .. j + m_2)) \rangle \}$ 
  if  $r[k] = g(p) \rightarrow \{ P[k] = p \}$ 
     $h, res := 0, \text{true}$ ;
    do  $h \neq m_1 \wedge res \rightarrow$ 
       $\{ \text{inv.} : res \equiv T[i * m_1 - 1 - k .. i * m_1 - 1 - k + h][j .. j + m_2) = P[0 .. h] \}$ 
      if  $f[m_1 - 1 - k + h] = \perp \rightarrow$ 
         $f[m_1 - 1 - k + h] := g(T[i * m_1 - 1 - k + h][j .. j + m_2))$ 
       $\parallel f[m_1 - 1 - k + h] \neq \perp \rightarrow \text{skip}$ 
      fi;  $\{ f[m_1 - 1 - k + h] \neq \perp \}$ 
       $res := f[m_1 - 1 - k + h] = r[h]$ ;
       $h := h + 1$ 
    od;
    if  $res \rightarrow O := O \cup \{(i * m_1 - 1 - k, j)\}$ 
       $\parallel \neg res \rightarrow \text{skip}$ 
    fi
   $\parallel r[k] \neq g(p) \rightarrow \{ P[k] \neq p \} \text{skip}$ 
fi
rof

```

This is essentially the so-called *Checkmatch* function in Baeza-Yates and Régner's algorithm, as presented in [BYR93]. The difference is that we have not specified how to compute the  $g$ -function yet. We will get to that later, in section 5.4. Note that string variable  $p$  by itself is now no longer relevant; we are only interested in its index,  $g(p)$ .

We should note that we added  $f[m_1 - 1] := g(p)$  to the initialisation of  $f$ . We use the value of  $g(p)$  in several other places in the algorithm (and  $f[m_1 - 1]$  is likely to be inspected), so we may as well initialise that value of  $f$  immediately.

## 5.2 Inspecting fewer pattern rows

In the version of the repetition presented in section 5.1, we examine for every value of  $k$  in the range  $[0 \uparrow ((i + 1) * m_1 - 1 - n_1), m_1)$ , whether row  $k$  of the pattern matches with string  $p$ . If so, we attempt to match the surrounding rows of the text with the corresponding rows of the pattern and conclude whether or not there is a match of our pattern in the text. However, after such an attempted match, we may be able to use the last inspected value of  $f$  to conclude that

we can safely skip a number of values of  $k$ . This can speed up the computation and prevent some unnecessary inspections of the text.

In the previous section we have left the order in which the values of  $k$  are examined unspecified by writing a **for**-statement. An easy way to incorporate the idea of skipping certain subranges is by considering the values of  $k$  in increasing order and, when possible, increasing  $k$  by more than 1. If we replace the **for**-statement by a **do**-statement we get the following algorithm:

```

for  $h : 0 \leq h < 2 * m_1 - 1 \rightarrow$ 
     $f[h] := \perp$ 
rof;
 $f[m_1 - 1] := g(p);$ 
 $k := 0 \uparrow ((i + 1) * m_1 - 1 - n_1);$ 
do  $k < m_1 \rightarrow$ 
    if  $r[k] = g(p) \rightarrow$ 
         $h, res := 0, true;$ 
        do  $h \neq m_1 \wedge res \rightarrow$ 
            if  $f[m_1 - 1 - k + h] = \perp \rightarrow$ 
                 $f[m_1 - 1 - k + h] := g(T[i * m_1 - 1 - k + h][j .. j + m_2])$ 
                 $\parallel f[m_1 - 1 - k + h] \neq \perp \rightarrow$  skip
            fi;
             $res := f[m_1 - 1 - k + h] = r[h];$ 
             $h := h + 1$ 
        od;
        if  $res \rightarrow O := O \cup \{(i * m_1 - 1 - k, j)\};$ 
         $\parallel \neg res \rightarrow$  skip
        fi
         $\{ f[m_1 - 2 - k + h] \neq \perp \}$ 
         $\parallel r[k] \neq g(p) \rightarrow$  skip
    fi;
     $k := k + 1$ 
od

```

We added an extra assertion after the inner repetition:  $f[m_1 - 2 - k + h] \neq \perp$ . Its correctness follows from the program structure and  $0 < m_1$ ; we assume we do not have an empty pattern matrix. If  $f[m_1 - 2 - k + h]$  is not equal to  $\perp$ , we know its value must be either in the interval  $[0, m_1)$  or equal to  $m_1$ . We will investigate both of these cases.

$$\begin{aligned}
 & 0 \leq f[m_1 - 2 - k + h] < m_1 \\
 \equiv & \{ \text{spec. } f \} \\
 & f[m_1 - 2 - k + h] = g(T[i * m_1 - 2 - k + h][j .. j + m_2]) \wedge 0 \leq f[m_1 - 2 - k + h] < m_1 \\
 \equiv & \{ \text{spec. } g \} \\
 & f[m_1 - 2 - k + h] = \langle \downarrow l : 0 \leq l < m_1 \wedge T[i * m_1 - 2 - k + h][j .. j + m_2] = P[l] : l \rangle \wedge \\
 & \quad 0 \leq f[m_1 - 2 - k + h] < m_1 \\
 \Rightarrow & \{ \text{property } \downarrow \} \\
 & \langle \forall l : 0 \leq l < f[m_1 - 2 - k + h] : T[i * m_1 - 2 - k + h][j .. j + m_2] \neq P[l] \rangle \\
 \Rightarrow & \{ \text{equality of matrices} \} \\
 & \langle \forall l : 0 \leq l < f[m_1 - 2 - k + h] : \\
 & \quad T[i * m_1 - 2 - k + h - l .. (i + 1) * m_1 - 2 - k + h - l][j .. j + m_2] \neq P \rangle \\
 \equiv & \{ \text{dummy transformation} \}
 \end{aligned}$$

$$\langle \forall l : i * m_1 - 1 - k + h - f[m_1 - 2 - k + h] \leq l < i * m_1 - 1 - k + h : \\ T[l .. l + m_1][j .. j + m_2] \neq P \rangle$$

So in this case, we can disregard the next  $(f[m_1 - 2 - k + h] - h + 1) \uparrow 1$  values of  $k$ .

$$\begin{aligned} & f[m_1 - 2 - k + h] = m_1 \\ \equiv & \quad \{ \text{invariant} \} \\ & g(T[i * m_1 - 2 - k + h][j .. j + m_2]) = m_1 \\ \equiv & \quad \{ \text{spec. } g \} \\ & T[i * m_1 - 2 - k + h][j .. j + m_2] \notin PR \\ \Rightarrow & \quad \{ PR \text{ contains all rows of } P \} \\ & \neg \langle \exists l : (i - 1) * m_1 - 1 - k + h \leq l < i * m_1 - 1 - k + h : T[l .. l + m_1][j .. j + m_2] = P \rangle \end{aligned}$$

So if we find an  $f$ -value of  $m_1$ , we know we can skip the next  $m_1 - h + 1$  values of  $k$ . (We know that this is a positive number, since  $h \leq m_1$ .) We can rewrite this expression as follows:

$$\begin{aligned} & m_1 - h + 1 \\ = & \quad \{ h \leq m_1 \} \\ & (m_1 - h + 1) \uparrow 1 \\ = & \quad \{ f[m_1 - 2 - k + h] = m_1 \} \\ & (f[m_1 - 2 - k + h] - h + 1) \uparrow 1 \end{aligned}$$

This is the same expression we achieved in the case where  $f[m_1 - 2 - k + h]$  was in the interval  $[0, m_1)$ . The algorithm now becomes:

```

for  $h : 0 \leq h < 2 * m_1 - 1 \rightarrow$ 
   $f[h] := \perp$ 
rof;
 $f[m_1 - 1] := g(p)$ ;
 $k := 0 \uparrow ((i + 1) * m_1 - 1 - n_1)$ ;
do  $k < m_1 \rightarrow$ 
  if  $r[k] = g(p) \rightarrow$ 
     $h, res := 0, \text{true}$ ;
    do  $h \neq m_1 \wedge res \rightarrow$ 
      if  $f[m_1 - 1 - k + h] = \perp \rightarrow$ 
         $f[m_1 - 1 - k + h] := g(T[i * m_1 - 1 - k + h][j .. j + m_2])$ 
      ||  $f[m_1 - 1 - k + h] \neq \perp \rightarrow$  skip
      fi;
       $res := f[m_1 - 1 - k + h] = r[h]$ ;
       $h := h + 1$ 
    od;
    if  $res \rightarrow O := O \cup \{(i * m_1 - 1 - k, j)\}$ ;
    ||  $\neg res \rightarrow$  skip
    fi;
     $k := k + (f[m_1 - 2 - k + h] - h + 1) \uparrow 1$ 
  ||  $r[k] \neq g(p) \rightarrow k := k + 1$ 
fi
od

```

### 5.3 Inspecting only matching pattern rows

So far, we have iterated over values of  $k$  in the range  $[0, m_1)$ , even though we are only interested in those values of  $k$  for which  $P[k]$  is equal to a given string. With a little extra administration, we can make sure we *only* inspect such values of  $k$ . For this purpose, we introduce array  $e[0 .. m_1)$ , with the following specification (for  $0 \leq h < m_1$ ):

$$\begin{cases} e[h] = \langle \downarrow l : h < l < m_1 \wedge P[h] = P[l] : l \rangle & \text{if } \langle \exists l : h < l < m_1 : P[h] = P[l] \rangle \\ e[h] = m_1 & \text{if } \neg \langle \exists l : h < l < m_1 : P[h] = P[l] \rangle \end{cases} \quad (5)$$

A shorter, equivalent specification is the following:

$$e[h] = \langle \downarrow l : h < l < m_1 \wedge P[h] = P[l] : l \rangle \downarrow m_1 \quad (6)$$

Note that we have, for  $0 \leq h < m_1$ :  $h < e[h]$ .

Using our earlier definition of function  $g$  ((3) on page 34), we can conclude that all occurrences of a string  $x$  as a row in our pattern are  $g(x)$ ,  $e[g(x)]$ ,  $e[e[g(x)]]$ ,  $\dots$ , until we encounter  $m_1$ . To prove this claim formally, we denote this set of row numbers by  $D(g(x))$ , where  $D$  is defined as follows:

$$\begin{aligned} D(m_1) &= \emptyset \\ D(h) &= \{h\} \cup D(e[h]) \quad \text{for } 0 \leq h < m_1 \end{aligned}$$

We need to prove:  $D(g(x)) = \langle \text{set } l : 0 \leq l < m_1 \wedge x = P[l] : l \rangle$ . First we prove the following lemma.

**Lemma 5.1** *For  $0 \leq h \leq m_1$ , we have:*

$$D(h) = \langle \text{set } l : h \leq l < m_1 \wedge P[h] = P[l] : l \rangle$$

*Technically, since  $h = m_1$  is included in this lemma, this expression is defined only if  $P[m_1]$  exists. For this purpose we can assume that there is an imaginary row  $P[m_1]$ ; its value is irrelevant.*

**Proof** We prove this theorem by induction, with decreasing values of  $h$ .

- Case  $h = m_1$ :

$$\begin{aligned} & \langle \text{set } l : m_1 \leq l < m_1 \wedge P[h] = P[l] : l \rangle \\ &= \{ \text{empty domain} \} \\ &= \emptyset \\ &= \{ \text{def. } D \} \\ &= D(m_1) \end{aligned}$$

- Case  $0 \leq h < m_1$ :

Induction hypothesis:  $D(e[h]) = \langle \text{set } l : e[h] \leq l < m_1 \wedge P[e[h]] = P[l] : l \rangle$ . Recall that  $h < e[h]$ , for  $0 \leq h < m_1$ , and that we assumed that an imaginary row  $P[m_1]$  exists.

$$\begin{aligned}
& \langle \text{set } l : h \leq l < m_1 \wedge P[h] = P[l] : l \rangle \\
= & \{ h < m_1; \text{ split off: } l = h \} \\
& \{h\} \cup \langle \text{set } l : h < l < m_1 \wedge P[h] = P[l] : l \rangle \\
= & \{ \text{from spec. } e: \langle \forall l : h < l < e[h] : P[h] \neq P[l]; h < e[h] \rangle \} \\
& \{h\} \cup \langle \text{set } l : e[h] \leq l < m_1 \wedge P[h] = P[l] : l \rangle \\
= & \{ \text{case } 0 \leq e[h] < m_1: P[h] = P[e[h]]; \text{ case } e[h] = m_1: \text{ empty domain } \} \\
& \{h\} \cup \langle \text{set } l : e[h] \leq l < m_1 \wedge P[e[h]] = P[l] : l \rangle \\
= & \{ \text{induction hypothesis } \} \\
& \{h\} \cup D(e[h]) \\
= & \{ \text{def. } D \} \\
& D(h)
\end{aligned}$$

□

**Theorem 5.2** *For any string  $x$ , we have:*

$$D(g(x)) = \langle \text{set } l : 0 \leq l < m_1 \wedge x = P[l] : l \rangle$$

**Proof**

- Case  $x \in PR$ :

$$\begin{aligned}
& \langle \text{set } l : 0 \leq l < m_1 \wedge x = P[l] : l \rangle \\
= & \{ \text{from spec. } g: \langle \forall l : 0 \leq l < g(x) : x \neq P[l]; 0 \leq g(x) \rangle \} \\
& \langle \text{set } l : g(x) \leq l < m_1 \wedge x = P[l] : l \rangle \\
= & \{ \text{spec. } g, x \in PR: x = P[g(x)] \} \\
& \langle \text{set } l : g(x) \leq l < m_1 \wedge P[g(x)] = P[l] : l \rangle \\
= & \{ \text{lemma 5.1 } \} \\
& D(g(x))
\end{aligned}$$

- Case  $x \notin PR$ :

$$\begin{aligned}
& \langle \text{set } l : 0 \leq l < m_1 \wedge x = P[l] : l \rangle \\
= & \{ x \notin PR \} \\
& \emptyset \\
= & \{ \text{def. } D \} \\
& D(m_1) \\
= & \{ x \notin PR, \text{ thus } g(x) = m_1 \} \\
& D(g(x))
\end{aligned}$$

□

So in conclusion, the only values of  $k$  we need to inspect are the ones in the set  $D(g(p))$ ; that is:  $g(p), e[g(p)], e[e[g(p)]], \dots$ , until we find a value that is equal to  $m_1$ . This leads to the following version of the inner repetition of our Baeza-Yates and Régnyier algorithm:

```

for  $h : 0 \leq h < 2 * m_1 - 1 \rightarrow$ 
   $f[h] := \perp$ 
rof;
 $f[m_1 - 1] := g(p);$ 
 $k := g(p);$ 
do  $k \neq m_1 \rightarrow$ 
  if  $(i + 1) * m_1 - 1 - k \leq n_1 \rightarrow$ 
     $h, res := 0, \text{true};$ 
    do  $h \neq m_1 \wedge res \rightarrow$ 
      if  $f[m_1 - 1 - k + h] = \perp \rightarrow$ 
         $f[m_1 - 1 - k + h] := g(T[i * m_1 - 1 - k + h][j .. j + m_2])$ 
         $\parallel f[m_1 - 1 - k + h] \neq \perp \rightarrow \text{skip}$ 
      fi;
       $res := f[m_1 - 1 - k + h] = r[h];$ 
       $h := h + 1$ 
    od;
    if  $res \rightarrow O := O \cup \{(i * m_1 - 1 - k, j)\};$ 
     $\parallel \neg res \rightarrow \text{skip}$ 
    fi
     $\parallel n_1 < (i + 1) * m_1 - 1 - k \rightarrow \text{skip}$ 
    fi;
     $k := e[k]$ 
  od

```

The improvement we introduced in section 5.2 can be applied here as well. This allows us to adjust the algorithm as follows:

```

for  $h : 0 \leq h < 2 * m_1 - 1 \rightarrow$ 
   $f[h] := \perp$ 
rof;
 $f[m_1 - 1] := g(p);$ 
 $k := g(p);$ 
do  $k \neq m_1 \rightarrow$ 
  if  $(i + 1) * m_1 - 1 - k \leq n_1 \rightarrow$ 
     $h, res := 0, \text{true};$ 
    do  $h \neq m_1 \wedge res \rightarrow$ 
      if  $f[m_1 - 1 - k + h] = \perp \rightarrow$ 
         $f[m_1 - 1 - k + h] := g(T[i * m_1 - 1 - k + h][j .. j + m_2])$ 
         $\parallel f[m_1 - 1 - k + h] \neq \perp \rightarrow \text{skip}$ 
      fi;
       $res := f[m_1 - 1 - k + h] = r[h];$ 
       $h := h + 1$ 
    od;
    if  $res \rightarrow O := O \cup \{(i * m_1 - 1 - k, j)\};$ 
     $\parallel \neg res \rightarrow \text{skip}$ 
    fi;
     $\tilde{k} := k;$ 
     $k := e[k];$ 
    do  $k < (\tilde{k} + f[m_1 - 2 - \tilde{k} + h] - h + 1) \downarrow m_1 \rightarrow$ 
       $k := e[k]$ 
    od
     $\parallel n_1 < (i + 1) * m_1 - 1 - k \rightarrow k := e[k]$ 
  fi

```

od

### Computation of $e$

So far we have ignored the question how to initialise the values of array  $e$ . We can of course easily come up with an implementation that, for each pattern row, searches for that row's next occurrence in the pattern using a bounded linear search. Using array  $r$ , this would give rise to an  $\mathcal{O}(m_1^2)$  algorithm. However, we can give an  $\mathcal{O}(m_1)$  algorithm.

Our postcondition is the specification of  $e$ , as given in (5) and, equivalently, (6) on page 39. We introduce integer variable  $i$ , along with the following invariants:

$$\begin{aligned} P0: & 0 \leq i \leq m_1 \\ P1: & \langle \forall h : 0 \leq h < m_1 : e[h] = \langle \downarrow l : h < l < i \wedge P[h] = P[l] : l \rangle \downarrow m_1 \rangle \end{aligned}$$

Then the postcondition is established when  $i = m_1$ . Initially, we can choose  $i = 0$  and  $e[h] = m_1$ , for  $0 \leq h < m_1$ . Ad  $P1(i := i + 1)$ :

- For  $h: i \leq h < m_1$ :

$$\begin{aligned} & \langle \downarrow l : h < l < i + 1 \wedge P[h] = P[l] : l \rangle \downarrow m_1 \\ = & \{ i \leq h: \text{empty domain} \} \\ & m_1 \\ = & \{ i \leq h: \text{empty domain} \} \\ & \langle \downarrow l : h < l < i \wedge P[h] = P[l] : l \rangle \downarrow m_1 \\ = & \{ P1 \} \\ & e[h] \end{aligned}$$

- For  $h: 0 \leq h < i$ :

$$\begin{aligned} & \langle \downarrow l : h < l < i + 1 \wedge P[h] = P[l] : l \rangle \downarrow m_1 \\ = & \{ h < i; \text{split off } l = i \} \\ & \begin{cases} \langle \downarrow l : h < l < i \wedge P[h] = P[l] : l \rangle \downarrow m_1 & \text{if } P[h] \neq P[i] \\ \langle \downarrow l : h < l < i \wedge P[h] = P[l] : l \rangle \downarrow m_1 \downarrow i & \text{if } P[h] = P[i] \end{cases} \\ = & \{ P1 \} \\ & \begin{cases} e[h] & \text{if } P[h] \neq P[i] \\ e[h] \downarrow i & \text{if } P[h] = P[i] \end{cases} \\ = & \{ \text{def. } \downarrow, \text{predicate calculus} \} \\ & \begin{cases} e[h] & \text{if } P[h] \neq P[i] \vee e[h] < i \\ i & \text{if } P[h] = P[i] \wedge i \leq e[h] \end{cases} \\ = & \{ P1: e[h] = m_1 \vee e[h] < i; i \leq m_1 \} \\ & \begin{cases} e[h] & \text{if } P[h] \neq P[i] \vee e[h] \neq m_1 \\ i & \text{if } P[h] = P[i] \wedge e[h] = m_1 \end{cases} \end{aligned}$$

So the only values of  $h$  for which  $e[h]$  needs to be updated are the ones for which the following holds:  $P[h] = P[i] \wedge e[h] = m_1$ .

$$\begin{aligned}
& P[h] = P[i] \wedge e[h] = m_1 \\
\equiv & \{ P1 \} \\
& P[h] = P[i] \wedge \langle \downarrow l : h < l < i \wedge P[h] = P[l] : l \rangle \downarrow m_1 = m_1 \\
\equiv & \{ \text{transitivity} = \} \\
& P[h] = P[i] \wedge \langle \downarrow l : h < l < i \wedge P[i] = P[l] : l \rangle \downarrow m_1 = m_1 \\
\equiv & \{ i \leq m_1, \text{definition } \downarrow \} \\
& P[h] = P[i] \wedge \langle \forall l : h < l < i : P[i] \neq P[l] \rangle \\
\equiv & \{ \text{def. } \uparrow, 0 \leq h < i \} \\
& h = \langle \uparrow l : 0 \leq l < i \wedge P[i] = P[l] : l \rangle \\
\equiv & \{ g(P[i]) \neq m_1, \text{theorem 5.0 (see page 34)} \} \\
& h = \langle \uparrow l : 0 \leq l < i \wedge g(P[i]) = g(P[l]) : l \rangle \\
\equiv & \{ \text{spec. } r \} \\
& h = \langle \uparrow l : 0 \leq l < i \wedge r[i] = r[l] : l \rangle
\end{aligned}$$

We conclude that there is at most one  $h$  for which  $e[h]$  needs to be updated, specifically:  $\langle \uparrow l : 0 \leq l < i \wedge r[i] = r[l] : l \rangle$ , if this value is greater than  $-\infty$ . To find this value, we introduce auxiliary array  $aux[0 .. m_1]$ , with the following invariant.

$$P2 : \langle \forall j : 0 \leq j < m_1 : aux[j] = \langle \uparrow l : 0 \leq l < i \wedge j = r[l] : l \rangle \rangle$$

Initially, we have  $i = 0$  and therefore  $aux[j] = -\infty$ , for  $0 \leq j < m_1$ . Ad  $P2(i := i + 1)$ :

$$\begin{aligned}
& \langle \uparrow l : 0 \leq l < i + 1 \wedge j = r[l] : l \rangle \\
= & \{ \text{split off: } l = i \} \\
& \begin{cases} \langle \uparrow l : 0 \leq l < i \wedge j = r[l] : l \rangle & \text{if } j \neq r[i] \\ \langle \uparrow l : 0 \leq l < i \wedge j = r[l] : l \rangle \uparrow i & \text{if } j = r[i] \end{cases} \\
= & \{ P2, \text{math} \} \\
& \begin{cases} aux[j] & \text{if } j \neq r[i] \\ i & \text{if } j = r[i] \end{cases}
\end{aligned}$$

In conclusion, this gives rise to the following initialisation algorithm for  $e$ :

```

for  $i : 0 \leq i < m_1 \rightarrow$ 
   $e[i], aux[i] := m_1, -\infty$ 
rof;
 $i := 0;$ 
do  $i \neq m_1 \rightarrow$ 
  if  $aux[r[i]] \neq -\infty \rightarrow$ 
     $e[aux[r[i]]] := i$ 
  ||  $aux[r[i]] = -\infty \rightarrow$ 
    skip
  fi;
   $aux[r[i]] := i;$ 
   $i := i + 1$ 
od

```

In an implementation we can represent  $-\infty$  by any negative number (or even any number outside of the range  $[0, m_1)$ ), for example:  $-1$ .

## 5.4 Computation of unique row indices

As we have seen in section 5.1, the value of string variable  $p$  is no longer of interest to us. Instead, we only use  $g(p)$  (defined in (3) on page 34). For this reason, we want to introduce a new specification for one-dimensional multipattern matching, which returns the value of  $g(p)$  immediately.

$$MPM'_1(PS, t) = \langle \text{set } l, p, r : p \in PS \wedge t = lpr : (|l|, g(p)) \rangle$$

The question remains, how to compute  $g$ -values. The method presented in [BYR93] is by building the Aho-Corasick automaton based on set  $PR$ , with an output function  $\lambda$  that returns  $g$ -values.

More formally, our Aho-Corasick automaton of  $PR$  is a Moore machine  $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ . As in section 4.3, we will not go into every detail on how an Aho-Corasick automaton can be constructed; instead we refer to [WZ92], [WZ93] or [NR02].

Our output alphabet  $\Delta$  is equal to  $[0, m_1]$ . For each state  $q$ , we want to establish:  $\lambda(q) = g(w_q)$ , where  $w_q$  is the string consisting of the labels of the shortest path from start state  $q_0$  to  $q$ . For non-accepting states  $q$  we can simply define  $\lambda(q) = m_1$ ; for final states  $q$ ,  $\lambda(q)$  needs to be equal to the index of the recognised row of  $P$ .

If we construct our Aho-Corasick automaton like a trie, by adding the rows of  $PR$  one by one, in increasing order, we can establish this quite easily. Whenever we add a *new final* state  $q$  while processing row  $l$  of our pattern matrix, we set  $\lambda(q) = l$ .

Now we have:  $g(p) = \lambda(\delta^*(q_0, p))$ . We can also use this Aho-Corasick automaton for our implementation of  $MPM'_1$ .

During the construction of the automaton we can also fill array  $r$  (as specified in (4) on page 35). After processing a row  $l$  of the pattern, the value of  $g(P[l])$  is known and, by specification, this is the required value of  $r[l]$ .

## 5.5 Generalisations

The Baeza-Yates and Régnier algorithm can be generalised to be used for two-dimensional multi-pattern matching. Say we have  $l$   $m_1 \times m_2$  two-dimensional patterns:  $P_0, P_1, \dots, P_{l-1}$ . Then our postcondition becomes:

$$O = \langle \text{set } i_1, i_2, j : \begin{array}{l} 0 \leq i_1 \leq n_1 - m_1 \wedge 0 \leq i_2 \leq n_2 - m_2 \wedge 0 \leq j < l \wedge \\ T[i_1 .. i_1 + m_1][i_2 .. i_2 + m_2] = P_j : (i_1, i_2, j) \end{array} \rangle$$

We introduce a new auxiliary pattern matrix  $Q[0 .. l * m_1][0 .. m_2)$ , where (for  $0 \leq i < l * m_1$ ):

$$Q[i] = P_{i \text{ div } m_1}[i \bmod m_1]$$

Similarly to  $PR$  in the single-pattern searching algorithm, we introduce the set  $QR$ :

$$QR = \langle \text{set } i : 0 \leq i < l * m_1 : Q[i] \rangle$$

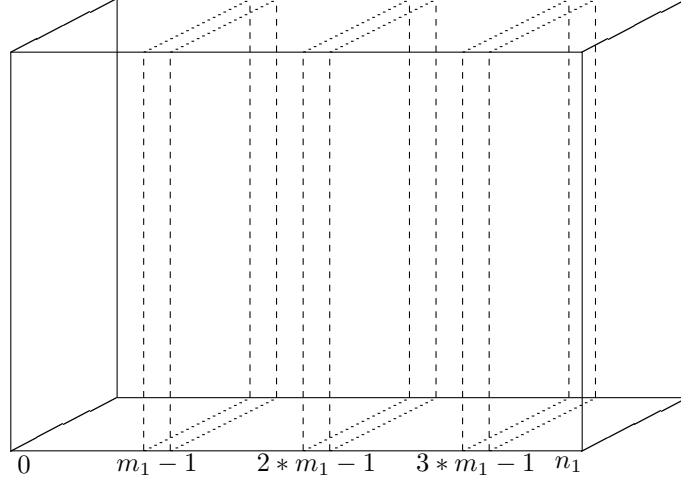


Figure 3: The Baeza-Yates algorithm idea, applied in three dimensions

Or, equivalently:

$$QR = \langle \text{set } i, j : 0 \leq i < m_1 \wedge 0 \leq j < l : P_j[i] \rangle$$

Now we can apply one-dimensional multipattern matching to search for the strings in  $QR$ . Once a match with one of these strings  $q \in QR$  has been found in one of the text's rows, we need to check whether there is a match on that position in the text, with any of the pattern matrices in which  $q$  occurs as a row. That is, for each row  $Q[i]$  that is equal to  $q$ , we check if there is a match with pattern matrix  $P_{i \text{ div } m_1}$ .

This boils down to only a slight modification of the Baeza-Yates and Régnier algorithm (or any of the variations discussed). We will however need a slightly different definition for our  $g$  function and auxiliary array  $r$  (and  $e$ , for the two algorithms presented in section 5.3). For any string  $x$  and  $0 \leq i < l * m_1$ :

$$\begin{cases} g(x) = \langle \downarrow j : 0 \leq j < l * m_1 \wedge x = Q[j] : j \rangle & \text{if } x \in QR \\ g(x) = l * m_1 & \text{if } x \notin QR \end{cases}$$

$$r[i] = g(Q[i])$$

$$\begin{cases} e[i] = \langle \downarrow j : i < j < l * m_1 \wedge Q[i] = Q[j] : l \rangle & \text{if } \langle \exists j : i < j < l * m_1 : Q[i] = Q[j] \rangle \\ e[i] = l * m_1 & \text{if } \neg \langle \exists j : i < j < l * m_1 : Q[i] = Q[j] \rangle \end{cases}$$

The values of  $g$ ,  $r$  and  $e$  can still be computed in the ways presented in sections 5.3 and 5.4.

Matching in multiple dimensions is also possible using the Baeza-Yates and Régnier approach. For matching in  $n + 1$  dimensions ( $1 \leq n$ ) we use an  $n$ -dimensional multipattern matching algorithm. Here, too, we need to choose a different definition of  $g$ , namely a function that works on  $n$ -dimensional shapes. Also, if we are matching in more than two dimensions, we cannot simply use an Aho-Corasick automaton for multipattern matching and for computing our  $g$ -function.

For example, if we are matching in three dimensions, we regard the  $m_1 \times m_2 \times m_3$  pattern as a sequence of  $m_1$  two-dimensional matrices and the  $n_1 \times n_2 \times n_3$  text as a sequence of  $n_1$  two-

dimensional matrices. We then use two-dimensional multipattern matching to match every  $m_1^{\text{th}}$  matrix of the text against the matrices of the pattern. (See also figure 3.) For every match found we check if there are any actual matches in three dimensions on that position.



## 6 Polcar

### 6.0 Introduction

In this section, we will present the algorithm that was presented by Tomáš Polcar in [Pol04]; a shorter version of the same article can be found in [MP04].

The algorithm uses tessellation automata, a data structure introduced in [IN77]. In this presentation however, we will attempt to derive the algorithm by manipulating sets of submatrices of the text.

### 6.1 Derivation

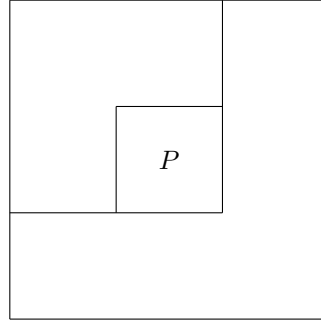


Figure 4: A pattern occurrence as a suffix of a prefix of the text

We start by rewriting our postcondition, so that we view an occurrence of the pattern as a suffix of a prefix of the text (see figure 4).

$$\begin{aligned}
& \langle \text{set } i_1, i_2 : 0 \leq i_1 \leq n_1 - m_1 \wedge 0 \leq i_2 \leq n_2 - m_2 \wedge \\
& \quad T[i_1 .. i_1 + m_1][i_2 .. i_2 + m_2] = P : (i_1, i_2) \rangle \\
= & \quad \{ \text{dummy transformation} \} \\
& \langle \text{set } i_1, i_2 : m_1 \leq i_1 \leq n_1 \wedge m_2 \leq i_2 \leq n_2 \wedge \\
& \quad T[i_1 - m_1 .. i_1][i_2 - m_2 .. i_2] = P : (i_1 - m_1, i_2 - m_2) \rangle \\
= & \quad \{ 0 \leq i_1 - m_1, 0 \leq i_2 - m_2, \text{def. suff} \} \\
& \langle \text{set } i_1, i_2 : m_1 \leq i_1 \leq n_1 \wedge m_2 \leq i_2 \leq n_2 \wedge \\
& \quad P \in \text{suff}(T[0 .. i_1][0 .. i_2]) : (i_1 - m_1, i_2 - m_2) \rangle \\
= & \quad \{ P \in \text{pref}(P) \} \\
& \langle \text{set } i_1, i_2 : m_1 \leq i_1 \leq n_1 \wedge m_2 \leq i_2 \leq n_2 \wedge \\
& \quad P \in \text{suff}(T[0 .. i_1][0 .. i_2]) \cap \text{pref}(P) : (i_1 - m_1, i_2 - m_2) \rangle \\
= & \quad \{ \text{theorem C.0: } \mathcal{ES}_{m_1, m_2} \subseteq \text{suff}(T[0 .. i_1][0 .. i_2]) \cap \text{pref}(P) \} \\
& \langle \text{set } i_1, i_2 : m_1 \leq i_1 \leq n_1 \wedge m_2 \leq i_2 \leq n_2 \wedge \\
& \quad P \in (\text{suff}(T[0 .. i_1][0 .. i_2]) \cap \text{pref}(P)) \cup \mathcal{ES}_{m_1, m_2} : (i_1 - m_1, i_2 - m_2) \rangle \\
= & \quad \{ \bullet \text{ introduction of function } f \} \\
& \langle \text{set } i_1, i_2 : m_1 \leq i_1 \leq n_1 \wedge m_2 \leq i_2 \leq n_2 \wedge P \in f(i_1, i_2) : (i_1 - m_1, i_2 - m_2) \rangle
\end{aligned}$$

We introduce function  $f : [0, n_1] \times [0, n_2] \rightarrow \mathcal{P}(\mathcal{M}_2(\Sigma))$ , with the following specification:

$$f(i_1, i_2) = (\text{suff}(T[0 .. i_1][0 .. i_2]) \cap \text{pref}(P)) \cup \mathcal{ES}_{m_1, m_2}$$

We have introduced this function because we can define  $f$  recursively and then (hopefully) construct an algorithm to compute  $f$ . Before we get to that, we introduce the following lemmas.

Lemma 6.0 concerns how the set of suffixes of a prefix of  $T$  can be split into a set of empty matrices and a set of nonempty matrices. Lemma 6.1 shows how we can conclude whether a submatrix of the text is a prefix of the pattern.

**Lemma 6.0** For  $0 \leq i_1 < n_1$  and  $0 \leq i_2 < n_2$ :

$$\begin{aligned} & \text{suff}(T[0 .. i_1 + 1][0 .. i_2 + 1]) = \\ & \langle \text{set } j_1, j_2 : 0 \leq j_1 \leq i_1 \wedge 0 \leq j_2 \leq i_2 : T[j_1 .. i_1 + 1][j_2 .. i_2 + 1] \rangle \cup \mathcal{ES}_{i_1+1, i_2+1} \end{aligned}$$

**Proof**

$$\begin{aligned} & \text{suff}(T[0 .. i_1 + 1][0 .. i_2 + 1]) \\ = & \{ \text{def. suff} \} \\ & \langle \text{set } j_1, j_2 : 0 \leq j_1 \leq i_1 + 1 \wedge 0 \leq j_2 \leq i_2 + 1 : T[j_1 .. i_1 + 1][j_2 .. i_2 + 1] \rangle \\ = & \{ \text{split off } j_1 = i_1 + 1 \text{ and } j_2 = i_2 + 1 \} \\ & \langle \text{set } j_1, j_2 : 0 \leq j_1 \leq i_1 \wedge 0 \leq j_2 \leq i_2 : T[j_1 .. i_1 + 1][j_2 .. i_2 + 1] \rangle \cup \\ & \langle \text{set } j_2 : 0 \leq j_2 \leq i_2 + 1 : \mathcal{E}_{0, i_2+1-j_2} \rangle \cup \langle \text{set } j_1 : 0 \leq j_1 \leq i_1 + 1 : \mathcal{E}_{i_1+1-j_1, 0} \rangle \\ = & \{ \text{dummy transformation: } j_1 := i_1 + 1 - j \} \\ & \langle \text{set } j_1, j_2 : 0 \leq j_1 \leq i_1 \wedge 0 \leq j_2 \leq i_2 : T[j_1 .. i_1 + 1][j_2 .. i_2 + 1] \rangle \cup \\ & \langle \text{set } j : 0 \leq j \leq i_2 + 1 : \mathcal{E}_{0, j} \rangle \cup \langle \text{set } j : 0 \leq j \leq i_1 + 1 : \mathcal{E}_{j, 0} \rangle \\ = & \{ \text{def. } \mathcal{ES} \} \\ & \langle \text{set } j_1, j_2 : 0 \leq j_1 \leq i_1 \wedge 0 \leq j_2 \leq i_2 : T[j_1 .. i_1 + 1][j_2 .. i_2 + 1] \rangle \cup \mathcal{ES}_{i_1+1, i_2+1} \end{aligned}$$

□

**Lemma 6.1** For  $i_1, j_1, i_2, j_2 \in \mathbb{N}$ , with  $j_1 \leq i_1 < n_1$  and  $j_2 \leq i_2 < n_2$ :

- if  $i_1 + 1 - j_1 \leq m_1 \wedge i_2 + 1 - j_2 \leq m_2$ :

$$\begin{aligned} & T[j_1 .. i_1 + 1][j_2 .. i_2 + 1] \in \text{pref}(P) \equiv \\ & T[j_1 .. i_1][j_2 .. i_2 + 1] \in \text{pref}(P) \wedge T[j_1 .. i_1 + 1][j_2 .. i_2] \in \text{pref}(P) \wedge \\ & T[i_1, i_2] = P[i_1 - j_1, i_2 - j_2] \end{aligned}$$

- if  $m_1 < i_1 + 1 - j_1 \vee m_2 < i_2 + 1 - j_2$ :

$$T[j_1 .. i_1 + 1][j_2 .. i_2 + 1] \notin \text{pref}(P)$$

**Proof**

- Case:  $i_1 + 1 - j_1 \leq m_1 \wedge i_2 + 1 - j_2 \leq m_2$  (our submatrix is not larger than the pattern)

$$\begin{aligned}
& T[j_1 \dots i_1 + 1][j_2 \dots i_2 + 1] \in \text{pref}(P) \\
\equiv & \{ \text{def. pref} \} \\
& T[j_1 \dots i_1 + 1][j_2 \dots i_2 + 1] \in \langle \text{set } l_1, l_2 : 0 \leq l_1 \leq m_1 \wedge 0 \leq l_2 \leq m_2 : P[0 \dots l_1][0 \dots l_2] \rangle \\
\equiv & \{ i_1 - j_1 < m_1, i_2 - j_2 < m_2 \} \\
& T[j_1 \dots i_1 + 1][j_2 \dots i_2 + 1] = P[0 \dots i_1 + 1 - j_1][0 \dots i_2 + 1 - j_2] \\
\equiv & \{ j_1 \leq i_1, j_2 \leq i_2, \text{def. equality of matrices} \} \\
& T[j_1 \dots i_1][j_2 \dots i_2 + 1] = P[0 \dots i_1 - j_1][0 \dots i_2 + 1 - j_2] \wedge \\
& T[j_1 \dots i_1 + 1][j_2 \dots i_2] = P[0 \dots i_1 + 1 - j_1][0 \dots i_2 - j_2] \wedge \\
& T[i_1, i_2] = P[i_1 - j_1, i_2 - j_2] \\
\equiv & \{ i_1 - j_1 < m_1, i_2 - j_2 < m_2, \text{def. pref} \} \\
& T[j_1 \dots i_1][j_2 \dots i_2 + 1] \in \text{pref}(P) \wedge T[j_1 \dots i_1 + 1][j_2 \dots i_2] \in \text{pref}(P) \wedge \\
& T[i_1, i_2] = P[i_1 - j_1, i_2 - j_2]
\end{aligned}$$

- Case:  $m_1 < i_1 + 1 - j_1 \vee m_2 < i_2 + 1 - j_2$  (our submatrix is larger than the pattern)

$$\begin{aligned}
& T[j_1 \dots i_1 + 1][j_2 \dots i_2 + 1] \in \text{pref}(P) \\
\equiv & \{ \text{def. pref} \} \\
& T[j_1 \dots i_1 + 1][j_2 \dots i_2 + 1] \in \langle \text{set } l_1, l_2 : 0 \leq l_1 \leq m_1 \wedge 0 \leq l_2 \leq m_2 : P[0 \dots l_1][0 \dots l_2] \rangle \\
\equiv & \{ m_1 \leq i_1 - j_1 \vee m_2 \leq i_2 - j_2 \} \\
& \text{false}
\end{aligned}$$

□

Now we can get back to finding a recursive definition for  $f$ . For  $0 \leq i_2 \leq n_2$  we derive:

$$\begin{aligned}
& f(0, i_2) \\
= & \{ \text{spec. } f \} \\
& (\text{suff}(T[0 \dots 0][0 \dots i_2]) \cap \text{pref}(P)) \cup \mathcal{ES}_{m_1, m_2} \\
= & \{ \text{def. submatrix} \} \\
& (\text{suff}(\mathcal{E}_{0, i_2}) \cap \text{pref}(P)) \cup \mathcal{ES}_{m_1, m_2} \\
= & \{ \text{def. suff, one point rule, def. submatrix} \} \\
& (\langle \text{set } j : 0 \leq j \leq i_2 : \mathcal{E}_{0, j} \rangle \cap \text{pref}(P)) \cup \mathcal{ES}_{m_1, m_2} \\
= & \{ \text{def. } \mathcal{ES} \} \\
& (\mathcal{ES}_{0, i_2} \cap \text{pref}(P)) \cup \mathcal{ES}_{m_1, m_2} \\
= & \{ \text{theorem C.1} \} \\
& \mathcal{ES}_{0, i_2 \downarrow m_2} \cup \mathcal{ES}_{m_1, m_2} \\
= & \{ 0 \leq m_1, i_2 \downarrow m_2 \leq m_2 \} \\
& \mathcal{ES}_{m_1, m_2}
\end{aligned}$$

Symmetrically, we know that  $f(i_1, 0) = \mathcal{ES}_{m_1, m_2}$  (for  $0 \leq i_1 \leq n_1$ ). We will now examine expression  $f(i_1 + 1, i_2 + 1)$ , for  $0 \leq i_1 < n_1$  and  $0 \leq i_2 < n_2$ .

$$\begin{aligned}
& f(i_1 + 1, i_2 + 1) \\
= & \{ \text{spec. } f \}
\end{aligned}$$

$$\begin{aligned}
& (\text{suff}(T[0 \dots i_1 + 1][0 \dots i_2 + 1]) \cap \text{pref}(P)) \cup \mathcal{ES}_{m_1, m_2} \\
= & \{ \text{lemma 6.0} \} \\
& ((\langle \text{set } j_1, j_2 : 0 \leq j_1 \leq i_1 \wedge 0 \leq j_2 \leq i_2 : T[j_1 \dots i_1 + 1][j_2 \dots i_2 + 1] \rangle \cup \mathcal{ES}_{i_1+1, i_2+1}) \cap \text{pref}(P)) \cup \\
& \mathcal{ES}_{m_1, m_2} \\
= & \{ \cap \text{ over } \cup; \text{theorem C.1} \} \\
& (\langle \text{set } j_1, j_2 : 0 \leq j_1 \leq i_1 \wedge 0 \leq j_2 \leq i_2 : T[j_1 \dots i_1 + 1][j_2 \dots i_2 + 1] \rangle \cap \text{pref}(P)) \cup \\
& \mathcal{ES}_{(i_1+1) \downarrow m_1, (i_2+1) \downarrow m_2} \cup \mathcal{ES}_{m_1, m_2} \\
= & \{ \text{def. } \mathcal{ES} \} \\
& (\langle \text{set } j_1, j_2 : 0 \leq j_1 \leq i_1 \wedge 0 \leq j_2 \leq i_2 : T[j_1 \dots i_1 + 1][j_2 \dots i_2 + 1] \rangle \cap \text{pref}(P)) \cup \mathcal{ES}_{m_1, m_2} \\
= & \{ \text{property } \cap \} \\
& \langle \text{set } j_1, j_2 : 0 \leq j_1 \leq i_1 \wedge 0 \leq j_2 \leq i_2 \wedge T[j_1 \dots i_1 + 1][j_2 \dots i_2 + 1] \in \text{pref}(P) : \\
& T[j_1 \dots i_1 + 1][j_2 \dots i_2 + 1] \rangle \cup \mathcal{ES}_{m_1, m_2} \\
= & \{ \text{lemma 6.1} \} \\
& \langle \text{set } j_1, j_2 : 0 \leq j_1 \leq i_1 \wedge 0 \leq j_2 \leq i_2 \wedge i_1 - j_1 < m_1 \wedge i_2 - j_2 < m_2 \wedge \\
& T[j_1 \dots i_1][j_2 \dots i_2 + 1] \in \text{pref}(P) \wedge T[j_1 \dots i_1 + 1][j_2 \dots i_2] \in \text{pref}(P) \wedge \\
& T[i_1, i_2] = P[i_1 - j_1, i_2 - j_2] : T[j_1 \dots i_1 + 1][j_2 \dots i_2 + 1] \rangle \cup \mathcal{ES}_{m_1, m_2} \\
= & \{ \text{prop. matrices} \} \\
& \langle \text{set } j_1, j_2 : 0 \leq j_1 \leq i_1 \wedge 0 \leq j_2 \leq i_2 \wedge i_1 - j_1 < m_1 \wedge i_2 - j_2 < m_2 \wedge \\
& \left[ \begin{array}{c|c} T[j_1 \dots i_1][j_2 \dots i_2] & T[j_1 \dots i_1][i_2] \\ \hline T[j_1 \dots i_1 + 1][j_2 \dots i_2] & T[i_1, i_2] \end{array} \right] \in \text{pref}(P) \wedge \\
& \left[ \begin{array}{c|c} T[j_1 \dots i_1][j_2 \dots i_2] & T[i_1, i_2] \\ \hline T[i_1, i_2] & T[i_1, i_2] \end{array} \right] \in \text{pref}(P) \wedge \\
& T[i_1, i_2] = P[i_1 - j_1, i_2 - j_2] : \left[ \begin{array}{c|c} T[j_1 \dots i_1][j_2 \dots i_2] & T[j_1 \dots i_1][i_2] \\ \hline T[i_1, i_2] & T[i_1, i_2] \end{array} \right] \rangle \cup \mathcal{ES}_{m_1, m_2} \\
= & \{ \text{one-point rule} \} \\
& \langle \text{set } j_1, j_2, A, b, c : 0 \leq j_1 \leq i_1 \wedge 0 \leq j_2 \leq i_2 \wedge i_1 - j_1 < m_1 \wedge i_2 - j_2 < m_2 \wedge \\
& A = T[j_1 \dots i_1][j_2 \dots i_2] \wedge b = T[j_1 \dots i_1][i_2] \wedge c = T[i_1][j_2 \dots i_2] \wedge \\
& \left[ \begin{array}{c|c} A & b \\ \hline c & T[i_1, i_2] \end{array} \right] \in \text{pref}(P) \wedge \left[ \begin{array}{c} A \\ c \end{array} \right] \in \text{pref}(P) \wedge T[i_1, i_2] = P[i_1 - j_1, i_2 - j_2] : \\
& \left[ \begin{array}{c|c} A & b \\ \hline c & T[i_1, i_2] \end{array} \right] \rangle \cup \mathcal{ES}_{m_1, m_2} \\
= & \{ \text{prop. matrices} \} \\
& \langle \text{set } j_1, j_2, A, b, c : 0 \leq j_1 \leq i_1 \wedge 0 \leq j_2 \leq i_2 \wedge i_1 - j_1 < m_1 \wedge i_2 - j_2 < m_2 \wedge \\
& \left[ \begin{array}{c|c} A & b \\ \hline c & T[i_1, i_2] \end{array} \right] = T[j_1 \dots i_1][j_2 \dots i_2 + 1] \wedge \text{row}(b) = \text{row}(A) \wedge \text{col}(b) = 1 \wedge \\
& \left[ \begin{array}{c} A \\ c \end{array} \right] = T[j_1 \dots i_1 + 1][j_2 \dots i_2] \wedge \text{row}(c) = 1 \wedge \text{col}(c) = \text{col}(A) \wedge \\
& \left[ \begin{array}{c|c} A & b \\ \hline c & T[i_1, i_2] \end{array} \right] \in \text{pref}(P) \wedge \left[ \begin{array}{c} A \\ c \end{array} \right] \in \text{pref}(P) \wedge T[i_1, i_2] = P[i_1 - j_1, i_2 - j_2] : \\
& \left[ \begin{array}{c|c} A & b \\ \hline c & T[i_1, i_2] \end{array} \right] \rangle \cup \mathcal{ES}_{m_1, m_2} \\
= & \{ \text{one-point rule: } j_1 = i_1 - \text{row}(A), j_2 = i_2 - \text{col}(A); \text{def. suff} \} \\
& \langle \text{set } A, b, c : \text{row}(A) < m_1 \wedge \text{col}(A) < m_2 \wedge \\
& \text{row}(b) = \text{row}(A) \wedge \text{col}(b) = 1 \wedge \text{row}(c) = 1 \wedge \text{col}(c) = \text{col}(A) \wedge \\
& \left[ \begin{array}{c|c} A & b \\ \hline c & T[i_1, i_2] \end{array} \right] \in \text{suff}(T[0 \dots i_1][0 \dots i_2 + 1]) \wedge \left[ \begin{array}{c} A \\ c \end{array} \right] \in \text{suff}(T[0 \dots i_1 + 1][0 \dots i_2]) \wedge \\
& \left[ \begin{array}{c|c} A & b \\ \hline c & T[i_1, i_2] \end{array} \right] \in \text{pref}(P) \wedge \left[ \begin{array}{c} A \\ c \end{array} \right] \in \text{pref}(P) \wedge T[i_1, i_2] = P[\text{row}(A), \text{col}(A)] : \\
& \left[ \begin{array}{c|c} A & b \\ \hline c & T[i_1, i_2] \end{array} \right] \rangle
\end{aligned}$$

$$\begin{aligned}
& \left[ \frac{A \mid b}{c \mid T[i_1, i_2]} \right] \rangle \cup \mathcal{ES}_{m_1, m_2} \\
= & \{ \text{set calculus} \} \\
& \langle \text{set } A, b, c : \text{row}(A) < m_1 \wedge \text{col}(A) < m_2 \wedge \\
& \text{row}(b) = \text{row}(A) \wedge \text{col}(b) = 1 \wedge \text{row}(c) = 1 \wedge \text{col}(c) = \text{col}(A) \wedge \\
& \left[ \frac{A \mid b}{c} \right] \in \text{suff}(T[0 \dots i_1][0 \dots i_2 + 1]) \cap \text{pref}(P) \wedge \\
& \left[ \frac{A}{c} \right] \in \text{suff}(T[0 \dots i_1 + 1][0 \dots i_2]) \cap \text{pref}(P) \wedge T[i_1, i_2] = P[\text{row}(A), \text{col}(A)] : \\
& \left[ \frac{A \mid b}{c \mid T[i_1, i_2]} \right] \rangle \cup \mathcal{ES}_{m_1, m_2} \\
= & \{ \bullet (?) \} \\
& \langle \text{set } A, b, c : \text{row}(A) < m_1 \wedge \text{col}(A) < m_2 \wedge \\
& \text{row}(b) = \text{row}(A) \wedge \text{col}(b) = 1 \wedge \text{row}(c) = 1 \wedge \text{col}(c) = \text{col}(A) \wedge \\
& \left[ \frac{A \mid b}{c} \right] \in (\text{suff}(T[0 \dots i_1][0 \dots i_2 + 1]) \cap \text{pref}(P)) \cup \mathcal{ES}_{m_1, m_2} \wedge \\
& \left[ \frac{A}{c} \right] \in (\text{suff}(T[0 \dots i_1 + 1][0 \dots i_2]) \cap \text{pref}(P)) \cup \mathcal{ES}_{m_1, m_2} \wedge T[i_1, i_2] = P[\text{row}(A), \text{col}(A)] : \\
& \left[ \frac{A \mid b}{c \mid T[i_1, i_2]} \right] \rangle \cup \mathcal{ES}_{m_1, m_2} \\
= & \{ \text{def. } f \} \\
& \langle \text{set } A, b, c : \text{row}(A) < m_1 \wedge \text{col}(A) < m_2 \wedge \\
& \text{row}(b) = \text{row}(A) \wedge \text{col}(b) = 1 \wedge \text{row}(c) = 1 \wedge \text{col}(c) = \text{col}(A) \wedge \\
& \left[ \frac{A \mid b}{c} \right] \in f(i_1, i_2 + 1) \wedge \left[ \frac{A}{c} \right] \in f(i_1 + 1, i_2) \wedge T[i_1, i_2] = P[\text{row}(A), \text{col}(A)] : \\
& \left[ \frac{A \mid b}{c \mid T[i_1, i_2]} \right] \rangle \cup \mathcal{ES}_{m_1, m_2} \\
= & \{ \bullet \text{ introduction of } \delta \} \\
& \delta(f(i_1, i_2 + 1), f(i_1 + 1, i_2), T[i_1, i_2])
\end{aligned}$$

Ad (?): we still need to justify that we can introduce  $\mathcal{ES}_{m_1, m_2}$  in this step of the derivation. We assume we have  $A$ ,  $b$  and  $c$ , satisfying:

$$\begin{aligned}
& \text{row}(A) < m_1 \wedge \text{col}(A) < m_2 \wedge \\
& \text{row}(b) = \text{row}(A) \wedge \text{col}(b) = 1 \wedge \text{row}(c) = 1 \wedge \text{col}(c) = \text{col}(A) \wedge \\
& \left[ \frac{A \mid b}{c} \right] \in (\text{suff}(T[0 \dots i_1][0 \dots i_2 + 1]) \cap \text{pref}(P)) \cup \mathcal{ES}_{m_1, m_2} \wedge \\
& \left[ \frac{A}{c} \right] \in (\text{suff}(T[0 \dots i_1 + 1][0 \dots i_2]) \cap \text{pref}(P)) \cup \mathcal{ES}_{m_1, m_2}
\end{aligned}$$

We derive:

$$\begin{aligned}
& \left[ \frac{A \mid b}{c} \right] \in \mathcal{ES}_{m_1, m_2} \\
\equiv & \{ \text{definition } \mathcal{ES} \} \\
& \left[ \frac{A \mid b}{c} \right] \in \langle \text{set } j : 0 \leq j \leq m_1 : \mathcal{E}_{j,0} \rangle \cup \langle \text{set } j : 0 \leq j \leq m_2 : \mathcal{E}_{0,j} \rangle \\
\equiv & \{ \text{col}(b) = 1, \text{ therefore } 0 < \text{col}(\left[ \frac{A \mid b}{c} \right]) \} \\
& \left[ \frac{A \mid b}{c} \right] \in \langle \text{set } j : 0 \leq j \leq m_2 : \mathcal{E}_{0,j} \rangle \\
\Rightarrow & \{ \text{set calculus, definition } \mathcal{E} \} \\
& \text{row}(\left[ \frac{A \mid b}{c} \right]) = 0 \wedge 0 < \text{col}(\left[ \frac{A \mid b}{c} \right]) \leq m_2 \\
\equiv & \{ \text{def. row, col; col}(b) = 1 \}
\end{aligned}$$

$$\begin{aligned}
& \text{row}(A) = 0 \wedge 0 \leq \text{col}(A) < m_2 \\
\equiv & \{ \text{def. row, col; row}(c) = 1 \} \\
& \text{row}\left(\begin{bmatrix} A \\ c \end{bmatrix}\right) = 1 \wedge 0 \leq \text{col}\left(\begin{bmatrix} A \\ c \end{bmatrix}\right) < m_2 \\
\equiv & \left\{ \begin{bmatrix} A \\ c \end{bmatrix} \in (\text{suff}(T[0 \dots i_1 + 1][0 \dots i_2])) \cap \text{pref}(P) \cup \mathcal{ES}_{m_1, m_2}; \right. \\
& \left. \text{row}\left(\begin{bmatrix} A \\ c \end{bmatrix}\right) = 1 \wedge \begin{bmatrix} A \\ c \end{bmatrix} \in \mathcal{ES}_{m_1, m_2} \Rightarrow \text{col}\left(\begin{bmatrix} A \\ c \end{bmatrix}\right) = 0 \right\} \\
& \text{row}\left(\begin{bmatrix} A \\ c \end{bmatrix}\right) = 1 \wedge 0 \leq \text{col}\left(\begin{bmatrix} A \\ c \end{bmatrix}\right) < i_2 + 1 \downarrow m_2 \\
\equiv & \{ \text{def. row, col; row}(c) = 1 \} \\
& \text{row}(A) = 0 \wedge 0 \leq \text{col}(A) < i_2 + 1 \downarrow m_2 \\
\equiv & \{ \text{def. row, col; col}(b) = 1 \} \\
& \text{row}([A \mid b]) = 0 \wedge 0 < \text{col}([A \mid b]) \leq i_2 + 1 \downarrow m_2 \\
\equiv & \{ \text{definition } \mathcal{ES} \} \\
& [A \mid b] \in \mathcal{ES}_{0, i_2 + 1 \downarrow m_2} \\
\Rightarrow & \{ \text{theorem C.0} \} \\
& [A \mid b] \in \text{suff}(T[0 \dots i_1][0 \dots i_2 + 1]) \cap \text{pref}(P)
\end{aligned}$$

Symmetrically,  $\begin{bmatrix} A \\ c \end{bmatrix} \in \mathcal{ES}_{m_1, m_2} \Rightarrow \begin{bmatrix} A \\ c \end{bmatrix} \in \text{suff}(T[0 \dots i_1 + 1][0 \dots i_2]) \cap \text{pref}(P)$ . The step marked (?) is therefore correct.

We introduce function  $\delta : \mathcal{P}(\text{pref}(P)) \times \mathcal{P}(\text{pref}(P)) \times \Sigma \rightarrow \mathcal{P}(\text{pref}(P))$ , with the following specification:

$$\begin{aligned}
\delta(Q, R, \sigma) = & \langle \text{set } A, b, c : \text{SIZE}(A, b, c) \wedge [A \mid b] \in Q \wedge \begin{bmatrix} A \\ c \end{bmatrix} \in R \wedge \\
& \sigma = P[\text{row}(A), \text{col}(A)] : \begin{bmatrix} A & b \\ c & \sigma \end{bmatrix} \rangle \cup \mathcal{ES}_{m_1, m_2}
\end{aligned}$$

Here we have used the auxiliary predicate  $\text{SIZE} : \mathcal{M}_2(\Sigma) \times \mathcal{M}_2(\Sigma) \times \mathcal{M}_2(\Sigma) \rightarrow \mathbb{B}$ , which is defined as follows:

$$\begin{aligned}
\text{SIZE}(A, b, c) \equiv & \text{row}(A) < m_1 \wedge \text{col}(A) < m_2 \wedge \\
& \text{row}(b) = \text{row}(A) \wedge \text{col}(b) = 1 \wedge \text{row}(c) = 1 \wedge \text{col}(c) = \text{col}(A)
\end{aligned}$$

The expression  $\text{SIZE}(A, b, c)$  means that matrices  $A$ ,  $b$  and  $c$  have such sizes that, for any additional symbol  $\sigma$ , the matrix  $\begin{bmatrix} A & b \\ c & \sigma \end{bmatrix}$  is well-defined and not larger, in either dimension, than the pattern  $P$ .

## 6.2 Algorithm structure

In summary of what we have seen so far, we have specified function  $f$  and derived the correctness of the following way to compute  $f$ :

$$\begin{aligned} f(0, i_2) &= \mathcal{ES}_{m_1, m_2} && (0 \leq i_2 \leq n_2) \\ f(i_1, 0) &= \mathcal{ES}_{m_1, m_2} && (0 \leq i_1 \leq n_1) \\ f(i_1 + 1, i_2 + 1) &= \delta(f(i_1, i_2 + 1), f(i_1 + 1, i_2), T[i_1, i_2]) && (0 \leq i_1 < n_1, 0 \leq i_2 < n_2) \end{aligned}$$

We are interested in those values of  $i_1$  and  $i_2$ , with  $m_1 \leq i_1 \leq n_1$  and  $m_2 \leq i_2 \leq n_2$ , where the following holds:  $P \in f(i_1, i_2)$ . A solution to the two-dimensional pattern matching problem is to compute all values of  $f(i_1, i_2)$  (for  $0 \leq i_1 \leq n_1, 0 \leq i_2 \leq n_2$ ) and then do a membership test on the relevant values. This gives rise to the following algorithm:

```

O := ∅;
for i1 : 0 ≤ i1 ≤ n1 → f(i1, 0) := ESm1, m2 rof;
for i2 : 0 ≤ i2 ≤ n2 → f(0, i2) := ESm1, m2 rof;
i1 := 0;
do i1 ≠ n1 →
  i2 := 0;
  do i2 ≠ n2 →
    f(i1 + 1, i2 + 1) := δ(f(i1, i2 + 1), f(i1 + 1, i2), T[i1, i2]);
    i2 := i2 + 1
  od;
  i1 := i1 + 1
od;
for i1, i2 : m1 ≤ i1 ≤ n1 ∧ m2 ≤ i2 ≤ n2 →
  if P ∈ f(i1, i2) → O := O ∪ {(i1 - m1, i2 - m2)}
  || P ∉ f(i1, i2) → skip
  fi
rof

```

However, if we assume that the pattern is not an empty matrix<sup>2</sup>, we can report the matches while computing  $f$ . To show this we will further rewrite the postcondition. First we introduce a lemma, stating that there can be no occurrence of  $P$  in a prefix of  $T$  that is too small in either dimension.

**Lemma 6.2** For  $0 \leq i_1 < m_1$  and  $0 \leq i_2 < m_2$ :

$$P \notin f(i_1, i_2)$$

**Proof**

$$\begin{aligned} &P \in f(i_1, i_2) \\ \equiv &\{ \text{definition } f \} \\ &P \in (\text{suff}(T[0 .. i_1][0 .. i_2]) \cap \text{pref}(P)) \cup \mathcal{ES}_{m_1, m_2} \\ \equiv &\{ P \notin \mathcal{ES} \} \end{aligned}$$

---

<sup>2</sup>If  $P$  is an empty matrix, the two-dimensional pattern matching problem is fairly trivial in itself, so (as in previous algorithms) we can safely assume that  $P \notin \mathcal{ES}$ ; we will not consider the case of an empty pattern in the remainder of this section.

$$\begin{aligned}
& P \in \text{suff}(T[0 \dots i_1][0 \dots i_2]) \cap \text{pref}(P) \\
\equiv & \{ P \in \text{pref}(P) \} \\
& P \in \text{suff}(T[0 \dots i_1][0 \dots i_2]) \\
\equiv & \{ i_1 < m_1 \vee i_2 < m_2 \} \\
& \text{false}
\end{aligned}$$

□

Now we can derive:

$$\begin{aligned}
& \langle \text{set } i_1, i_2 : m_1 \leq i_1 \leq n_1 \wedge m_2 \leq i_2 \leq n_2 \wedge P \in f(i_1, i_2) : (i_1 - m_1, i_2 - m_2) \rangle \\
= & \{ P \notin \mathcal{ES}, \text{ therefore: } 1 \leq m_1 \text{ and } 1 \leq m_2; \text{ lemma 6.2} \} \\
& \langle \text{set } i_1, i_2 : 1 \leq i_1 \leq n_1 \wedge 1 \leq i_2 \leq n_2 \wedge P \in f(i_1, i_2) : (i_1 - m_1, i_2 - m_2) \rangle
\end{aligned}$$

This gives rise to the following algorithm:

```

O := ∅;
for i1 : 0 ≤ i1 ≤ n1 → f(i1, 0) := ESm1, m2 rof;
for i2 : 0 ≤ i2 ≤ n2 → f(0, i2) := ESm1, m2 rof;
i1 := 0;
do i1 ≠ n1 →
  i2 := 0;
  do i2 ≠ n2 →
    f(i1 + 1, i2 + 1) := δ(f(i1, i2 + 1), f(i1 + 1, i2), T[i1, i2]);
    if P ∈ f(i1 + 1, i2 + 1) → O := O ∪ {(i1 + 1 - m1, i2 + 1 - m2)}
    || P ∉ f(i1 + 1, i2 + 1) → skip
    fi;
    i2 := i2 + 1
  od;
  i1 := i1 + 1
od

```

Here we can introduce a space improvement, similarly to what we have done for the filter-based algorithms in section 4.2. Instead of storing all  $(n_1+1)*(n_2+1)$  values of  $f$ , an array of length  $n_2+1$  suffices. Let us call this array  $e$  and introduce the following invariant:

$$\langle \forall j : 0 \leq j \leq i_2 : e[j] = f(i_1 + 1, j) \rangle \wedge \langle \forall j : i_2 < j \leq n_2 : e[j] = f(i_1, j) \rangle$$

This gives us the following algorithm:

```

O := ∅;
for j : 0 ≤ j ≤ n2 → e[j] := ESm1, m2 rof;
i1, i2 := 0, 0;
do i1 ≠ n1 →
  do i2 ≠ n2 →
    e[i2 + 1] := δ(e[i2 + 1], e[i2], T[i1, i2]);
    if P ∈ e[i2 + 1] → O := O ∪ {(i1 + 1 - m1, i2 + 1 - m2)}
    || P ∉ e[i2 + 1] → skip
  od
  i1 := i1 + 1
od

```

```

    fi;
    i2 := i2 + 1
  od;
  i2 := 0;
  i1 := i1 + 1
od

```

## 6.3 Precomputation

### 6.3.0 Representing sets of matrices by lists of maximal elements

The algorithm given in section 6.2 is correct, if we can find a way to compute  $\delta$ . However, computation of  $\delta(Q, R, \sigma)$  can be rather inefficient, so repeating this for each element of the, potentially very large, text is undesirable. We would prefer precomputing the values of  $\delta$ . We do not need to precompute  $\delta(Q, R, \sigma)$  for *all*  $Q, R \subseteq \text{pref}(P)$ , since not all such sets  $Q$  and  $R$  can actually occur. We only need to examine *reachable* sets of prefixes of  $P$ .

**Definition 6.3 (Reachability)** *The set of all reachable sets of prefixes of  $P$  is the smallest set  $V$  satisfying:*

- $\mathcal{ES}_{m_1, m_2} \in V$ ;
- if  $Q, R \in V$  and  $\sigma \in \Sigma$  then  $\delta(Q, R, \sigma) \in V$ .

We will now prove two properties of reachable sets of prefixes of  $P$ .

**Theorem 6.4** *Let  $S$  be a reachable set of prefixes of  $P$ . Then we have:*

$$\langle \forall A, B : A \in S \wedge B \in \text{suff}(A) \cap \text{pref}(P) : B \in S \rangle$$

**Proof** We prove this by structural induction.

- $S = \mathcal{ES}_{m_1, m_2}$ .

$$\begin{aligned}
& A \in \mathcal{ES}_{m_1, m_2} \wedge B \in \text{suff}(A) \cap \text{pref}(P) \\
\Rightarrow & \{ \text{def. } \mathcal{ES}, \text{ def. suff} \} \\
& ((\text{row}(A) = 0 \wedge \text{col}(A) \leq m_2) \vee (\text{row}(A) \leq m_1 \wedge \text{col}(A) = 0)) \wedge \\
& \quad \text{row}(B) \leq \text{row}(A) \wedge \text{col}(B) \leq \text{col}(A) \\
\Rightarrow & \{ \text{pred. calc, math} \} \\
& (\text{row}(B) = 0 \wedge \text{col}(B) \leq m_2) \vee (\text{row}(B) \leq m_1 \wedge \text{col}(B) = 0) \\
\equiv & \{ \text{def. } \mathcal{ES} \} \\
& B \in \mathcal{ES}_{m_1, m_2}
\end{aligned}$$

- $S = \delta(Q, R, \sigma)$ , for some reachable  $Q, R \subseteq \text{pref}(P)$  and  $\sigma \in \Sigma$ . Our induction hypothesis is:

$$\begin{aligned}
& \langle \forall A, B : A \in Q \wedge B \in \text{suff}(A) \cap \text{pref}(P) : B \in Q \rangle \wedge \\
& \langle \forall A, B : A \in R \wedge B \in \text{suff}(A) \cap \text{pref}(P) : B \in R \rangle
\end{aligned}$$

We must prove:

$$\langle \forall A, B : A \in \delta(Q, R, \sigma) \wedge B \in \text{suff}(A) \cap \text{pref}(P) : B \in \delta(Q, R, \sigma) \rangle$$

We assume  $A \in \delta(Q, R, \sigma) \wedge B \in \text{suff}(A) \cap \text{pref}(P)$  and prove  $B \in \delta(Q, R, \sigma)$ .

- $B \in \mathcal{E}S_{m_1, m_2}$ . Then we have, by definition of  $\delta$ :  $B \in \delta(Q, R, \sigma)$ .
- $B \notin \mathcal{E}S_{m_1, m_2}$ . This means that  $B$  is not an empty matrix and can therefore be written as  $\left[ \begin{array}{c|c} B' & b_0 \\ \hline b_1 & \tau \end{array} \right]$ . From  $B \in \text{suff}(A)$  we know that  $A$  is not an empty matrix either and

can be written as  $\left[ \begin{array}{c|c} A' & a_0 \\ \hline a_1 & \tau \end{array} \right]$ . We derive:

$$\begin{aligned} & \left[ \begin{array}{c|c} A' & a_0 \\ \hline a_1 & \tau \end{array} \right] \in \delta(Q, R, \sigma) \wedge \left[ \begin{array}{c|c} B' & b_0 \\ \hline b_1 & \tau \end{array} \right] \in \text{suff}\left(\left[ \begin{array}{c|c} A' & a_0 \\ \hline a_1 & \tau \end{array} \right]\right) \cap \text{pref}(P) \\ \Rightarrow & \{ \text{def. } \delta, \left[ \begin{array}{c|c} A' & a_0 \\ \hline a_1 & \tau \end{array} \right] \notin \mathcal{E}S_{m_1, m_2}; \text{property } \cap \} \\ & \left[ A' \mid a_0 \right] \in Q \wedge \left[ \begin{array}{c} A' \\ \hline a_1 \end{array} \right] \in R \wedge \sigma = \tau \wedge \left[ \begin{array}{c|c} B' & b_0 \\ \hline b_1 & \tau \end{array} \right] \in \text{suff}\left(\left[ \begin{array}{c|c} A' & a_0 \\ \hline a_1 & \tau \end{array} \right]\right) \wedge \\ & \left[ \begin{array}{c|c} B' & b_0 \\ \hline b_1 & \tau \end{array} \right] \in \text{pref}(P) \\ \equiv & \{ \text{property pref} \} \\ & \left[ A' \mid a_0 \right] \in Q \wedge \left[ \begin{array}{c} A' \\ \hline a_1 \end{array} \right] \in R \wedge \sigma = \tau \wedge \left[ \begin{array}{c|c} B' & b_0 \\ \hline b_1 & \tau \end{array} \right] \in \text{suff}\left(\left[ \begin{array}{c|c} A' & a_0 \\ \hline a_1 & \tau \end{array} \right]\right) \wedge \\ & \left[ B' \mid b_0 \right] \in \text{pref}(P) \wedge \left[ \begin{array}{c} B' \\ \hline b_1 \end{array} \right] \in \text{pref}(P) \wedge P[\text{row}(B'), \text{col}(B')] = \tau \\ \equiv & \{ \text{property suff} \} \\ & \left[ A' \mid a_0 \right] \in Q \wedge \left[ \begin{array}{c} A' \\ \hline a_1 \end{array} \right] \in R \wedge \sigma = \tau \wedge \\ & \left[ B' \mid b_0 \right] \in \text{suff}\left(\left[ A' \mid a_0 \right]\right) \wedge \left[ \begin{array}{c} B' \\ \hline b_1 \end{array} \right] \in \text{suff}\left(\left[ \begin{array}{c} A' \\ \hline a_1 \end{array} \right]\right) \wedge \\ & \left[ B' \mid b_0 \right] \in \text{pref}(P) \wedge \left[ \begin{array}{c} B' \\ \hline b_1 \end{array} \right] \in \text{pref}(P) \wedge P[\text{row}(B'), \text{col}(B')] = \tau \\ \equiv & \{ \text{property } \cap \} \\ & \left[ A' \mid a_0 \right] \in Q \wedge \left[ B' \mid b_0 \right] \in \text{suff}\left(\left[ A' \mid a_0 \right]\right) \cap \text{pref}(P) \wedge \\ & \left[ \begin{array}{c} A' \\ \hline a_1 \end{array} \right] \in R \wedge \left[ \begin{array}{c} B' \\ \hline b_1 \end{array} \right] \in \text{suff}\left(\left[ \begin{array}{c} A' \\ \hline a_1 \end{array} \right]\right) \cap \text{pref}(P) \wedge P[\text{row}(B'), \text{col}(B')] = \tau \wedge \\ & \sigma = \tau \\ \Rightarrow & \{ \text{induction hypothesis} \} \\ & \left[ B' \mid b_0 \right] \in Q \wedge \left[ \begin{array}{c} B' \\ \hline b_1 \end{array} \right] \in R \wedge P[\text{row}(B'), \text{col}(B')] = \tau \wedge \sigma = \tau \\ \Rightarrow & \{ \text{def. } \delta \} \\ & \left[ \begin{array}{c|c} B' & b_0 \\ \hline b_1 & \tau \end{array} \right] \in \delta(Q, R, \tau) \wedge \sigma = \tau \\ \Rightarrow & \{ \text{substitution} \} \\ & \left[ \begin{array}{c|c} B' & b_0 \\ \hline b_1 & \tau \end{array} \right] \in \delta(Q, R, \sigma) \end{aligned}$$

□

In other words, theorem 6.4 tells us that every reachable set  $S \subseteq \text{pref}(P)$  is a so-called *ideal* of the partial order  $(\text{pref}(P), \leq_s)$  (where  $A \leq_s B \equiv A \in \text{suff}(B)$ ).

**Theorem 6.5** *Let  $S$  be a reachable set of prefixes of  $P$ . Then we have:*

$$\langle \forall A, B : A, B \in S \wedge A \in \text{pref}(B) : A \in \text{suff}(B) \rangle$$

**Proof** Again we use structural induction.

- $S = \mathcal{E}S_{m_1, m_2}$ . In this case  $A$  is an empty matrix. The definitions of  $\text{pref}$  and  $\text{suff}$  then give us  $A \in \text{pref}(B) \equiv A \in \text{suff}(B)$ .
- $S = \delta(Q, R, \sigma)$ , for some reachable  $Q, R \subseteq \text{pref}(P)$  and  $\sigma \in \Sigma$ . Our induction hypothesis is:

$$\begin{aligned} & \langle \forall A, B : A, B \in Q \wedge A \in \text{pref}(B) : A \in \text{suff}(B) \rangle \quad \wedge \\ & \langle \forall A, B : A, B \in R \wedge A \in \text{pref}(B) : A \in \text{suff}(B) \rangle \end{aligned}$$

Then we should still prove:

$$\langle \forall A, B : A, B \in \delta(Q, R, \sigma) \wedge A \in \text{pref}(B) : A \in \text{suff}(B) \rangle$$

We use the same proof structure as in theorem 6.4: we assume we have  $A, B \in \delta(Q, R, \sigma)$ , with  $A \in \text{pref}(B)$ , and try to prove  $A \in \text{suff}(B)$ . Again we distinguish two cases.

- $A \in \mathcal{E}S_{m_1, m_2}$ . In this case the theorem holds trivially, again because the definitions of  $\text{pref}$  and  $\text{suff}$  tell us:  $A \in \text{pref}(B) \equiv A \in \text{suff}(B)$ .
- $A \notin \mathcal{E}S_{m_1, m_2}$ . As in theorem 6.4, we can then write the following:

$$\begin{aligned} A &= \left[ \begin{array}{c|c} A' & a_0 \\ \hline a_1 & \tau \end{array} \right] \\ B &= \left[ \begin{array}{c|c} B' & b_0 \\ \hline b_1 & \tau \end{array} \right] \end{aligned}$$

We get the following derivation:

$$\begin{aligned} & \left[ \begin{array}{c|c} A' & a_0 \\ \hline a_1 & \tau \end{array} \right] \in \delta(Q, R, \sigma) \wedge \left[ \begin{array}{c|c} B' & b_0 \\ \hline b_1 & \tau \end{array} \right] \in \delta(Q, R, \sigma) \wedge \\ & \left[ \begin{array}{c|c} A' & a_0 \\ \hline a_1 & \tau \end{array} \right] \in \text{pref}\left(\left[ \begin{array}{c|c} B' & b_0 \\ \hline b_1 & \tau \end{array} \right]\right) \\ \Rightarrow & \{ \text{property pref} \} \\ & \left[ \begin{array}{c|c} A' & a_0 \\ \hline a_1 & \tau \end{array} \right] \in \delta(Q, R, \sigma) \wedge \left[ \begin{array}{c|c} B' & b_0 \\ \hline b_1 & \tau \end{array} \right] \in \delta(Q, R, \sigma) \wedge \\ & [ A' \mid a_0 ] \in \text{pref}([ B' \mid b_0 ]) \wedge \left[ \begin{array}{c} A' \\ \hline a_1 \end{array} \right] \in \text{pref}\left(\left[ \begin{array}{c} B' \\ \hline b_1 \end{array} \right]\right) \\ \Rightarrow & \{ \text{def. } \delta \} \\ & [ A' \mid a_0 ] \in Q \wedge \left[ \begin{array}{c} A' \\ \hline a_1 \end{array} \right] \in R \wedge [ B' \mid b_0 ] \in Q \wedge \left[ \begin{array}{c} B' \\ \hline b_1 \end{array} \right] \in R \wedge \\ & [ A' \mid a_0 ] \in \text{pref}([ B' \mid b_0 ]) \wedge \left[ \begin{array}{c} A' \\ \hline a_1 \end{array} \right] \in \text{pref}\left(\left[ \begin{array}{c} B' \\ \hline b_1 \end{array} \right]\right) \\ \Rightarrow & \{ \text{induction hypothesis} \} \\ & [ A' \mid a_0 ] \in \text{suff}([ B' \mid b_0 ]) \wedge \left[ \begin{array}{c} A' \\ \hline a_1 \end{array} \right] \in \text{suff}\left(\left[ \begin{array}{c} B' \\ \hline b_1 \end{array} \right]\right) \end{aligned}$$

$$\equiv \{ \text{property suff} \}$$

$$\left[ \frac{A'}{a_1} \mid \frac{a_0}{\tau} \right] \in \text{suff} \left( \left[ \frac{B'}{b_1} \mid \frac{b_0}{\tau} \right] \right)$$

□

These properties suggest that a reachable set of prefixes of  $P$  can be represented by its maximal elements.

**Definition 6.6 (Maximal element)** *A maximal element of a reachable set  $S \subseteq \text{pref}(P)$  is a matrix  $A \in S$  for which the following holds:*

$$\neg(\exists B : B \in S \wedge B \neq A : A \in \text{suff}(B))$$

Note that it is possible (and likely) for a set to have more than one maximal element, since  $\leq_s$  is not a total order.

For  $A, B$  in some reachable set of prefixes of  $P$ , we derive:

$$\begin{aligned} & A \in \text{suff}(B) \\ \Leftarrow & \{ \text{theorem 6.5} \} \\ & A \in \text{pref}(B) \\ \equiv & \{ A, B \in \text{pref}(P) \} \\ & \text{row}(A) \leq \text{row}(B) \wedge \text{col}(A) \leq \text{col}(B) \end{aligned}$$

Since the definition of  $\text{suff}$  also tells us:  $A \in \text{suff}(B) \Rightarrow \text{row}(A) \leq \text{row}(B) \wedge \text{col}(A) \leq \text{col}(B)$ , we can now conclude that for  $A, B$  in some reachable set  $S$ , we have:

$$A \in \text{suff}(B) \equiv \text{row}(A) \leq \text{row}(B) \wedge \text{col}(A) \leq \text{col}(B)$$

So we note that for each pair of different maximal elements  $A, B$ :

$$(\text{row}(A) < \text{row}(B) \wedge \text{col}(B) < \text{col}(A)) \vee (\text{row}(B) < \text{row}(A) \wedge \text{col}(A) < \text{col}(B))$$

We want to represent a reachable set by a list of its maximal elements, ordered by increasing number of columns and decreasing number of rows. We can then express  $\delta$  in terms of these lists of only maximal elements, instead of entire sets of matrices. The advantage of this is that a set of prefixes of  $P$  can have  $(m_1 + 1) * (m_2 + 1)$  elements, whereas such a list has at most  $m_1 \downarrow m_2$  elements (which follows from the way it is ordered). Also, we can (hopefully) use knowledge about the list to come up with an efficient way to compute  $\delta$ .

Besides the improvements in space and time complexity of the precomputation step, there is another good reason to examine this approach, where sets of matrices are represented by a list of their maximal elements. The Polcar algorithm is essentially rather similar to the Aho-Corasick and Knuth-Morris-Pratt algorithms ([AC75, KMP77]). As seen in [WZ92, WZ93], there too we reason in terms of strings of maximal length, as opposed to sets of strings. The main difference is that in the one-dimensional case, there is one unique string of maximum length, for each nonempty set of prefixes of the pattern. But as we have seen, there can be several different maximal elements in a set of prefixes of our two-dimensional pattern.

For an overview of the notation we use for lists and list operations, we refer to appendix D.

We introduce abstraction function  $q : \mathcal{L}(\mathcal{M}_2(\Sigma)) \rightarrow \mathcal{P}(\mathcal{M}_2(\Sigma))$ , with the following recursive definition:

$$\begin{aligned} q([]) &= \emptyset \\ q(A \triangleright As) &= (\text{suff}(A) \cap \text{pref}(P)) \cup q(As) \end{aligned} \quad (7)$$

It is easy to see that the following properties hold for function  $q$  (proof omitted):

$$q(As \triangleleft A) = q(As) \cup (\text{suff}(A) \cap \text{pref}(P)) \quad (8)$$

$$q(As \uparrow\uparrow Bs) = q(As) \cup q(Bs) \quad (9)$$

Next we introduce the predicate  $OL$ , to express that a list is ordered.

$$\begin{aligned} OL([]) &\equiv \text{true} \\ OL([A]) &\equiv \text{true} \\ OL(A_0 \triangleright A_1 \triangleright As) &\equiv \text{row}(A_1) < \text{row}(A_0) \wedge \text{col}(A_0) < \text{col}(A_1) \wedge OL(A_1 \triangleright As) \end{aligned} \quad (10)$$

Again, we have several useful properties of  $OL$  (and again, we omit their proof):

$$\begin{aligned} OL(As \triangleleft A_0 \triangleleft A_1) &\equiv OL(As \triangleleft A_0) \wedge \text{row}(A_1) < \text{row}(A_0) \wedge \\ &\quad \text{col}(A_0) < \text{col}(A_1) \end{aligned} \quad (11)$$

$$\begin{aligned} OL((As \triangleleft A) \uparrow\uparrow (B \triangleright Bs)) &\equiv OL(As \triangleleft A) \wedge \text{row}(B) < \text{row}(A) \wedge \\ &\quad \text{col}(A) < \text{col}(B) \wedge OL(B \triangleright Bs) \end{aligned} \quad (12)$$

So for each reachable set  $S \subseteq \text{pref}(P)$  we want to find a list  $As$ , with  $q(As) = S$  and  $OL(As)$ .

Using our assumption that  $P$  is not an empty matrix, the list corresponding to  $\mathcal{E}S_{m_1, m_2}$  is  $[\mathcal{E}_{m_1, 0}, \mathcal{E}_{0, m_2}]$ :

$$\begin{aligned} &q([\mathcal{E}_{m_1, 0}, \mathcal{E}_{0, m_2}]) \\ = &\quad \{ \text{def. } q \} \\ &(\text{suff}(\mathcal{E}_{m_1, 0}) \cap \text{pref}(P)) \cup (\text{suff}(\mathcal{E}_{0, m_2}) \cap \text{pref}(P)) \\ = &\quad \{ \text{def. suff, def. pref, def. } \mathcal{E}S, \text{row}(P) = m_1 \wedge \text{col}(P) = m_2 \} \\ &\mathcal{E}S_{m_1, m_2} \end{aligned}$$

$$\begin{aligned} &OL([\mathcal{E}_{m_1, 0}, \mathcal{E}_{0, m_2}]) \\ \equiv &\quad \{ \text{def. } OL \} \\ &0 < m_1 \wedge 0 < m_2 \\ \equiv &\quad \{ P \notin \mathcal{E}S_{m_1, m_2} \} \\ &\text{true} \end{aligned}$$

Note that if the pattern  $P$  were an empty matrix, then the list  $[P]$  would suffice.

### 6.3.1 Precomputation algorithm structure

Now what remains is to find an algorithm to compute a list corresponding to  $\delta(Q, R, \sigma)$ , given the lists corresponding to  $Q$  and  $R$  and the symbol  $\sigma$ . More formally, we initially have two lists  $Bs, Cs \in \mathcal{L}(\text{pref}(P))$  with:

$$\begin{aligned} q(Bs) &= Q \\ OL(Bs) & \\ q(Cs) &= R \\ OL(Cs) & \end{aligned}$$

We now want to compute a list  $Ds \in \mathcal{L}(\text{pref}(P))$ , satisfying:

$$\begin{aligned} q(Ds) &= \delta(Q, R, \sigma) \\ OL(Ds) & \end{aligned}$$

We introduce the following invariants:

$$\begin{aligned} P0 : \quad & q(Ds) \cup \delta(q(Bs), q(Cs), \sigma) = \delta(Q, R, \sigma) \\ P1 : \quad & OL(Bs) \\ P2 : \quad & OL(Cs) \\ P3 : \quad & OL(Ds) \end{aligned}$$

Initially, we can choose  $Ds = []$ . Invariants  $P1$ ,  $P2$  and  $P3$  then all hold trivially; ad  $P0$ :

$$\begin{aligned} & q([]) \cup \delta(q(Bs), q(Cs), \sigma) \\ = & \quad \{ \text{def. } q \} \\ & \delta(q(Bs), q(Cs), \sigma) \\ = & \quad \{ q(Bs) = Q, q(Cs) = R \} \\ & \delta(Q, R, \sigma) \end{aligned}$$

We can terminate the computation when  $Bs = [] \vee Cs = []$ :

$$\begin{aligned} & q(Ds) \cup \delta(q([], q(Cs), \sigma) \\ = & \quad \{ (7) \text{ (def. } q) \} \\ & q(Ds) \cup \delta(\emptyset, q(Cs), \sigma) \\ = & \quad \{ \text{def. } \delta \} \\ & q(Ds) \cup \mathcal{ES}_{m_1, m_2} \end{aligned}$$

$$\begin{aligned} & q(Ds) \cup \delta(q(Bs), q([], \sigma) \\ = & \quad \{ \text{def. } q \} \\ & q(Ds) \cup \delta(q(Bs), \emptyset, \sigma) \\ = & \quad \{ \text{def. } \delta \} \\ & q(Ds) \cup \mathcal{ES}_{m_1, m_2} \end{aligned}$$

So after termination of the algorithm we have:  $q(Ds) \cup \mathcal{ES}_{m_1, m_2} = \delta(Q, R, \sigma)$ . Then we still need to ensure that  $Ds$  also represents  $\mathcal{ES}_{m_1, m_2}$ . There are two possible cases:  $Ds = []$  and  $Ds \neq []$ .

- If  $Ds = []$  we can obviously conclude:  $\mathcal{ES}_{m_1, m_2} \not\subseteq q(Ds)$ . We have already seen that in that case (since we assumed that  $P$  is not an empty matrix),  $\mathcal{ES}_{m_1, m_2}$  can be represented by the list  $[\mathcal{E}_{m_1, 0}, \mathcal{E}_{0, m_2}]$ , so we can then simply write  $Ds := [\mathcal{E}_{m_1, 0}, \mathcal{E}_{0, m_2}]$ .
- If  $Ds \neq []$ , then we can write  $Ds$  as  $D' \triangleright Ds'$ , but also as  $Ds'' \triangleleft D''$ . We make the following note:

$$\begin{aligned}
& \mathcal{ES}_{m_1, m_2} \subseteq q(Ds) \\
\equiv & \quad \{ \text{def. } \mathcal{ES}, \text{ property } \subseteq \} \\
& \mathcal{ES}_{m_1, 0} \subseteq q(Ds) \wedge \mathcal{ES}_{0, m_2} \subseteq q(Ds) \\
& \mathcal{ES}_{m_1, 0} \subseteq q(D' \triangleright Ds') \\
\equiv & \quad \{ (7) \text{ (def. } q), \text{ property } \cup, \cap \} \\
& \mathcal{ES}_{m_1, 0} \subseteq (\text{suff}(D') \cap \text{pref}(P)) \cup q(Ds') \\
\Leftarrow & \quad \{ \text{set calculus} \} \\
& \mathcal{ES}_{m_1, 0} \subseteq \text{suff}(D') \wedge \mathcal{ES}_{m_1, 0} \subseteq \text{pref}(P) \\
\equiv & \quad \{ \text{def. } \text{pref}, \text{ row}(P) = m_1 \} \\
& \mathcal{ES}_{m_1, 0} \subseteq \text{suff}(D') \\
\equiv & \quad \{ \text{def. } \mathcal{ES}, \text{ def. } \text{suff} \} \\
& m_1 \leq \text{row}(D')
\end{aligned}$$

If this does not hold, we need to replace  $Ds$  by  $\mathcal{E}_{m_1, 0} \triangleright Ds$ :

$$\begin{aligned}
& \mathcal{ES}_{m_1, 0} \\
= & \quad \{ \text{def. } \mathcal{ES}, \text{ def. } \text{suff} \} \\
& \text{suff}(\mathcal{E}_{m_1, 0}) \\
= & \quad \{ \text{row}(P) = m_1 \} \\
& \text{suff}(\mathcal{E}_{m_1, 0}) \cap \text{pref}(P) \\
\subseteq & \quad \{ \text{set calculus} \} \\
& (\text{suff}(\mathcal{E}_{m_1, 0}) \cap \text{pref}(P)) \cup q(D' \triangleright Ds') \\
= & \quad \{ \text{def. } q \} \\
& q(\mathcal{E}_{m_1, 0} \triangleright D' \triangleright Ds') \\
& OL(\mathcal{E}_{m_1, 0} \triangleright D' \triangleright Ds') \\
\equiv & \quad \{ (10) \text{ (def. } OL) \} \\
& \text{row}(D') < m_1 \wedge 0 < \text{col}(D') \wedge OL(D' \triangleright Ds') \\
\equiv & \quad \{ \text{row}(D') < m_1, OL(D' \triangleright Ds') \} \\
& 0 < \text{col}(D')
\end{aligned}$$

We still need to be sure that  $0 < \text{col}(D')$  holds. In the remainder of the derivation of our algorithm we will see that this is not a problem, since we will only add nonempty matrices to  $Ds$ .

The observation for  $\mathcal{E}S_{0,m_2} \subseteq q(Ds'' \triangleleft D'')$  is completely symmetrical; instead of the definitions of  $q$  and  $OL$  we can use properties (8) and (11). We conclude that if  $\text{col}(D'') < m_2$ , we need to replace  $Ds$  by  $Ds \triangleleft \mathcal{E}_{0,m_2}$ .

So in conclusion, after termination we need to add the following program fragment:

```

if  $Ds = [] \rightarrow Ds := [\mathcal{E}_{m_1,0}, \mathcal{E}_{0,m_2}]$ 
 $\parallel Ds :: D' \triangleright Ds' \wedge Ds :: Ds'' \triangleleft D'' \rightarrow$ 
  if  $\text{row}(D') < m_1 \rightarrow Ds := \mathcal{E}_{m_1,0} \triangleright Ds$ 
   $\parallel m_1 \leq \text{row}(D') \rightarrow \text{skip}$ 
  fi;
  if  $\text{col}(D'') < m_2 \rightarrow Ds := Ds \triangleleft \mathcal{E}_{0,m_2}$ 
   $\parallel m_2 \leq \text{col}(D'') \rightarrow \text{skip}$ 
  fi
fi

```

So we have seen initialisation and termination of our algorithm; what remains is the update of  $Bs$ ,  $Cs$  and  $Ds$ . As we have seen, the algorithm terminates when  $Bs$  or  $Cs$  is empty, so we can assume  $Bs = B' \triangleright Bs'$  and  $Cs = C' \triangleright Cs'$ , for some  $B', C' \in \mathcal{M}_2(\Sigma)$  and  $Bs', Cs' \in \mathcal{L}(\text{pref}(P))$ . We will examine the following cases:

0.  $\text{row}(B') + 1 = \text{row}(C') \wedge \text{col}(B') = \text{col}(C') + 1 \wedge P[\text{row}(B'), \text{col}(C')] = \sigma$ ;
1.  $\text{row}(C') < \text{row}(B') + 1$ ;
2.  $\text{row}(B') + 1 < \text{row}(C')$ ;
3.  $\text{row}(B') + 1 = \text{row}(C') \wedge \text{col}(C') + 1 < \text{col}(B')$ ;
4.  $\text{row}(B') + 1 = \text{row}(C') \wedge \text{col}(B') < \text{col}(C') + 1$ ;
5.  $\text{row}(B') + 1 = \text{row}(C') \wedge \text{col}(B') = \text{col}(C') + 1 \wedge P[\text{row}(B'), \text{col}(C')] \neq \sigma$ .

### 6.3.2 Case analysis

#### Case 0

The first case is:  $\text{row}(B') + 1 = \text{row}(C') \wedge \text{col}(B') = \text{col}(C') + 1 \wedge P[\text{row}(B'), \text{col}(C')] = \sigma$ . Informally, this is the case where a combination of  $B', C'$  and  $\sigma$  actually occurs as a prefix of  $P$ .

First we examine the left-hand side of invariant  $P0$ :

$$\begin{aligned}
& q(Ds) \cup \delta(q(B' \triangleright Bs'), q(C' \triangleright Cs'), \sigma) \\
= & \{ \text{def. } \delta \} \\
& q(Ds) \cup \langle \text{set } A, b, c : \text{SIZE}(A, b, c) \wedge [A \mid b] \in q(B' \triangleright Bs') \wedge \\
& \left[ \begin{array}{c} A \\ c \end{array} \right] \in q(C' \triangleright Cs') \wedge \sigma = P[\text{row}(A), \text{col}(A)] : \left[ \begin{array}{c|c} A & b \\ c & \sigma \end{array} \right] \rangle \cup \mathcal{E}S_{m_1, m_2} \\
= & \{ (7) \text{ (def. } q) \} \\
& q(Ds) \cup \langle \text{set } A, b, c : \text{SIZE}(A, b, c) \wedge [A \mid b] \in (\text{suff}(B') \cap \text{pref}(P)) \cup q(Bs') \wedge \\
& \left[ \begin{array}{c} A \\ c \end{array} \right] \in (\text{suff}(C') \cap \text{pref}(P)) \cup q(Cs') \wedge \sigma = P[\text{row}(A), \text{col}(A)] : \left[ \begin{array}{c|c} A & b \\ c & \sigma \end{array} \right] \rangle \cup
\end{aligned}$$

$$\begin{aligned}
& \mathcal{E}S_{m_1, m_2} \\
= & \{ \text{set calculus} \} \\
& q(Ds) \cup \langle \text{set } A, b, c : \text{SIZE}(A, b, c) \wedge [A \mid b] \in \text{suff}(B') \cap \text{pref}(P) \wedge \\
& \left[ \frac{A}{c} \right] \in (\text{suff}(C') \cap \text{pref}(P)) \cup q(Cs') \wedge \sigma = P[\text{row}(A), \text{col}(A)] : \left[ \frac{A \mid b}{c \mid \sigma} \right] \rangle \cup \\
& \langle \text{set } A, b, c : \text{SIZE}(A, b, c) \wedge [A \mid b] \in (\text{suff}(B') \cap \text{pref}(P)) \cup q(Bs') \wedge \\
& \left[ \frac{A}{c} \right] \in \text{suff}(C') \cap \text{pref}(P) \wedge \sigma = P[\text{row}(A), \text{col}(A)] : \left[ \frac{A \mid b}{c \mid \sigma} \right] \rangle \cup \\
& \langle \text{set } A, b, c : \text{SIZE}(A, b, c) \wedge \\
& [A \mid b] \in q(Bs') \wedge \left[ \frac{A}{c} \right] \in q(Cs') \wedge \sigma = P[\text{row}(A), \text{col}(A)] : \\
& \left[ \frac{A \mid b}{c \mid \sigma} \right] \rangle \cup \mathcal{E}S_{m_1, m_2} \\
= & \{ \text{def. } \delta \} \\
& q(Ds) \cup \langle \text{set } A, b, c : \text{SIZE}(A, b, c) \wedge [A \mid b] \in \text{suff}(B') \cap \text{pref}(P) \wedge \\
& \left[ \frac{A}{c} \right] \in (\text{suff}(C') \cap \text{pref}(P)) \cup q(Cs') \wedge \sigma = P[\text{row}(A), \text{col}(A)] : \left[ \frac{A \mid b}{c \mid \sigma} \right] \rangle \cup \\
& \langle \text{set } A, b, c : \text{SIZE}(A, b, c) \wedge [A \mid b] \in (\text{suff}(B') \cap \text{pref}(P)) \cup q(Bs') \wedge \\
& \left[ \frac{A}{c} \right] \in \text{suff}(C') \cap \text{pref}(P) \wedge \sigma = P[\text{row}(A), \text{col}(A)] : \left[ \frac{A \mid b}{c \mid \sigma} \right] \rangle \cup \\
& \delta(q(Bs'), q(Cs'), \sigma)
\end{aligned}$$

To further simplify this complex expression, we make the following observation, for matrices  $A$ ,  $b$  and  $c$  with  $\text{SIZE}(A, b, c)$  and  $\left[ \frac{A}{c} \right] \in R$ :

$$\begin{aligned}
& [A \mid b] \in \text{suff}(B') \cap \text{pref}(P) \\
\Rightarrow & \{ \text{definition suff} \} \\
& \text{row}([A \mid b]) \leq \text{row}(B') \wedge \text{col}([A \mid b]) \leq \text{col}(B') \\
\equiv & \{ \text{definition row, col} \} \\
& \text{row}(A) \leq \text{row}(B') \wedge \text{col}(A) < \text{col}(B') \\
\equiv & \{ \text{row}(B') + 1 = \text{row}(C') \wedge \text{col}(B') = \text{col}(C') + 1 \} \\
& \text{row}(A) < \text{row}(C') \wedge \text{col}(A) \leq \text{col}(C') \\
\equiv & \{ \text{def. row, col} \} \\
& \text{row}\left(\left[\frac{A}{c}\right]\right) \leq \text{row}(C') \wedge \text{col}\left(\left[\frac{A}{c}\right]\right) \leq \text{col}(C') \\
\equiv & \left\{ \left[\frac{A}{c}\right], C' \in R, \text{ therefore } \left[\frac{A}{c}\right], C' \in \text{pref}(P) \right\} \\
& \left[\frac{A}{c}\right] \in \text{pref}(C') \\
\Rightarrow & \{ \text{theorem 6.5; } \left[\frac{A}{c}\right], C' \in R; R \text{ is a reachable set} \} \\
& \left[\frac{A}{c}\right] \in \text{suff}(C') \\
\equiv & \left\{ \left[\frac{A}{c}\right] \in \text{pref}(P) \right\}
\end{aligned}$$

$$\left[ \frac{A}{c} \right] \in \text{suff}(C') \cap \text{pref}(P)$$

And symmetrically, with  $[ A \mid b ] \in Q$ :

$$\begin{aligned} & \left[ \frac{A}{c} \right] \in \text{suff}(C') \cap \text{pref}(P) \\ \Rightarrow & \{ \text{def. suff, row, col} \} \\ & \text{row}(A) < \text{row}(C') \wedge \text{col}(A) \leq \text{col}(C') \\ \equiv & \{ \text{row}(B') + 1 = \text{row}(C') \wedge \text{col}(B') = \text{col}(C') + 1 \} \\ & \text{row}(A) \leq \text{row}(B') \wedge \text{col}(A) < \text{col}(B') \\ \equiv & \{ \text{def. row, col} \} \\ & \text{row}([ A \mid b ]) \leq \text{row}(B') \wedge \text{col}([ A \mid b ]) \leq \text{col}(B') \\ \equiv & \{ [ A \mid b ], B' \in Q, \text{ therefore } [ A \mid b ], B' \in \text{pref}(P) \} \\ & [ A \mid b ] \in \text{pref}(B') \\ \Rightarrow & \{ \text{theorem 6.5; } [ A \mid b ], B' \in Q; Q \text{ is a reachable set} \} \\ & [ A \mid b ] \in \text{suff}(B') \\ \equiv & \{ [ A \mid b ] \in \text{pref}(P) \} \\ & [ A \mid b ] \in \text{suff}(B') \cap \text{pref}(P) \end{aligned}$$

So we know that in this case the left hand side of  $P0$  is equal to:

$$\begin{aligned} & q(Ds) \cup \langle \text{set } A, b, c : \text{SIZE}(A, b, c) \wedge [ A \mid b ] \in \text{suff}(B') \cap \text{pref}(P) \wedge \\ & \left[ \frac{A}{c} \right] \in \text{suff}(C') \cap \text{pref}(P) \wedge \sigma = P[\text{row}(A), \text{col}(A)] : \left[ \frac{A \mid b}{c \mid \sigma} \right] \rangle \cup \\ & \delta(q(Bs'), q(Cs'), \sigma) \end{aligned}$$

Again, for the following derivation, we look at a subexpression, assuming we have matrices  $A$ ,  $b$  and  $c$  with  $\text{SIZE}(A, b, c)$ :

$$\begin{aligned} & [ A \mid b ] \in \text{suff}(B') \cap \text{pref}(P) \wedge \left[ \frac{A}{c} \right] \in \text{suff}(C') \cap \text{pref}(P) \wedge \sigma = P[\text{row}(A), \text{col}(A)] \\ \equiv & \{ \text{prop. } \cap \} \\ & [ A \mid b ] \in \text{suff}(B') \wedge \left[ \frac{A}{c} \right] \in \text{suff}(C') \wedge [ A \mid b ] \in \text{pref}(P) \wedge \left[ \frac{A}{c} \right] \in \text{pref}(P) \wedge \\ & \sigma = P[\text{row}(A), \text{col}(A)] \\ \equiv & \{ \text{property pref} \} \\ & [ A \mid b ] \in \text{suff}(B') \wedge \left[ \frac{A}{c} \right] \in \text{suff}(C') \wedge \left[ \frac{A \mid b}{c \mid \sigma} \right] \in \text{pref}(P) \\ \equiv & \{ B', C' \in \text{pref}(P) \} \\ & [ A \mid b ] \in \text{suff}(P[0 \dots \text{row}(B')][0 \dots \text{col}(B')]) \wedge \\ & \left[ \frac{A}{c} \right] \in \text{suff}(P[0 \dots \text{row}(C')][0 \dots \text{col}(C')]) \wedge \left[ \frac{A \mid b}{c \mid \sigma} \right] \in \text{pref}(P) \\ \equiv & \{ \text{row}(B') + 1 = \text{row}(C'), \text{col}(B') = \text{col}(C') + 1 \} \end{aligned}$$

$$\begin{aligned}
& \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \in \text{suff}(P[0 \dots \text{row}(C') - 1][0 \dots \text{col}(B')]) \wedge \\
& \left[ \begin{array}{c|c} A & \\ \hline c & \sigma \end{array} \right] \in \text{suff}(P[0 \dots \text{row}(C')][0 \dots \text{col}(B') - 1]) \wedge \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \in \text{pref}(P) \\
\equiv & \{ P[\text{row}(B'), \text{col}(C')] = \sigma; \text{property suff} \} \\
& \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \in \text{suff}(P[0 \dots \text{row}(C')][0 \dots \text{col}(B')]) \wedge \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \in \text{pref}(P) \\
\equiv & \{ \text{property} \cap \} \\
& \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \in \text{suff}(P[0 \dots \text{row}(C')][0 \dots \text{col}(B')]) \cap \text{pref}(P)
\end{aligned}$$

Now we can get back to our main derivation:

$$\begin{aligned}
& q(Ds) \cup \langle \text{set } A, b, c : \text{SIZE}(A, b, c) \wedge \\
& \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \in \text{suff}(P[0 \dots \text{row}(C')][0 \dots \text{col}(B')]) \cap \text{pref}(P) : \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \rangle \cup \\
& \delta(q(Bs'), q(Cs'), \sigma) \\
= & \{ \text{def. SIZE} \} \\
& q(Ds) \cup \langle \text{set } A, b, c : \text{row}(A) < m_1 \wedge \text{col}(A) < m_2 \wedge \\
& \text{row}(b) = \text{row}(A) \wedge \text{col}(b) = 1 \wedge \text{row}(c) = 1 \wedge \text{col}(c) = \text{col}(A) \wedge \\
& \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \in \text{suff}(P[0 \dots \text{row}(C')][0 \dots \text{col}(B')]) \cap \text{pref}(P) : \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \rangle \cup \\
& \delta(q(Bs'), q(Cs'), \sigma) \\
= & \{ \text{row}(C') \leq m_1, \text{col}(B') \leq m_2 \} \\
& q(Ds) \cup \langle \text{set } A : A \notin \mathcal{ES} \wedge A \in \text{suff}(P[0 \dots \text{row}(C')][0 \dots \text{col}(B')]) \cap \text{pref}(P) : A \rangle \cup \\
& \delta(q(Bs'), q(Cs'), \sigma) \\
= & \{ \text{row}(C') \leq m_1 \text{ and } \text{col}(B') \leq m_2, \text{ therefore } \mathcal{ES}_{\text{row}(C'), \text{col}(B')} \in \delta(q(Bs'), q(Cs'), \sigma) \} \\
& q(Ds) \cup \langle \text{set } A : A \in \text{suff}(P[0 \dots \text{row}(C')][0 \dots \text{col}(B')]) \cap \text{pref}(P) : A \rangle \cup \\
& \delta(q(Bs'), q(Cs'), \sigma) \\
= & \{ \text{set calculus} \} \\
& q(Ds) \cup \text{suff}(P[0 \dots \text{row}(C')][0 \dots \text{col}(B')]) \cap \text{pref}(P) \cup \delta(q(Bs'), q(Cs'), \sigma) \\
= & \{ \text{property (8)} \} \\
& q(Ds \triangleleft P[0 \dots \text{row}(C')][0 \dots \text{col}(B')]) \cup \delta(q(Bs'), q(Cs'), \sigma)
\end{aligned}$$

This leads to the following program fragment:

```

Ds := Ds < P[0 .. row(C')][0 .. col(B)];
Bs := Bs';
Cs := Cs'

```

Of course we still need to verify that this does not violate the other invariants  $P1$ ,  $P2$  and  $P3$ . From (10), the definition of  $OL$ , it is not hard to see that  $P1$  and  $P2$  still hold after removing the first elements of  $Bs$  and  $Cs$ . For  $P3$ , there are two relevant cases:  $Ds = []$  and  $Ds = Ds' \triangleleft D'$ .

$$\begin{aligned}
& OL([] \triangleleft Ds) \\
\equiv & \{ (10) \} \\
& \text{true}
\end{aligned}$$

$$\begin{aligned}
& OL(Ds' \triangleleft D' \triangleleft P[0 \dots \text{row}(C')][0 \dots \text{col}(B')]) \\
\equiv & \{ \text{property (11)} \} \\
& OL(Ds' \triangleleft D') \wedge \text{row}(P[0 \dots \text{row}(C')][0 \dots \text{col}(B')]) < \text{row}(D') \wedge \\
& \quad \text{col}(D') < \text{col}(P[0 \dots \text{row}(C')][0 \dots \text{col}(B')]) \\
\equiv & \{ Ds' \triangleleft D' = Ds, P2; \text{def. row, col} \} \\
& \text{row}(C') < \text{row}(D') \wedge \text{col}(D') < \text{col}(B') \\
\equiv & \{ \text{idempotence of } \wedge, \text{row}(B') = \text{row}(C') + 1, \text{col}(B') + 1 = \text{col}(C') \} \\
& \text{row}(B') + 1 < \text{row}(D') \wedge \text{col}(D') < \text{col}(B') \wedge \\
& \quad \text{row}(C') < \text{row}(D') \wedge \text{col}(D') < \text{col}(C') + 1 \\
\equiv & \{ \bullet OL_r(Ds' \triangleleft D', B' \triangleright Bs'), OL_c(Ds' \triangleleft D', C' \triangleright Cs') \} \\
& \text{true}
\end{aligned}$$

We introduce the following two extra invariants:

$$\begin{aligned}
P4 : & \quad OL_r(Ds, Bs) \\
P5 : & \quad OL_c(Ds, Cs)
\end{aligned}$$

Predicates  $OL_r$  and  $OL_c$  are defined as follows:

$$\begin{aligned}
OL_r([], As) & \equiv \text{true} \\
OL_r(Es, []) & \equiv \text{true} \\
OL_r(Es \triangleleft E, A \triangleright As) & \equiv \text{row}(A) + 1 < \text{row}(E) \wedge \text{col}(E) < \text{col}(A)
\end{aligned} \tag{13}$$

$$\begin{aligned}
OL_c([], As) & \equiv \text{true} \\
OL_c(Es, []) & \equiv \text{true} \\
OL_c(Es \triangleleft E, A \triangleright As) & \equiv \text{row}(A) < \text{row}(E) \wedge \text{col}(E) < \text{col}(A) + 1
\end{aligned} \tag{14}$$

From these definitions we can see that extra invariants  $P4$  and  $P5$  hold initially, since  $[]$  is the initial value of  $Ds$ . It should also be obvious that  $OL_r(Ds \triangleleft P[0 \dots \text{row}(C')][0 \dots \text{col}(B')], Bs')$  follows from  $\text{row}(B') + 1 = \text{row}(C')$  and  $OL(B' \triangleright Bs')$  (which is invariant  $P1$ ). Symmetrically, we know that  $OL_c(Ds \triangleleft P[0 \dots \text{row}(C')][0 \dots \text{col}(B')], Cs')$  follows from  $\text{col}(B') = \text{col}(C') + 1$  and  $OL(C' \triangleright Cs')$ .

In case 0 we have seen how to update  $Ds$ ,  $Bs$  and  $Cs$  when a combination of  $B'$ ,  $C'$  and  $\sigma$  actually occurs as a prefix of the text. The remaining cases are the various situations in which there is no way to combine  $B'$ ,  $C'$  and  $\sigma$  into such a prefix.

### Case 1

Here we will take a look at the case where  $\text{row}(C') < \text{row}(B') + 1$ . Again we start by examining the left-hand side of invariant  $P0$ .

$$\begin{aligned}
& q(Ds) \cup \delta(q(B' \triangleright Bs'), q(C' \triangleright Cs'), \sigma) \\
= & \{ \text{definition } \delta \} \\
& q(Ds) \cup \langle \text{set } A, b, c : \text{SIZE}(A, b, c) \wedge [A \mid b] \in q(B' \triangleright Bs') \wedge
\end{aligned}$$

$$\left[ \frac{A}{c} \right] \in q(C' \triangleright Cs') \wedge \sigma = P[\text{row}(A), \text{col}(A)] : \left[ \frac{A}{c} \mid \frac{b}{\sigma} \right] \in \mathcal{ES}_{m_1, m_2} \cup$$

Here too, we will take a look at a subexpression in detail. We assume we have  $A$ ,  $b$  and  $c$  satisfying  $SIZE(A, b, c)$  and we derive:

$$\begin{aligned} & \left[ \frac{A}{c} \right] \in q(C' \triangleright Cs') \\ \equiv & \{ \text{definition } q \} \\ & \left[ \frac{A}{c} \right] \in (\text{suff}(C') \cap \text{pref}(P)) \cup q(Cs') \\ \Rightarrow & \{ \text{property suff, invariant } P2: OL(C' \triangleright Cs') \} \\ & \text{row}\left(\left[ \frac{A}{c} \right]\right) \leq \text{row}(C') \\ \Rightarrow & \{ \text{row}(C') < \text{row}(B') + 1 \} \\ & \text{row}\left(\left[ \frac{A}{c} \right]\right) < \text{row}(B') + 1 \\ \equiv & \{ \text{definition row} \} \\ & \text{row}([A \mid b]) < \text{row}(B') \\ \Rightarrow & \{ \text{property of functions} \} \\ & [A \mid b] \neq B' \end{aligned}$$

We conclude:

$$\begin{aligned} & [A \mid b] \in q(B' \triangleright Bs') \\ \equiv & \{ (7) \text{ (definition } q) \} \\ & [A \mid b] \in (\text{suff}(B') \cap \text{pref}(P)) \cup q(Bs') \\ \equiv & \{ [A \mid b] \neq B' \} \\ & [A \mid b] \in ((\text{suff}(B') \cap \text{pref}(P)) \setminus \{B'\}) \cup q(Bs') \\ \equiv & \{ \bullet \text{ spec. } r \} \\ & [A \mid b] \in q(r(B')) \cup q(Bs') \end{aligned}$$

Here we introduce function  $r : \text{pref}(P) \rightarrow \mathcal{L}(\text{pref}(P))$ , which, given  $A \in \text{pref}(P)$ , returns the list of maximal elements of  $(\text{suff}(A) \cap \text{pref}(P)) \setminus \{A\}$ . In other words,  $r(A)$  is the list satisfying the following properties:

$$\begin{aligned} q(r(A)) &= (\text{suff}(A) \cap \text{pref}(P)) \setminus \{A\} \\ OL(r(A)) & \end{aligned}$$

The values of function  $r$  can be precomputed for every prefix of  $P$ . We will get back to this later, in section 6.3.4; for now we just assume we have such a function  $r$ .

We can now conclude that the left-hand-side of  $P0$  is in this case equal to:

$$q(Ds) \cup \langle \text{set } A, b, c : \text{SIZE}(A, b, c) \wedge [A \mid b] \in q(r(B')) \cup q(Bs') \wedge \left[ \frac{A}{c} \right] \in q(C' \triangleright Cs') \wedge \sigma = P[\text{row}(A), \text{col}(A)] : \left[ \frac{A \mid b}{c \mid \sigma} \right] \rangle \cup \mathcal{ES}_{m_1, m_2}$$

This suggests that we might want to replace  $Bs$  by  $r(B') \ ++ \ Bs'$ . That would be sufficient to ensure that  $P0$  holds again. However, there is no way to guarantee that  $OL(r(B') \ ++ \ Bs')$  and  $OL_r(Ds, r(B') \ ++ \ Bs')$  hold.

For example,  $r(B')$  may very well contain a matrix  $R$  that is a strict suffix of an element of  $Bs'$ . Then  $R$  is not a maximal element of  $q(r(B') \ ++ \ Bs')$  and that means that  $OL(r(B') \ ++ \ Bs')$  does not hold. However, in this case we also know that every matrix in  $\text{suff}(R)$  is already represented by  $Bs'$ , so we can simply omit matrix  $R$ .

Because of invariant  $OL(B' \triangleright Bs')$  and  $OL(r(B'))$ , we know that the elements of  $r(B')$  that are also a suffix of some element of  $Bs'$  form a tail of  $r(B')$ . So eliminating these elements of  $r(B')$  consists of a linear search starting at the last element of  $r(B')$ , eliminating every element until one is found that is not a suffix of some element of  $Bs'$ .

Similarly, we cannot guarantee that  $OL_r(Ds, r(B') \ ++ \ Bs')$  holds, but we do know that we can omit the matrices in  $r(B')$  which have already been considered in the computation of  $Ds$ . We also know that those matrices occur at the beginning of  $r(B')$ .

First we will focus on  $OL$  and try to compute a list  $Rs$ , such that  $q(Rs \ ++ \ Bs') = q(r(B') \ ++ \ Bs')$ , but also  $OL(Rs \ ++ \ Bs')$ . We distinguish two possible cases for  $Bs'$ :  $Bs' = []$  and  $Bs' \neq []$ .

- If  $Bs' = []$ , we can make the following observation:

$$\begin{aligned} & OL(r(B') \ ++ \ []) \\ \equiv & \quad \{ \text{definition } ++ \} \\ & OL(r(B')) \\ \equiv & \quad \{ \text{specification } r \} \\ & \text{true} \end{aligned}$$

So in this case, the choice  $Rs = r(B')$  does suffice.

- If  $Bs'$  is not empty, we can write it as  $B'' \triangleright Bs''$ , for some matrix  $B''$  and list  $Bs''$ . To compute a list  $Rs$  satisfying  $q(Rs \ ++ \ Bs') = q(r(B') \ ++ \ Bs')$  and  $OL(Rs \ ++ \ Bs')$ , we initially choose  $Rs = r(B')$  and introduce the following invariants:

$$\begin{aligned} Q0 & \quad OL(Rs) \\ Q1 & \quad q(Rs \ ++ \ Bs') = q(r(B') \ ++ \ Bs') \end{aligned}$$

Concerning termination, we again look at two different cases:  $Rs = []$  and  $Rs \neq []$ .

- Assuming  $Rs = []$ , we derive:

$$\begin{aligned} & OL([] \ ++ \ (B'' \triangleright Bs'')) \\ \equiv & \quad \{ \text{definition } ++ \} \\ & OL(B'' \triangleright Bs'') \\ \Leftarrow & \quad \{ \text{definition } OL \} \\ & OL(B' \triangleright B'' \triangleright Bs'') \\ \equiv & \quad \{ P1 \} \\ & \text{true} \end{aligned}$$

So if  $Rs$  is empty, we can terminate this computation.

– If  $R \neq []$ , we can write  $Rs$  as  $Rs' \triangleleft R'$ .

$$\begin{aligned}
& OL((Rs' \triangleleft R') \uplus (B'' \triangleright Bs'')) \\
\equiv & \{ \text{property (12)} \} \\
& OL(Rs' \triangleleft R') \wedge OL(B'' \triangleright Bs'') \wedge \text{row}(B'') < \text{row}(R') \wedge \text{col}(R') < \text{col}(B') \\
\equiv & \{ Q0, P1 \} \\
& \text{row}(B'') < \text{row}(R') \wedge \text{col}(R') < \text{col}(B'') \\
\Leftarrow & \{ R' \in r(B'), \text{ therefore } R' \in \text{suff}(B') \text{ and } \text{col}(R') \leq \text{col}(B') \} \\
& \text{row}(B'') < \text{row}(R') \wedge \text{col}(B') < \text{col}(B'') \\
\equiv & \{ P1 : OL(B' \triangleright B'' \triangleright Bs'') \} \\
& \text{row}(B'') < \text{row}(R')
\end{aligned}$$

If  $\text{row}(B'') < \text{row}(R')$ , we can terminate the computation as well. Otherwise, we can derive:

$$\begin{aligned}
& \text{row}(R') \leq \text{row}(B'') \\
\Rightarrow & \{ \text{col}(R') < \text{col}(B'') \text{ (see previous derivation); } R', B'' \in \text{pref}(P) \} \\
& R' \in \text{pref}(B'') \\
\Rightarrow & \{ R', B'' \in Q, \text{ theorem 6.5} \} \\
& R' \in \text{suff}(B'') \\
\equiv & \{ \text{property suff} \} \\
& \text{suff}(R') \subseteq \text{suff}(B') \\
\Rightarrow & \{ \text{set calculus} \} \\
& \text{suff}(R') \cap \text{pref}(P) \subseteq \text{suff}(B') \cap \text{pref}(P)
\end{aligned}$$

We can use this in the following derivation, which starts with the left-hand side of invariant  $Q1$ :

$$\begin{aligned}
& q((Rs' \triangleleft R') \uplus (B'' \triangleright Bs'')) \\
= & \{ (7), (8) \text{ and } (9) \} \\
& q(Rs') \cup (\text{suff}(R') \cap \text{pref}(P)) \cup (\text{suff}(B'') \cap \text{pref}(P)) \cup q(Bs'') \\
= & \{ \text{suff}(R') \cap \text{pref}(P) \subseteq \text{suff}(B') \cap \text{pref}(P) \} \\
& q(Rs') \cup (\text{suff}(B'') \cap \text{pref}(P)) \cup q(Bs'') \\
= & \{ (7), (9) \} \\
& q(Rs' \uplus (B'' \triangleright Bs''))
\end{aligned}$$

So the algorithm to compute  $Rs$  becomes:

```

Rs := r(B');
if Bs' = [] → skip
|| Bs' :: B'' ▷ Bs'' →
  do Rs :: Rs' ◁ R' cand row(R') < row(B'') →
    Rs := Rs'
  od
fi

```

Now we still need to establish  $OL_r(Ds, Rs \uparrow Bs')$ . We do this by an algorithm similar to the one presented above. We will only eliminate elements of  $Rs$ , so we can be sure that this algorithm will not falsify  $OL(Rs \uparrow Bs')$ .

Again, we will consider two cases:  $Ds = []$  and  $Ds \neq []$ .

- If  $Ds = []$ , we can simply observe:

$$\begin{aligned} & OL_r([], Rs \uparrow Bs') \\ \equiv & \quad \{ (13) \} \\ & \text{true} \end{aligned}$$

- If  $Ds \neq []$ , it is of the form  $Ds' \triangleleft D'$ . We distinguish two different cases for the structure of  $Rs$ .

- If  $Rs = []$ , we get:

$$\begin{aligned} & OL_r(Ds' \triangleleft D', [] \uparrow Bs') \\ \equiv & \quad \{ \text{definition } \uparrow \} \\ & OL_r(Ds' \triangleleft D', Bs') \\ \equiv & \quad \{ P1, P4 \} \\ & \text{true} \end{aligned}$$

- If  $Rs \neq []$ , we can write  $Rs$  as  $R' \triangleright Rs'$ . We get the following derivation:

$$\begin{aligned} & OL_r(Ds' \triangleleft D', (R' \triangleright Rs') \uparrow Bs') \\ \equiv & \quad \{ (13) \} \\ & \text{row}(R') + 1 < \text{row}(D') \wedge \text{col}(D') < \text{col}(R') \\ \Leftarrow & \quad \{ R' \in \text{suff}(B') \} \\ & \text{row}(B') + 1 < \text{row}(D') \wedge \text{col}(D') < \text{col}(R') \\ \equiv & \quad \{ P4 \} \\ & \text{col}(D') < \text{col}(R') \end{aligned}$$

If this does not hold (that is, in the case that  $\text{col}(R') \leq \text{col}(D')$ ), we want to eliminate  $R'$  from the list, much like in the previous algorithm. We will now show that this does not falsify  $P0$ , with  $Bs$  replaced by  $Rs \uparrow Bs'$ . We get the following derivation:

$$\begin{aligned} & q(Ds' \triangleleft D') \cup \delta(q(R' \triangleright Rs' \uparrow Bs'), q(C' \triangleright Cs'), \sigma) \\ = & \quad \{ \text{definition } \delta \} \\ & q(Ds' \triangleleft D') \cup \\ & \quad \langle \text{set } A, b, c : \text{SIZE}(A, b, c) \wedge [A \mid b] \in q(R' \triangleright Rs' \uparrow Bs') \wedge \\ & \quad \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \in q(C' \triangleright Cs') \wedge \sigma = P[\text{row}(A), \text{col}(A)] : \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \rangle \cup \\ & \quad \mathcal{E}S_{m_1, m_2} \\ = & \quad \{ \text{domain split, definition } \delta \} \\ & q(Ds' \triangleleft D') \cup \\ & \quad \langle \text{set } A, b, c : \text{SIZE}(A, b, c) \wedge [A \mid b] \in \text{suff}(R') \cap \text{pref}(P) \wedge \\ & \quad \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \in q(C' \triangleright Cs') \wedge \sigma = P[\text{row}(A), \text{col}(A)] : \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \rangle \cup \end{aligned}$$

$$\begin{aligned}
& \delta(q(Rs' \# Bs'), q(C' \triangleright Cs'), \sigma) \\
= & \{ \bullet \text{ see below } \} \\
& q(Ds' \triangleleft D') \cup \delta(q(Rs' \# Bs'), q(C' \triangleright Cs'), \sigma)
\end{aligned}$$

For the step that we still need to prove, we assume we have matrices  $A, b$  and  $c$ , which satisfy:

$$\begin{aligned}
& SIZE(A, b, c) \wedge [A \mid b] \in \text{suff}(R') \cap \text{pref}(P) \wedge \\
& \left[ \begin{array}{c|c} A & b \\ \hline c & \end{array} \right] \in q(C' \triangleright Cs') \wedge \sigma = P[\text{row}(A), \text{col}(A)]
\end{aligned}$$

We get the following derivation:

$$\begin{aligned}
& [A \mid b] \in \text{suff}(R') \cap \text{pref}(P) \\
\Rightarrow & \{ \text{property suff} \} \\
& \text{row}([A \mid b]) \leq \text{row}(R') \wedge \text{col}([A \mid b]) \leq \text{col}(R') \\
\Rightarrow & \{ \text{col}(R') \leq \text{col}(D') \} \\
& \text{row}([A \mid b]) \leq \text{row}(R') \wedge \text{col}([A \mid b]) \leq \text{col}(D') \\
\Rightarrow & \{ R' \in \text{suff}(B') \} \\
& \text{row}([A \mid b]) \leq \text{row}(B') \wedge \text{col}([A \mid b]) \leq \text{col}(D') \\
\Rightarrow & \{ P4 : OL_r(Ds' \triangleleft D', B' \triangleright Bs') \} \\
& \text{row}([A \mid b]) < \text{row}(D') - 1 \wedge \text{col}([A \mid b]) \leq \text{col}(D') \\
\equiv & \{ \text{def. row, col} \} \\
& \text{row}\left(\left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right]\right) < \text{row}(D') \wedge \text{col}\left(\left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right]\right) \leq \text{col}(D') \\
\Rightarrow & \{ [A \mid b] \in \text{pref}(P), \left[ \begin{array}{c|c} A & b \\ \hline c & \end{array} \right] \in \text{pref}(P) \text{ and } \sigma = P[\text{row}(A), \text{col}(A)], \\
& \text{therefore: } \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \in \text{pref}(P); D' \in \text{pref}(P) \\
& \} \\
& \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \in \text{pref}(D') \\
\Rightarrow & \{ \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right], D' \in \delta(Q, R, \sigma); \text{theorem 6.5} \} \\
& \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \in \text{suff}(D') \\
\equiv & \{ [A \mid b] \in \text{pref}(P), \left[ \begin{array}{c|c} A & b \\ \hline c & \end{array} \right] \in \text{pref}(P) \text{ and } \sigma = P[\text{row}(A), \text{col}(A)] \} \\
& \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \in \text{suff}(D') \cap \text{pref}(P) \\
\Rightarrow & \{ \text{set calculus} \} \\
& \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \in q(Ds') \cup (\text{suff}(D') \cap \text{pref}(P)) \\
\equiv & \{ \text{property (8)} \} \\
& \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \in q(Ds' \triangleleft D')
\end{aligned}$$

So the following program fragment suffices:

```

if  $Ds = [] \rightarrow$  skip
 $\parallel$   $Ds :: Ds' \triangleleft D' \rightarrow$ 
  do  $Rs :: R' \triangleright Rs' \text{ and } \text{col}(R') \leq \text{col}(D') \rightarrow$ 
     $Rs := Rs'$ 
  od
fi

```

## Case 2

We will now examine the case:  $\text{row}(B') + 1 < \text{row}(C')$ .

$$\begin{aligned}
& q(Ds) \cup \delta(q(B' \triangleright Bs'), q(C' \triangleright Cs'), \sigma) \\
= & \{ \text{definition } \delta \} \\
& q(Ds) \cup \langle \text{set } A, b, c : \text{SIZE}(A, b, c) \wedge [A \mid b] \in q(B' \triangleright Bs') \wedge \\
& \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \in q(C' \triangleright Cs') \wedge \sigma = P[\text{row}(A), \text{col}(A)] : \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \rangle \cup \\
& \mathcal{E}S_{m_1, m_2}
\end{aligned}$$

Here, we can derive, somewhat symmetrically to case 1:

$$\begin{aligned}
& [A \mid b] \in q(B \triangleright Bs') \\
\equiv & \{ (7) \text{ (definition } q) \} \\
& [A \mid b] \in (\text{suff}(B') \cap \text{pref}(P)) \cup q(Bs') \\
\Rightarrow & \{ OL(B' \triangleright Bs') \} \\
& \text{row}([A \mid b]) \leq \text{row}(B') \\
\Rightarrow & \{ \text{row}(B') + 1 < \text{row}(C') \} \\
& \text{row}([A \mid b]) < \text{row}(C') - 1 \\
\equiv & \{ \text{def. row} \} \\
& \text{row}\left(\left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right]\right) < \text{row}(C') \\
\Rightarrow & \{ \text{property functions} \} \\
& \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \neq C'
\end{aligned}$$

And we can conclude:

$$\begin{aligned}
& \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \in q(C' \triangleright Cs') \\
\equiv & \{ (7) \text{ (definition } q) \} \\
& \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \in (\text{suff}(C') \cap \text{pref}(P)) \cup q(Cs') \\
\equiv & \{ \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \neq C' \}
\end{aligned}$$

$$\begin{aligned}
& \left[ \frac{A}{c} \right] \in ((\text{suff}(C') \cap \text{pref}(P)) \setminus \{C'\}) \cup q(Cs') \\
\equiv & \quad \{ \text{specification } r \} \\
& \left[ \frac{A}{c} \right] \in r(C') \cup q(Cs')
\end{aligned}$$

To ensure that  $P2$  and  $P5$  hold, we need to eliminate certain elements of  $r(C')$ . The derivation of such an algorithm is symmetrical to case 1 and will be omitted here; instead we just present the entire algorithm immediately.

```

Rs := r(C');
if Cs' = [] → skip
|| Cs' :: C'' ▷ Cs''' →
  do Rs :: Rs' ◁ R' and row(R') ≤ row(C'') →
    Rs := Rs'
  od
fi;
if Ds = [] → skip
|| Ds :: Ds' ◁ D' →
  do Rs :: R' ◁ Rs' and col(R') < col(D') →
    Rs := Rs'
  od
fi

```

After computing  $Rs$  using this algorithm, we can safely replace  $Cs$  with  $Rs ++ Cs'$ .

### Case 3

Here we examine the case:  $\text{row}(B') + 1 = \text{row}(C') \wedge \text{col}(C') + 1 < \text{col}(B')$ .

Then:

$$\begin{aligned}
& q(Ds) \cup \delta(q(B' \triangleright Bs'), q(C' \triangleright Cs'), \sigma) \\
= & \quad \{ \text{definition } \delta \} \\
& q(Ds) \cup \langle \text{set } A, b, c : \text{SIZE}(A, b, c) \wedge [A \mid b] \in q(B' \triangleright Bs') \wedge \\
& \quad \left[ \frac{A}{c} \right] \in q(C' \triangleright Cs') \wedge \sigma = P[\text{row}(A), \text{col}(A)] : \left[ \frac{A \mid b}{c \mid \sigma} \right] \rangle \cup \\
& \quad \mathcal{E}S_{m_1, m_2}
\end{aligned}$$

As in case 1, we can now show:  $\left[ \frac{A}{c} \right] \in q(C' \triangleright Cs') \Rightarrow [A \mid b] \neq B'$  (for  $A$ ,  $b$  and  $c$  satisfying  $\text{SIZE}(A, b, c)$ ).

$$\begin{aligned}
& \left[ \frac{A}{c} \right] \in q(C' \triangleright Cs') \\
\equiv & \quad \{ (7) \} \\
& \left[ \frac{A}{c} \right] \in (\text{suff}(C') \cap \text{pref}(P)) \cup Cs' \\
\equiv & \quad \{ \text{property } \cup \}
\end{aligned}$$

$$\left[ \frac{A}{c} \right] \in \text{suff}(C') \cap \text{pref}(P) \vee \left[ \frac{A}{c} \right] \in Cs'$$

We proceed to show that both disjuncts imply  $[A \mid b] \neq B'$ .

- $\left[ \frac{A}{c} \right] \in \text{suff}(C') \cap \text{pref}(P)$ 
  - $\Rightarrow$  { property suff }
  - $\text{col}\left(\left[ \frac{A}{c} \right]\right) \leq \text{col}(C')$
  - $\equiv$  { definition col }
  - $\text{col}([A \mid b]) \leq \text{col}(C') + 1$
  - $\Rightarrow$  {  $\text{col}(C') + 1 < \text{col}(B')$  }
  - $\text{col}([A \mid b]) < \text{col}(B')$
  - $\Rightarrow$  { property functions }
  - $[A \mid b] \neq B'$
- $\left[ \frac{A}{c} \right] \in q(Cs')$ 
  - $\Rightarrow$  { P2:  $OL(C' \triangleright Cs')$  }
  - $\text{row}\left(\left[ \frac{A}{c} \right]\right) < \text{row}(C')$
  - $\equiv$  {  $\text{row}(B') + 1 = \text{row}(C')$  }
  - $\text{row}\left(\left[ \frac{A}{c} \right]\right) < \text{row}(B') + 1$
  - $\equiv$  { definition row }
  - $\text{row}([A \mid b]) < \text{row}(B')$
  - $\Rightarrow$  { property functions }
  - $[A \mid b] \neq B'$

Now we can draw the same conclusion as in case 1:

$$\begin{aligned}
& [A \mid b] \in q(B' \triangleright Bs') \\
\equiv & \{ (7) \} \\
& [A \mid b] \in (\text{suff}(B') \cap \text{pref}(P)) \cup q(Bs') \\
\equiv & \{ [A \mid b] \neq B' \} \\
& [A \mid b] \in ((\text{suff}(B') \cap \text{pref}(P)) \setminus \{B'\}) \cup q(Bs') \\
\equiv & \{ \text{spec. } r \} \\
& [A \mid b] \in q(r(B')) \cup q(Bs')
\end{aligned}$$

Note that in the rest of the derivation in case 1 we *never* use the assumption  $\text{row}(C') \leq \text{row}(B')$ . That means that we can now use the same algorithm for computing a list  $Rs$  from list  $r(B')$ , such that we can replace  $Bs$  by  $Rs \# Bs'$ .

#### Case 4

Case 4 is:  $\text{row}(B') + 1 = \text{row}(C') \wedge \text{col}(B') < \text{col}(C') + 1$ . We derive:

$$\begin{aligned}
& q(Ds) \cup \delta(q(B' \triangleright Bs'), q(C' \triangleright Cs'), \sigma) \\
= & \{ \text{definition } \delta, (7) \text{ (definition } q) \} \\
& q(Ds) \cup \langle \text{set } A, b, c : \text{SIZE}(A, b, c) \wedge [A \mid b] \in q(B' \triangleright Bs') \wedge \\
& \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \in q(C' \triangleright Cs') \wedge \sigma = P[\text{row}(A), \text{col}(A)] : \left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \rangle \cup \\
& \mathcal{ES}_{m_1, m_2}
\end{aligned}$$

For  $A$ ,  $b$  and  $c$  satisfying  $\text{SIZE}(A, b, c)$ , we derive:

$$\begin{aligned}
& [A \mid b] \in q(B' \triangleright Bs') \\
\equiv & \{ (7), \text{property } \cap \} \\
& [A \mid b] \in \text{suff}(B') \cap \text{pref}(P) \vee [A \mid b] \in q(Bs')
\end{aligned}$$

We examine both disjuncts seperately to show that both imply  $\left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \neq C'$ :

- $[A \mid b] \in \text{suff}(B') \cap \text{pref}(P)$ 
  - $\Rightarrow$  { property suff }
  - $\text{col}([A \mid b]) \leq \text{col}(B')$
  - $\Rightarrow$  {  $\text{col}(B') \leq \text{col}(C')$  }
  - $\text{col}([A \mid b]) \leq \text{col}(C')$
  - $\equiv$  { definition col }
  - $\text{col}\left(\left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right]\right) < \text{col}(C')$
  - $\Rightarrow$  { property functions }
  - $\left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \neq C'$
- $[A \mid b] \in q(Bs')$ 
  - $\Rightarrow$  {  $OL(B' \triangleright Bs')$  }
  - $\text{row}([A \mid b]) < \text{row}(B')$
  - $\equiv$  { definition row }
  - $\text{row}\left(\left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right]\right) < \text{row}(B') + 1$
  - $\equiv$  {  $\text{row}(B') + 1 = \text{row}(C')$  }
  - $\text{row}\left(\left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right]\right) < \text{row}(C')$
  - $\Rightarrow$  { property functions }
  - $\left[ \begin{array}{c|c} A & b \\ \hline c & \sigma \end{array} \right] \neq C'$

Therefore we have the same conclusion as in case 2:

$$\begin{aligned}
& \left[ \frac{A}{c} \right] \in q(C' \triangleright Cs') \\
\equiv & \{ (7) \} \\
& \left[ \frac{A}{c} \right] \in (\text{suff}(C') \cap \text{pref}(P)) \cup q(Cs') \\
\equiv & \left\{ \left[ \frac{A}{c} \right] \neq C' \right\} \\
& \left[ \frac{A}{c} \right] \in ((\text{suff}(C') \cap \text{pref}(P)) \setminus \{C'\}) \cup q(Cs') \\
\equiv & \{ \text{specification } r \} \\
& \left[ \frac{A}{c} \right] \in r(C') \cup q(Cs')
\end{aligned}$$

And again, we can use the same algorithm presented in case 2 to compute a list  $R_s$ , so we can replace  $C_s$  by  $R_s \uplus Cs'$ .

### Case 5

Our final case is the case where the sizes of  $B'$  and  $C'$  do match, but  $\sigma$  does not occur in the right spot in the pattern:  $\text{row}(B') + 1 = \text{row}(C') \wedge \text{col}(B') = \text{col}(C') + 1 \wedge P[\text{row}(B'), \text{col}(C')] \neq \sigma$ .

$$\begin{aligned}
& q(Ds) \cup \delta(q(B' \triangleright Bs'), q(C' \triangleright Cs'), \sigma) \\
= & \{ \text{definition } \delta \} \\
& q(Ds) \cup \langle \text{set } A, b, c : \text{SIZE}(A, b, c) \wedge [A \mid b] \in q(B' \triangleright Bs') \wedge \\
& \left[ \frac{A}{c} \right] \in q(C' \triangleright Cs') \wedge \sigma = P[\text{row}(A), \text{col}(A)] : \left[ \frac{A \mid b}{c \mid \sigma} \right] \rangle \cup \\
& \mathcal{E}S_{m_1, m_2}
\end{aligned}$$

We assume we have matrices  $A$ ,  $b$  and  $c$ , satisfying:

$$\begin{aligned}
& \text{SIZE}(A, b, c) \wedge [A \mid b] \in q(B' \triangleright Bs') \wedge \\
& \left[ \frac{A}{c} \right] \in q(C' \triangleright Cs') \wedge \sigma = P[\text{row}(A), \text{col}(A)]
\end{aligned}$$

Then the assumption  $P[\text{row}(B'), \text{col}(C')] \neq \sigma$  gives us the following:

$$\begin{aligned}
& P[\text{row}(B'), \text{col}(C')] \neq \sigma \\
\Rightarrow & \{ P[\text{row}(A), \text{col}(A)] = \sigma \} \\
& \text{row}(A) \neq \text{row}(B') \vee \text{col}(A) \neq \text{col}(C')
\end{aligned}$$

We examine both of these disjuncts:

- $\text{row}(A) \neq \text{row}(B')$
- $$\equiv \{ \text{idempotence } \wedge, \text{definition row} \}$$

$$\begin{aligned}
& \text{row}([A \mid b]) \neq \text{row}(B') \wedge \text{row}\left(\left[\frac{A}{c}\right]\right) \neq \text{row}(B') + 1 \\
\equiv & \quad \{ \text{row}(B') + 1 = \text{row}(C') \} \\
& \text{row}([A \mid b]) \neq \text{row}(B') \wedge \text{row}\left(\left[\frac{A}{c}\right]\right) \neq \text{row}(C') \\
\Rightarrow & \quad \{ \text{property functions} \} \\
& [A \mid b] \neq B' \wedge \left[\frac{A}{c}\right] \neq C' \\
\bullet & \quad \text{col}(A) \neq \text{col}(C') \\
\equiv & \quad \{ \text{idempotence } \wedge, \text{ definition col} \} \\
& \text{col}([A \mid b]) \neq \text{col}(C') + 1 \wedge \text{col}\left(\left[\frac{A}{c}\right]\right) \neq \text{col}(C') \\
\equiv & \quad \{ \text{col}(B') = \text{col}(C') + 1 \} \\
& \text{col}([A \mid b]) \neq \text{col}(B') \wedge \text{col}\left(\left[\frac{A}{c}\right]\right) \neq \text{col}(C') \\
\Rightarrow & \quad \{ \text{property functions} \} \\
& [A \mid b] \neq B' \wedge \left[\frac{A}{c}\right] \neq C'
\end{aligned}$$

Using  $[A \mid b] \neq B'$ , we can draw the same conclusion we did in case 1 and case 3:

$$\begin{aligned}
& [A \mid b] \in q(B' \triangleright Bs') \\
\equiv & \quad \{ (7) \} \\
& [A \mid b] \in (\text{suff}(B') \cap \text{pref}(P)) \cup q(Bs') \\
\equiv & \quad \{ [A \mid b] \neq B' \} \\
& [A \mid b] \in ((\text{suff}(B') \cap \text{pref}(P)) \setminus \{B'\}) \cup q(Bs') \\
\equiv & \quad \{ \text{spec. } r \} \\
& [A \mid b] \in q(r(B')) \cup q(Bs')
\end{aligned}$$

However, using  $\left[\frac{A}{c}\right] \neq C'$ , the observation seen in case 2 and case 4 is also correct:

$$\begin{aligned}
& \left[\frac{A}{c}\right] \in q(C' \triangleright Cs') \\
\equiv & \quad \{ (7) \} \\
& \left[\frac{A}{c}\right] \in (\text{suff}(C') \cap \text{pref}(P)) \cup q(Cs') \\
\equiv & \quad \left\{ \left[\frac{A}{c}\right] \neq C' \right\} \\
& \left[\frac{A}{c}\right] \in ((\text{suff}(C') \cap \text{pref}(P)) \setminus \{C'\}) \cup q(Cs') \\
\equiv & \quad \{ \text{specification } r \} \\
& \left[\frac{A}{c}\right] \in r(C') \cup q(Cs')
\end{aligned}$$

Now it is possible that to either update  $Bs$  as in case 1, or to update  $Cs$  as in case 2. However, a third option is to update both  $Bs$  and  $Cs$  here, by replacing  $Bs$  by  $Qs \# Bs'$  and  $Cs$  by  $Rs \# Cs'$ , where  $Qs$  is the result of the algorithm we presented in case 1 and  $Rs$  the result of the algorithm in case 2.

We will not provide the full proof why, after replacing  $Cs$  like this, we can still replace  $Bs$  as well (or, symmetrically, the other way around). Referring to the correctness proof of the algorithm presented in case 1, we can see that the only property of  $\left[ \frac{A}{c} \right]$  that we have used is:  $\left[ \frac{A}{c} \right] \in R$ . This property is not violated by referring to matrices  $\left[ \frac{A}{c} \right]$  in the set  $q(Rs \# Cs')$  instead of  $q(C' \triangleright Cs')$ , because  $q(Rs \# Cs')$  is in fact a subset of  $q(C' \triangleright Cs')$ .

### 6.3.3 Entire precomputation algorithm

In this section we have derived a method of computing a list  $Ds$  which represents  $\delta(Q, R, \sigma)$ , given lists  $Bs$  and  $Cs$ , representing respectively  $Q$  and  $R$ , and symbol  $\sigma$ . Here we will give the entire program text.

```

Ds := [];
do Bs :: B' ▷ Bs' ∧ Cs :: C' ▷ Cs' →
  if row(B') + 1 = row(C') ∧ col(B') = col(C') + 1 ∧ P[row(B'), col(C')] = σ →
    Ds := Ds ◁ P[0 .. row(C')][0 .. col(B')];
    Bs := Bs';
    Cs := Cs'
  || row(C') ≤ row(B') ∨ (row(B') + 1 = row(C') ∧ col(C') + 1 < col(B')) →
    Bs := reducer(B', Ds, Bs')
  || row(B') + 1 < row(C') ∨ (row(B') + 1 = row(C') ∧ col(B') < col(C') + 1) →
    Cs := reducec(C', Ds, Cs')
  || row(B') + 1 = row(C') ∧ col(B') = col(C') + 1 ∧ P[row(B'), col(C')] ≠ σ →
    Bs := reducer(B', Ds, Bs');
    Cs := reducec(C', Ds, Cs')
od;
if Ds = [] → Ds := [Em1,0, E0,m2]
  || Ds :: D' ▷ Ds' ∧ Ds :: Ds'' ◁ D'' →
    if row(D') < m1 → Ds := Em1,0 ▷ Ds
      || m1 ≤ row(D') → skip
    fi;
    if col(D'') < m2 → Ds := Ds ◁ E0,m2
      || m2 ≤ col(D'') → skip
    fi
fi

```

Here the auxiliary function  $reduce_r(B', Ds, Bs')$  is the algorithm from case 1 in section 6.3.2:

```

Rs := r(B');
if Bs' = [] → skip
  || Bs' :: B'' ▷ Bs'' →
    do Rs :: Rs' ◁ R' cand row(R') < row(B'') →
      Rs := Rs'
    od

```

```

fi;
if  $Ds = [] \rightarrow$  skip
   $\parallel Ds :: Ds' \triangleleft D' \rightarrow$ 
    do  $R_s :: R' \triangleright R_s' \text{ cand } \text{col}(R') \leq \text{col}(D') \rightarrow$ 
       $R_s := R_s'$ 
    od
fi;
return( $R_s$ )

```

Function  $reduce_c(C', Ds, C_s')$  is, of course, the algorithm from case 2:

```

 $R_s := r(C')$ ;
if  $C_s' = [] \rightarrow$  skip
   $\parallel C_s' :: C'' \triangleright C_s'' \rightarrow$ 
    do  $R_s :: R_s' \triangleleft R' \text{ cand } \text{row}(R') \leq \text{row}(C'') \rightarrow$ 
       $R_s := R_s'$ 
    od
fi;
if  $Ds = [] \rightarrow$  skip
   $\parallel Ds :: Ds' \triangleleft D' \rightarrow$ 
    do  $R_s :: R' \triangleleft R_s' \text{ cand } \text{col}(R') < \text{col}(D') \rightarrow$ 
       $R_s := R_s'$ 
    od
fi;
return( $R_s$ )

```

We have now seen how to compute the list corresponding to  $\delta(Q, R, \sigma)$  given symbol  $\sigma$  and lists  $Bs$  and  $Cs$ , with  $q(Bs) = Q$  and  $q(Cs) = R$ . We have not given this function a name yet; let us call it  $\delta' : \mathcal{L}(\text{pref}(P)) \times \mathcal{L}(\text{pref}(P)) \times \Sigma \rightarrow \mathcal{L}(\text{pref}(P))$ .

Precomputing  $\delta'$  for all lists that represent reachable sets can be done using the following algorithm. We will not give a full correctness proof for this algorithm. Informally, the invariant of the main repetition is that  $\delta'(Bs, Cs, \sigma)$  has been computed for all pairs  $Bs, Cs$  in  $Q_{closed}$  and all symbols  $\sigma \in \Sigma$ .

```

 $Q_{opened} := \{[\mathcal{E}_{m_1,0}, \mathcal{E}_{0,m_2}]\}$ ;
 $Q_{closed} := \emptyset$ ;
do  $Q_{opened} \neq \emptyset \rightarrow$ 
   $Bs : Bs \in Q_{opened}$ ;
   $Q_{new} := \emptyset$ ;
  for  $\sigma : \sigma \in \Sigma \rightarrow$ 
    “compute  $\delta'(Bs, Bs, \sigma)$ ”;
     $Q_{new} := Q_{new} \cup \{\delta'(Bs, Bs, \sigma)\}$ ;
    for  $Cs : Cs \in Q_{closed} \rightarrow$ 
      “compute  $\delta'(Bs, Cs, \sigma)$ ”;
       $Q_{new} := Q_{new} \cup \{\delta'(Bs, Cs, \sigma)\}$ ;
      “compute  $\delta'(Cs, Bs, \sigma)$ ”;
       $Q_{new} := Q_{new} \cup \{\delta'(Cs, Bs, \sigma)\}$ 
    rof
  rof;
   $Q_{closed} := Q_{closed} \cup \{Bs\}$ ;
   $Q_{opened} := (Q_{opened} \cup Q_{new}) \setminus Q_{closed}$ 
od

```

Referring to [IN77, MP04, Pol04] for the definition of tessellation automata, we can say that this corresponds to precomputing a two-dimensional deterministic online tessellation automaton with the following properties:

- alphabet:  $\Sigma$ ;
- state set:  $Q_{closed}$ ;
- transition function:  $\delta'$ ;
- initial state:  $[\mathcal{E}_{m_1,0}, \mathcal{E}_{0,m_2}]$ ;
- set of final states:  $\{[P]\}$ . (Note that  $\text{suff}(P) \cap \text{pref}(P)$  is indeed a reachable set and therefore  $[P] \in Q_{closed}$ .)

The matrices that occur in the lists in these algorithms are all prefixes of  $P$ . If pattern  $P$  is known, we can represent a prefix of  $P$  uniquely by its size. This means that in an implementation, it is not necessary to store lists of matrices; lists of integer pairs suffice.

### 6.3.4 Computation of the “failure function”

As we have seen in section 6.3.2, we still need to find a way to (pre)compute the values of auxiliary function  $r : \text{pref}(P) \rightarrow \mathcal{L}(\text{pref}(P))$ , specified as follows, for all  $A \in \text{pref}(P)$ :

$$q(r(A)) = (\text{suff}(A) \cap \text{pref}(P)) \setminus \{A\}$$

$$OL(r(A))$$

First we will consider the cases where  $A$  is an empty matrix in isolation.

- $\text{row}(A) = 0 \wedge \text{col}(A) = 0$ :  $A = \mathcal{E}_{0,0}$ . Then we can derive:

$$\begin{aligned} & (\text{suff}(\mathcal{E}_{0,0}) \cap \text{pref}(P)) \setminus \{\mathcal{E}_{0,0}\} \\ = & \quad \{ \text{definition suff} \} \\ & (\{\mathcal{E}_{0,0}\} \cap \text{pref}(P)) \setminus \{\mathcal{E}_{0,0}\} \\ = & \quad \{ \text{set calculus} \} \\ & \emptyset \\ = & \quad \{ \text{definition } q \} \\ & q([\ ]) \end{aligned}$$

- $\text{row}(A) = 0 \wedge 0 < \text{col}(A)$ :  $A = \mathcal{E}_{0,\text{col}(A)}$ .

$$\begin{aligned} & (\text{suff}(\mathcal{E}_{0,\text{col}(A)}) \cap \text{pref}(P)) \setminus \{\mathcal{E}_{0,\text{col}(A)}\} \\ = & \quad \{ \text{definition suff, } \mathcal{E}S \} \\ & (\mathcal{E}S_{0,\text{col}(A)} \cap \text{pref}(P)) \setminus \{\mathcal{E}_{0,\text{col}(A)}\} \\ = & \quad \{ \text{col}(A) \leq m_1 \} \\ & \mathcal{E}S_{0,\text{col}(A)} \setminus \{\mathcal{E}_{0,\text{col}(A)}\} \\ = & \quad \{ \text{definition } \mathcal{E}S, \text{ set calculus, } 0 < \text{col}(A) \} \end{aligned}$$

$$\begin{aligned}
& \mathcal{E}S_{0, \text{col}(A)-1} \\
= & \{ \text{definition pref, } \mathcal{E}S; \text{col}(A) \leq m_1 \} \\
& \mathcal{E}S_{0, \text{col}(A)-1} \cap \text{pref}(P) \\
= & \{ \text{definition } q \} \\
& q([\mathcal{E}_{0, \text{col}(A)-1}])
\end{aligned}$$

- $0 < \text{row}(A) \wedge \text{col}(A) = 0$ . This is completely symmetrical to the previous case and gives us:  $(\text{suff}(A) \cap \text{pref}(P)) \setminus \{A\} = q([\mathcal{E}_{\text{row}(A)-1, 0}])$ .

The case where  $A$  is a nonempty matrix remains. That is,  $0 < \text{row}(A) \wedge 0 < \text{col}(A)$ .

Informally, we want to find a way to compute the list  $r(A)$ . This list consists of the maximal elements of the set  $(\text{suff}(A) \cap \text{pref}(P)) \setminus \{A\}$ , ordered by decreasing number of rows and increasing number of columns. The method we will describe consists of running a (bounded) linear search for each  $j : 0 \leq j \leq \text{row}(A)$ , to find the maximal element with exactly  $j$  rows, if such a maximal element exists.

We start with the following observation:

$$\begin{aligned}
& (\text{suff}(A) \cap \text{pref}(P)) \setminus \{A\} \\
= & \{ \text{property suff, set calculus, } 0 < \text{row}(A) \wedge 0 < \text{col}(A) \} \\
& (\{A\} \cap \text{pref}(P)) \setminus \{A\} \cup \\
& (\text{suff}(A[1 .. \text{row}(A)][0 .. \text{col}(A)]) \cap \text{pref}(P)) \setminus \{A\} \cup \\
& (\text{suff}(A[0 .. \text{row}(A)][1 .. \text{col}(A)]) \cap \text{pref}(P)) \setminus \{A\} \\
= & \{ \text{set calculus, definition suff} \} \\
& (\text{suff}(A[1 .. \text{row}(A)][0 .. \text{col}(A)]) \cap \text{pref}(P)) \cup \\
& (\text{suff}(A[0 .. \text{row}(A)][1 .. \text{col}(A)]) \cap \text{pref}(P)) \\
= & \{ \bullet \text{ introduction of } g \} \\
& g(\text{row}(A) - 1) \cup (\text{suff}(A[0 .. \text{row}(A)][1 .. \text{col}(A)]) \cap \text{pref}(P))
\end{aligned}$$

Here we introduce the following auxiliary function:

$$g(j) = \text{suff}(A[\text{row}(A) - j .. \text{row}(A)][0 .. \text{col}(A)]) \cap \text{pref}(P)$$

It may seem that at this point we could simply conclude that  $A[0 .. \text{row}(A)][1 .. \text{col}(A)]$  is the first element of  $r(A)$ . However, we know that  $r(A) \in \mathcal{L}(\text{pref}(P))$  and in general,  $A[0 .. \text{row}(A)][1 .. \text{col}(A)]$  does not need to be a prefix of  $P$ .

We now introduce variables  $Rs : \mathcal{L}(\text{pref}(P))$  and  $i_1 : \mathbb{N}$  and the following tail invariant:

$$P0 : q(r(A)) = q(Rs) \cup g(i_1)$$

We could initially establish  $P0$  by choosing  $Rs = []$  and  $i_1 = \text{row}(A)$ . However, it will be useful to consider the case of  $i_1 = \text{row}(A)$  separately. Therefore, we choose to initialise  $i_1 = \text{row}(A) - 1$ . The initialisation of  $Rs$  then consists of a repetition; we introduce integer variable  $i_2$  and the following tail invariant:

$$\begin{aligned}
Q0 : & q(r(A)) = g(\text{row}(A) - 1) \cup \\
& (\text{suff}(A[0 .. \text{row}(A)][\text{col}(A) - i_2 .. \text{col}(A)]) \cap \text{pref}(P))
\end{aligned}$$

As we have seen, we can initially choose  $i_2 = \text{col}(A) - 1$ . We can terminate the computation when  $A[0 \dots \text{row}(A)][\text{col}(A) - i_2 \dots \text{col}(A)] = P[0 \dots \text{row}(A)][0 \dots i_2]$ , because then the assignment  $Rs := [P[0 \dots \text{row}(A)][0 \dots i_2]]$  establishes  $P0$ . We know that a value of  $i_2$  exists for which this termination condition holds, namely:  $i_2 = 0$ .

When  $A[0 \dots \text{row}(A)][\text{col}(A) - i_2 \dots \text{col}(A)] \neq P[0 \dots \text{row}(A)][0 \dots i_2]$ , we get:

$$\begin{aligned}
& g(\text{row}(A) - 1) \cup (\text{suff}(A[0 \dots \text{row}(A)][\text{col}(A) - i_2 \dots \text{col}(A)]) \cap \text{pref}(P)) \\
= & \{ 0 < \text{row}(A), 0 < i_2 \} \\
& g(\text{row}(A) - 1) \cup (\{A[0 \dots \text{row}(A)][\text{col}(A) - i_2 \dots \text{col}(A)]\} \cap \text{pref}(P)) \cup \\
& (\text{suff}(A[1 \dots \text{row}(A)][\text{col}(A) - i_2 \dots \text{col}(A)]) \cap \text{pref}(P)) \cup \\
& (\text{suff}(A[0 \dots \text{row}(A)][\text{col}(A) - i_2 + 1 \dots \text{col}(A)]) \cap \text{pref}(P)) \\
= & \{ A[0 \dots \text{row}(A)][\text{col}(A) - i_2 \dots \text{col}(A)] \neq P[0 \dots \text{row}(A)][0 \dots i_2] \} \\
& g(\text{row}(A) - 1) \cup (\text{suff}(A[1 \dots \text{row}(A)][\text{col}(A) - i_2 \dots \text{col}(A)]) \cap \text{pref}(P)) \cup \\
& (\text{suff}(A[0 \dots \text{row}(A)][\text{col}(A) - i_2 + 1 \dots \text{col}(A)]) \cap \text{pref}(P)) \\
= & \{ \text{definition } g \} \\
& g(\text{row}(A) - 1) \cup (\text{suff}(A[0 \dots \text{row}(A)][\text{col}(A) - i_2 + 1 \dots \text{col}(A)]) \cap \text{pref}(P))
\end{aligned}$$

So the initialisation step of our algorithm now becomes:

```

i2 := col(A) - 1;
{ invariant: Q0 }
do A[0 .. row(A)] [col(A) - i2 .. col(A)] ≠ P[0 .. row(A)] [0 .. i2] →
    i2 := i2 - 1
od;
R' := P[0 .. row(A)] [0 .. i2];
Rs := [R'];
i1 := row(A) - 1
{ P0 }

```

As we can see, we initialise  $Rs$  as a list containing one element,  $R'$ . This list will only grow in the rest of the algorithm text; it will never be an empty list. As an extra invariant we ensure that  $R'$  is always the last element of  $Rs$ :

$$P1 : \langle \exists Rs' : Rs' \in \mathcal{L}(\text{pref}(P)) : Rs = Rs' \triangleleft R' \rangle$$

Additionally, we also have as an invariant:

$$P2 : i_1 < \text{row}(R')$$

Now we can see that we can terminate the computation when  $\text{col}(A) \leq \text{col}(R')$ :

$$\begin{aligned}
& q(\text{row}(A)) \\
= & \{ P0 \} \\
& q(Rs) \cup g(i_1) \\
= & \{ \text{definition } g \}
\end{aligned}$$

$$\begin{aligned}
& q(Rs) \cup (\text{suff}(A[\text{row}(A) - i_1 \dots \text{row}(A)][0 \dots \text{col}(A)]) \cap \text{pref}(P)) \\
\subseteq & \{ \text{col}(A) \leq \text{col}(R') \} \\
& q(Rs) \cup (\text{suff}(A[\text{row}(A) - i_1 \dots \text{row}(A)][\text{col}(A) - \text{col}(R') \dots \text{col}(A)]) \cap \text{pref}(P)) \\
\subseteq & \{ P2 \} \\
& q(Rs) \cup (\text{suff}(A[\text{row}(A) - \text{row}(R') \dots \text{row}(A)][\text{col}(A) - \text{col}(R') \dots \text{col}(A)]) \cap \text{pref}(P)) \\
= & \{ R' \in \text{suff}(A) \} \\
& q(Rs) \cup (\text{suff}(R') \cap \text{pref}(P)) \\
= & \{ P1; (8) \text{ on page 60} \} \\
& q(Rs)
\end{aligned}$$

Invariant  $P0$  tells us  $q(Rs) \subseteq q(r(A))$ , so we have  $q(Rs) = q(r(A))$ . And since  $q(r(A))$  is the unique list corresponding to  $r(A)$  (as we have seen in section 6.3.0), we can then conclude:  $Rs = r(A)$ .

All that remains is to see how to update  $Rs$  and  $i_1$ . For this we again introduce a repetition, much like the one used for initialisation of  $i_1$  and  $Rs$ , with the following invariant:

$$\begin{aligned}
R0 : \quad & q(r(A)) = q(Rs) \cup g(i_1 - 1) \cup \\
& (\text{suff}(A[\text{row}(A) - i_1 \dots \text{row}(A)][\text{col}(A) - i_2 \dots \text{col}(A)]) \cap \text{pref}(P))
\end{aligned}$$

Initially we choose  $i_2 = \text{col}(A)$  and, like in the initialisation of  $i_1$  and  $Rs$ , we can stop the computation when we find  $A[\text{row}(A) - i_1 \dots \text{row}(A)][\text{col}(A) - i_2 \dots \text{col}(A)] = P[0 \dots i_1][0 \dots i_2]$ . However, we have an additional bound; we should also stop the computation when  $i_2 \leq \text{col}(R')$ :

$$\begin{aligned}
& q(r(A)) \\
= & \{ R0 \} \\
& q(Rs) \cup g(i_1 - 1) \cup (\text{suff}(A[\text{row}(A) - i_1 \dots \text{row}(A)][\text{col}(A) - i_2 \dots \text{col}(A)]) \cap \text{pref}(P)) \\
\subseteq & \{ i_2 \leq \text{col}(R') \} \\
& q(Rs) \cup g(i_1 - 1) \cup (\text{suff}(A[\text{row}(A) - i_1 \dots \text{row}(A)][\text{col}(A) - \text{col}(R') \dots \text{col}(A)]) \cap \text{pref}(P)) \\
\subseteq & \{ P2 \} \\
& q(Rs) \cup g(i_1 - 1) \cup (\text{suff}(A[\text{row}(A) - \text{row}(R') \dots \text{row}(A)][\text{col}(A) - \text{col}(R') \dots \text{col}(A)]) \cap \text{pref}(P)) \\
= & \{ R' \in \text{suff}(A) \} \\
& q(Rs) \cup g(i_1 - 1) \cup (\text{suff}(R') \cap \text{pref}(P)) \\
= & \{ P1; (8) \text{ on page 60} \} \\
& q(Rs) \cup g(i_1 - 1)
\end{aligned}$$

All this gives rise to the following algorithm to compute  $r(A)$ :

```

if row(A) = 0  $\wedge$  col(A) = 0  $\rightarrow$ 
  Rs := []
|| row(A) = 0  $\wedge$  0 < col(A)  $\rightarrow$ 
  Rs := [ $\mathcal{E}_{0, \text{col}(A)-1}$ ]
|| 0 < row(A)  $\wedge$  col(A) = 0  $\rightarrow$ 
  Rs := [ $\mathcal{E}_{\text{row}(A)-1, 0}$ ]
|| 0 < row(A)  $\wedge$  0 < col(A)  $\rightarrow$ 
   $i_2 := \text{col}(A) - 1$ ;

```

```

do  $A[0 \dots \text{row}(A)][\text{col}(A) - i_2 \dots \text{col}(A)] \neq P[0 \dots \text{row}(A)][0 \dots i_2] \rightarrow$ 
     $i_2 := i_2 - 1$ 
od;
 $R' := P[0 \dots \text{row}(A)][0 \dots i_2]$ ;
 $Rs := [R']$ ;
 $i_1 := \text{row}(A) - 1$ ;
do  $\text{col}(R') < \text{col}(A) \rightarrow$ 
     $i_2 := \text{col}(A)$ ;
    do  $\text{col}(R') < i_2 \wedge A[\text{row}(A) - i_1 \dots \text{row}(A)][\text{col}(A) - i_2 \dots \text{col}(A)] \neq P[0 \dots i_1][0 \dots i_2] \rightarrow$ 
         $i_2 := i_2 - 1$ 
    od;
    if  $\text{col}(r') < i_2 \rightarrow$ 
         $R' := P[0 \dots i_1][0 \dots i_2]$ ;
         $Rs := Rs \triangleleft R'$ 
        ||  $i_2 \leq \text{col}(R') \rightarrow$ 
            skip
    fi;
     $i_1 := i_1 + 1$ 
od
fi

```

## 6.4 Entire algorithm

In section 6.2 we have presented a preliminary version of Polcar's algorithm, expressed in terms of sets of prefixes of  $P$ . To make use of function  $\delta'$ , precomputed as described in section 6.3, these sets will need to be represented by the ordered lists of their maximal elements. To replace array  $e$ , we introduce array  $e' : [0 \dots n_2]$  of  $\mathcal{L}(\text{pref}(P))$ , with  $q(e'[j]) = e[j]$  and  $OL(e'[j])$  (for  $0 \leq j \leq n_2$ ). Furthermore, we note:

$$\begin{aligned}
 & P \in q(e'[j]) \\
 \equiv & \quad \{ OL(e'[j]); e' \in \mathcal{L}(\text{pref}(P)) \} \\
 & e'[j] = [P]
 \end{aligned}$$

We get the following algorithm:

```

 $O := \emptyset$ ;
for  $j : 0 \leq j \leq n_2 \rightarrow$   $e'[j] := [\mathcal{E}_{m_1,0}, \mathcal{E}_{0,m_2}]$  rof;
 $i_1, i_2 := 0, 0$ ;
do  $i_1 \neq n_1 \rightarrow$ 
    do  $i_2 \neq n_2 \rightarrow$ 
         $e'[i_2 + 1] := \delta'(e'[i_2 + 1], e'[i_2], T[i_1, i_2])$ ;
        if  $e'[i_2 + 1] = [P] \rightarrow O := O \cup \{(i_1 + 1 - m_1, i_2 + 1 - m_2)\}$ 
        ||  $e'[i_2 + 1] \neq [P] \rightarrow$  skip
        fi;
         $i_2 := i_2 + 1$ 
    od;
     $i_2 := 0$ ;
     $i_1 := i_1 + 1$ 
od

```

## 6.5 Remarks

There are a few differences between this version of the algorithm and the original presentations in [Pol04, MP04].

In the original presentations, a so-called (nondeterministic) two-dimensional online tessellation automaton (2OTA) is constructed, where each state in the 2OTA corresponds to a prefix of the pattern. This 2OTA is transformed into a two-dimensional *deterministic* online tessellation automaton (2DOTA), using an algorithm very similar to the subset construction.<sup>3</sup> As a result, the states in the resulting 2DOTA (implicitly) each correspond to a set of prefixes of  $P$ .

In our presentation, we do not need to introduce tessellation automata. Nor do we first construct an equivalent of the 2OTA; we immediately derive an algorithm to precompute the values of  $\delta'$ , which is the equivalent of the 2DOTA's transition function.

As we have seen in section 6.3.0, we do start out our derivation of the “transition function”  $\delta'$  by examining sets of prefixes of  $P$ , but we represent these sets by a list of their maximal elements. This approach improves the precomputation's performance and is more similar to the one-dimensional pattern matching algorithms which Polcar's algorithm is based on: Aho-Corasick and Knuth-Morris-Pratt ([AC75, KMP77]).

Even with these improvements, the precomputation for this algorithm can be very unefficient. However, the precomputation's performance depends only on the size of the pattern. As a result, Polcar's algorithm is most useful when the pattern is relatively very small, compared to the text.

---

<sup>3</sup>The subset construction is an algorithm used for transforming a (one-dimensional) nondeterministic finite automaton (NFA) into a deterministic finite automaton (DFA). See [ASU96], pages 117–121.



## 7 Conclusions and future work

### 7.0 Conclusions

We have described several, very different, two-dimensional pattern matching algorithms. Some are very well-known, such as the Baker-Bird algorithm ([Bak78, Bir77]), the first known solution to the problem. On the other hand we have also described the very recent Polcar algorithm ([Pol04, MP04]), which was first presented in 2004.

All of these algorithms have been formally derived. The derivations are formal proofs that these algorithms are correct solutions to the two-dimensional pattern matching problem, while their presentations in existing literature usually lack such full correctness proofs. The derivations also show where we have taken some of the major design decisions, such as the choice of a filter function in the filter-based algorithms (see section 4.0 on page 20). This can be very useful in discovering new two-dimensional pattern matching algorithms.

We have described the similarities between the Baker-Bird and Takaoka-Zhu ([TZ89, TZ94]) algorithms, by presenting both using a uniform description, as so-called filter-based algorithms. Because of this it became immediately obvious that a space complexity improvement, first suggested in the original presentation of Takaoka-Zhu, was also applicable to the Baker-Bird algorithm (and almost any other possible filter-based algorithm; see section 4.2 on page 22).

We have derived Baeza-Yates and Régnier’s algorithm ([BYR93]) and also proposed two improvements over its original presentation (section 5.2 on page 36 and section 5.3 on page 39). Both of these improvements use information obtained during a failed matching attempt to speed up the computation by skipping unnecessary row comparisons.

Polcar’s algorithm was derived without using the tessellation automata data structure or any of its properties. In addition, we have shown an improvement in the precomputation step of this algorithm: where Polcar’s approach uses sets of matrices, we have represented these sets by lists of the maximal elements of these sets (see section 6.3 on page 56). This is an improvement of both the time and space complexity of the precomputation. It is also an interesting improvement because it is so similar to the approach of the one-dimensional Aho-Corasick and Knuth-Morris-Pratt algorithms ([AC75, KMP77]): there too we reason in terms of strings of maximal length, as opposed to sets of strings (see also [WZ92, WZ93]).

Part of the original goal of this research was to create a taxonomy of two-dimensional pattern matching algorithms. We have not done this yet, because the algorithms we have seen differ greatly from each other, with the exception of the filter-based algorithms: Baker-Bird and Takaoka-Zhu. Given these great differences a taxonomy would only have a very coarse structure, not giving any additional information or insight.

### 7.1 Future work

Although we have derived several two-dimensional pattern matching algorithms, there are still known algorithms that have not been described in this thesis. Future research can include the derivation of more known algorithms. Also, there may be many solutions to the two-dimensional pattern matching problem that have not been discovered yet. In particular, exploring other choices for the filter function in the filter-based approach (see section 4 on page 20) may lead to new solutions. When more algorithms have been derived, constructing a taxonomy will be useful.

Existing pattern matching toolkits, such as SPARE Time and SPARE Parts (see [Cle03]), can be expanded to include the two-dimensional pattern matching algorithms described in this thesis.

However, matrices are a rather different data structure than strings, so generalising existing toolkits to include two-dimensional pattern matching strategies may be difficult; in that case, a separate toolkit for two-dimensional pattern matching can be developed, in the same style as SPARE Time and SPARE Parts.

Once an efficient and practical implementation has been constructed, it is possible to perform a benchmarking; that is, a thorough practical performance analysis. A complete theoretical performance analysis (of memory and time complexity) can be done for the algorithms described here as well. We can refer to the original presentations of the algorithms for their space and time complexity; however, it is unknown how the improvements made to the Baeza-Yates and Régner algorithm and the Polcar precomputation algorithm affect their average-case theoretical performance exactly.

Further generalisations of the two-dimensional pattern matching problem are also possible and worth exploring. We have already briefly discussed generalisation to multipattern matching and matching in more than two dimensions for most algorithms. Other generalisations include approximate two-dimensional pattern matching (where we need to define a distance function in terms of matrices) and matching patterns of non-rectangular shapes. (This last generalisation can be solved using the algorithms presented here, by introducing a “don’t care” symbol, which matches with every symbol of the text, and expanding the pattern to a bounding rectangle with this symbol. However, more efficient solutions may be possible.)

## A Properties of div and mod

**Definition A.0** For  $a \in \mathbb{N}$  and  $q \in \mathbb{N}^+$  we define **div** and **mod** by:

$$\begin{aligned} a &= (a \mathbf{div} q) * q + a \mathbf{mod} q \\ 0 &\leq a \mathbf{mod} q < q \end{aligned}$$

**Theorem A.1** For all  $a, b \in \mathbb{N}$  and  $q \in \mathbb{N}^+$  we have:

$$(a * q + b) \mathbf{mod} q = b \mathbf{mod} q$$

**Proof**

$$\begin{aligned} &(a * q + b) \mathbf{mod} q \\ = &\{ \text{def. } \mathbf{div} \text{ and } \mathbf{mod} : b = (b \mathbf{div} q) * q + b \mathbf{mod} q \} \\ &(a * q + (b \mathbf{div} q) * q + b \mathbf{mod} q) \mathbf{mod} q \\ = &\{ * \text{ over } + \} \\ &((a + b \mathbf{div} q) * q + b \mathbf{mod} q) \mathbf{mod} q \\ = &\{ 0 \leq b \mathbf{mod} q < q; \text{ def. } \mathbf{div} \text{ and } \mathbf{mod} : X = Y * q + Z \wedge 0 \leq Z < q \Rightarrow Z = X \mathbf{mod} q \} \\ &b \mathbf{mod} q \end{aligned}$$

□

**Theorem A.2** For all  $a, b, c \in \mathbb{N}$  and  $q \in \mathbb{N}^+$  we have:

$$(a * b + c) \mathbf{mod} q = ((a \mathbf{mod} q) * b + c) \mathbf{mod} q$$

**Proof**

$$\begin{aligned} &(a * b + c) \mathbf{mod} q \\ = &\{ a = (a \mathbf{div} q) * q + a \mathbf{mod} q \} \\ &(((a \mathbf{div} q) * q + a \mathbf{mod} q) * b + c) \mathbf{mod} q \\ = &\{ * \text{ over } + \} \\ &((a \mathbf{div} q) * q * b + (a \mathbf{mod} q) * b + c) \mathbf{mod} q \\ = &\{ \text{theorem A.1} \} \\ &((a \mathbf{mod} q) * b + c) \mathbf{mod} q \end{aligned}$$

□

**Theorem A.3** For  $a \in \mathbb{N}$  and  $q \in \mathbb{N}^+$ :

$$a \leq (a \mathbf{div} q + 1) * q - 1 < a + q$$

## Proof

First we note:

$$(a \mathbf{div} q + 1) * q - 1 = (a \mathbf{div} q) * q + q - 1$$

Now we can prove the theorem with the following derivation:

$$\begin{aligned} & a \\ = & \{ \text{definition } \mathbf{div} \text{ and } \mathbf{mod} \} \\ & (a \mathbf{div} q) * q + a \mathbf{mod} q \\ \leq & \{ a \mathbf{mod} q < q \} \\ & (a \mathbf{div} q) * q + q - 1 \\ \leq & \{ 0 \leq a \mathbf{mod} q \} \\ & (a \mathbf{div} q) * q + a \mathbf{mod} q + q - 1 \\ = & \{ \text{definition } \mathbf{div} \text{ and } \mathbf{mod} \} \\ & a + q - 1 \\ < & \{ \text{math} \} \\ & a + q \end{aligned}$$

□

## B Properties of **pref** and **suff** (for strings)

**Theorem B.0** For  $x, y \in \Sigma^*$ :

$$\text{suff}(xy) = \text{suff}(x)y \cup \text{suff}(y)$$

**Proof** In the following derivation, bound variables  $v$  and  $w$  are both strings over  $\Sigma$ . We will omit  $v, w \in \Sigma^*$  in our set quantifications.

$$\begin{aligned}
& \text{suff}(x)y \cup \text{suff}(y) \\
= & \{ \text{suff, twice} \} \\
& \langle \text{set } v, w : vw = x : w \rangle y \cup \langle \text{set } v, w : vw = y : w \rangle \\
= & \{ \text{concatenation over set} \} \\
& \langle \text{set } v, w : vw = x : wy \rangle \cup \langle \text{set } v, w : vw = y : w \rangle \\
= & \{ \text{dummy transformation, twice} \} \\
& \langle \text{set } v, w : |y| \leq |w| \wedge vw = xy : w \rangle \cup \langle \text{set } v, w : |w| \leq |y| \wedge vw = xy : w \rangle \\
= & \{ \text{set calculus (note overlap of } |w| = |y|) \} \\
& \langle \text{set } v, w : vw = xy : w \rangle \\
= & \{ \text{suff} \} \\
& \text{suff}(xy)
\end{aligned}$$

□

**Theorem B.1** For  $x, y \in \Sigma^*$  and  $a \in \Sigma$ :

$$\text{suff}(xay) = \text{suff}(x)ay \cup \text{suff}(y)$$

**Proof**

$$\begin{aligned}
& \text{suff}(xay) \\
= & \{ \text{theorem B.0} \} \\
& \text{suff}(xa)y \cup \text{suff}(y) \\
= & \{ \text{property suff} \} \\
& \text{suff}(x)ay \cup \{y\} \cup \text{suff}(y) \\
= & \{ y \in \text{suff}(y) \} \\
& \text{suff}(x)ay \cup \text{suff}(y)
\end{aligned}$$

□



## C Properties of **pref** and **suff** (for matrices)

**Theorem C.0** For matrix  $A$  and integers  $i_1$  and  $i_2$ , with  $i_1 \leq \text{row}(A)$  and  $i_2 \leq \text{col}(A)$ , we have:

$$\mathcal{ES}_{i_1, i_2} \subseteq \text{pref}(A)$$

and:

$$\mathcal{ES}_{i_1, i_2} \subseteq \text{suff}(A)$$

**Proof**

$$\begin{aligned}
& \mathcal{ES}_{i_1, i_2} \\
= & \{ \text{def. } \mathcal{ES} \} \\
& \langle \text{set } j_1, j_2 : 0 \leq j_1 \leq i_1 \wedge 0 \leq j_2 \leq i_2 \wedge (j_1 = 0 \vee j_2 = 0) : \mathcal{E}_{j_1, j_2} \rangle \\
= & \{ i_1 \leq \text{row}(A), i_2 \leq \text{col}(A), \text{def. submatrix} \} \\
& \langle \text{set } j_1, j_2 : 0 \leq j_1 \leq i_1 \wedge 0 \leq j_2 \leq i_2 \wedge (j_1 = 0 \vee j_2 = 0) : A[0 .. j_1][0 .. j_2] \rangle \\
\subseteq & \{ \text{set calculus} \} \\
& \langle \text{set } j_1, j_2 : 0 \leq j_1 \leq i_1 \wedge 0 \leq j_2 \leq i_2 : A[0 .. j_1][0 .. j_2] \rangle \\
\subseteq & \{ \text{set calculus, } i_1 \leq \text{row}(A), i_2 \leq \text{col}(A) \} \\
& \langle \text{set } j_1, j_2 : 0 \leq j_1 \leq \text{row}(A) \wedge 0 \leq j_2 \leq \text{col}(A) : A[0 .. j_1][0 .. j_2] \rangle \\
= & \{ \text{def. pref} \} \\
& \text{pref}(A) \\
\\
& \mathcal{ES}_{i_1, i_2} \\
= & \{ \text{def. } \mathcal{ES} \} \\
& \langle \text{set } j_1, j_2 : 0 \leq j_1 \leq i_1 \wedge 0 \leq j_2 \leq i_2 \wedge (j_1 = 0 \vee j_2 = 0) : \mathcal{E}_{j_1, j_2} \rangle \\
= & \{ i_1 \leq \text{row}(A), i_2 \leq \text{col}(A), \text{def. submatrix} \} \\
& \langle \text{set } j_1, j_2 : 0 \leq j_1 \leq i_1 \wedge 0 \leq j_2 \leq i_2 \wedge (j_1 = 0 \vee j_2 = 0) : \\
& \quad A[\text{row}(A) - j_1 .. \text{row}(A)][\text{col}(A) - j_2 .. \text{col}(A)] \rangle \\
\subseteq & \{ \text{set calculus} \} \\
& \langle \text{set } j_1, j_2 : 0 \leq j_1 \leq i_1 \wedge 0 \leq j_2 \leq i_2 : A[\text{row}(A) - j_1 .. \text{row}(A)][\text{col}(A) - j_2 .. \text{col}(A)] \rangle \\
\subseteq & \{ \text{set calculus, } i_1 \leq \text{row}(A), i_2 \leq \text{col}(A) \} \\
& \langle \text{set } j_1, j_2 : 0 \leq j_1 \leq \text{row}(A) \wedge 0 \leq j_2 \leq \text{col}(A) : \\
& \quad A[\text{row}(A) - j_1 .. \text{row}(A)][\text{col}(A) - j_2 .. \text{col}(A)] \rangle \\
= & \{ \text{dummy transformation: } j_1 := \text{row}(A) - j_1, j_2 := \text{col}(A) - j_2 \} \\
& \langle \text{set } j_1, j_2 : 0 \leq j_1 \leq \text{row}(A) \wedge 0 \leq j_2 \leq \text{col}(A) : A[j_1 .. \text{row}(A)][j_2 .. \text{col}(A)] \rangle \\
= & \{ \text{def. suff} \} \\
& \text{suff}(A)
\end{aligned}$$

□

**Theorem C.1** For all  $A \in \mathcal{M}_2(\Sigma)$  and  $k_1, k_2 \in \mathbb{N}$ :

$$\mathcal{E}S_{k_1, k_2} \cap \text{pref}(A) = \mathcal{E}S_{k_1 \downarrow \text{row}(A), k_2 \downarrow \text{col}(A)}$$

**Proof**

$$\begin{aligned}
& \mathcal{E}S_{k_1, k_2} \cap \text{pref}(A) \\
= & \{ \text{def. } \mathcal{E}S, \text{ def. pref} \} \\
& (\langle \text{set } j : 0 \leq j \leq k_1 : \mathcal{E}_{j,0} \rangle \cup \langle \text{set } j : 0 \leq j \leq k_2 : \mathcal{E}_{0,j} \rangle) \cap \\
& \langle \text{set } i_1, i_2 : 0 \leq i_1 \leq \text{row}(A) \wedge 0 \leq i_2 \leq \text{col}(A) : A[0 .. i_1][0 .. i_2] \rangle \\
= & \{ \cap \text{ over } \cup, \text{ set calculus} \} \\
& \langle \text{set } j, i_1, i_2 : 0 \leq j \leq k_1 \wedge 0 \leq i_1 \leq \text{row}(A) \wedge 0 \leq i_2 \leq \text{col}(A) \wedge A[0 .. i_1][0 .. i_2] = \mathcal{E}_{j,0} : \mathcal{E}_{j,0} \rangle \cup \\
& \langle \text{set } j, i_1, i_2 : 0 \leq j \leq k_2 \wedge 0 \leq i_1 \leq \text{row}(A) \wedge 0 \leq i_2 \leq \text{col}(A) \wedge A[0 .. i_1][0 .. i_2] = \mathcal{E}_{0,j} : \mathcal{E}_{0,j} \rangle \\
= & \{ \text{def. submatrix, math} \} \\
& \langle \text{set } j : 0 \leq j \leq k_1 \downarrow \text{row}(A) : \mathcal{E}_{j,0} \rangle \cup \langle \text{set } j : 0 \leq j \leq k_2 \downarrow \text{col}(A) : \mathcal{E}_{0,j} \rangle \\
= & \{ \text{def. } \mathcal{E}S \} \\
& \mathcal{E}S_{k_1 \downarrow \text{row}(A), k_2 \downarrow \text{col}(A)}
\end{aligned}$$

□

## D Lists

Here we will briefly describe the notation we use for lists.

- $[]$ : the empty list.
- $As ++ Bs$ : the concatenation of lists  $As$  and  $Bs$ .
- $A \triangleright As$ : the list which consists of element  $A$ , followed by  $As$ ; equivalently:  $[A] ++ As$ . In other terminology,  $A$  is the *head* of the list and  $As$  the *tail*.
- $As \triangleleft A$ : the list which consists of  $As$ , followed by element  $A$ ; equivalently:  $As ++ [A]$ .



## 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.
- [ASU96] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: principles, techniques and tools*. Addison-Wesley, 1996.
- [Bak78] Theodore P. Baker. A technique for extending rapid exact-match string matching to arrays of more than one dimension. *SIAM Journal on Computing*, 7(4):533–541, 1978.
- [Bir77] Richard Bird. Two dimensional pattern matching. *Information Processing Letters*, 6(5):168–170, 1977.
- [BYR93] Ricardo Baeza-Yates and Mireille Régner. Fast two-dimensional pattern matching. *Information Processing Letters*, 45(1):51–57, 1993.
- [Cle03] Loek Cleophas. Towards SPARE Time. Master’s thesis, Technische Universiteit Eindhoven, August 2003.
- [CR02] Maxime Crochemore and Wojciech Rytter. *Jewels of stringology*. World Scientific, 2002.
- [IN77] Katsushi Inoue and Akira Nakamura. Some properties of two-dimensional online tessellation acceptors. *Information sciences*, 43:169–184, 1977.
- [KMP77] Donald E. Knuth, James H. Morris, and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal of Computing*, 6(2):323–350, 1977.
- [KR87] Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, March 1987.
- [MP04] Bořivoj Melichar and Tomáš Polcar. A two-dimensional online tessellation automata approach to two-dimensional pattern matching. In Loek Cleophas and Bruce W. Watson, editors, *Proceedings of the Eindhoven FASTAR days 2004*. Computer science report 04/40, Technische Universiteit Eindhoven, December 2004.
- [MŽ05] Bořivoj Melichar and Jan Žďárek. On two-dimensional pattern matching by finite automata. In J. Farré, I. Litovsky, and S. Schmitz, editors, *CIAA 2005 pre-proceedings*, pages 185–194, 2005.
- [NR02] Gonzalo Navarro and Mathieu Raffinot. *Flexible pattern matching in strings*. Cambridge University Press, 2002.
- [Pol04] Tomáš Polcar. Two-dimensional pattern matching. Postgraduate study report DC-PSR-04-05, Czech Technical University, January 2004.
- [RS97] G. Rozenberg and A. Salomaa. *Handbook of formal languages*, volume 3: Beyond words, chapter 4: Two-dimensional languages, pages 215–267. Springer-Verlag, Berlin, 1997.
- [TZ89] Tadao Takaoka and Rui Feng Zhu. A technique for two-dimensional pattern matching. *Communications of the ACM*, 32(9):1110–1120, 1989.
- [TZ94] Tadao Takaoka and Rui Feng Zhu. A technique for two-dimensional pattern matching. In Jun-Ichi Aoe, editor, *String pattern matching strategies*, pages 220–230. IEEE Computer Society Press, 1994.
- [Wat95] Bruce W. Watson. *Taxonomies and toolkits of regular language algorithms*. PhD thesis, Technische Universiteit Eindhoven, 1995.

- [WZ92] Bruce W. Watson and Gerard Zwaan. A taxonomy of keyword pattern matching algorithms. Computing science report 92/27, Technische Universiteit Eindhoven, 1992.
- [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, 1993.
- [WZ95] Bruce W. Watson and Gerard Zwaan. A taxonomy of sublinear keyword pattern matching algorithms. Computing science report 95/13, Technische Universiteit Eindhoven, 1995.
- [WZ96] Bruce W. Watson and Gerard Zwaan. A taxonomy of sublinear multiple keyword pattern matching algorithms. *Science of Computer Programming*, 27(2):85–118, September 1996.