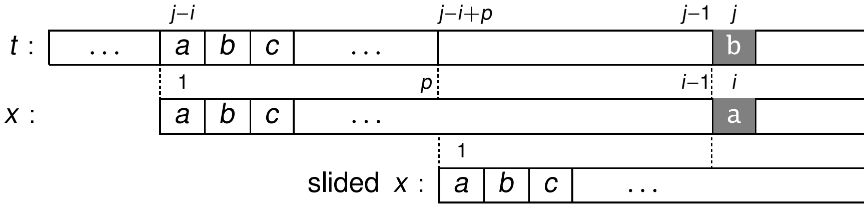
Morris-Pratt algorithm

The slowness of the naive algorithm in the worst case is due to the fact that, in case of failure, it starts again to compare the first letters of x etc. (see line 5), *without using the information of the partial success of the previous attempt.*

If the failure occurred at the index i in the word x and index j in the text, we have

$$x[1 \dots i-1] = t[j-i \dots j-1]$$
$$x[i] \neq t[j]$$

Morris-Pratt algorithm (cont)



Any subsequent search in the factor $t[j-i \dots j-1]$ could use the information that it is a prefix of x — the first comparison failure is coloured in grey.

If a new search starts in t at position $j-i+p$, with $2 \leqslant p \leqslant i-1$, it compares $x[1 \dots]$ to $t[j-i+p \dots]$. Since $t[j-i+p \dots j-1]$ is a suffix of $t[j-i \dots j-1]$, which is equal to $x[1 \dots i-1]$, it compares $x[1 \dots]$ to a suffix of $x[1 \dots i-1]$, i.e., to a part of itself!

Morris-Pratt algorithm (cont)

The important point here is that these comparisons are **independent** of the text t, since they apply to part of the pattern itself, i.e., the word x.

The strategy devised here is to compute all the comparisons on x *before* we start the search in the text, and then use this information to avoid repeating the same comparisons at different positions.

This **preprocessing** on x will indeed allow to recognise the sole configurations where the search may succeed.

Therefore the subproblem now is to determine the positions i where the word $x[1 \ldots i-1]$ ends with a prefix of x.

Morris-Pratt algorithm/Maximum borders

Let u be a non-empty word. A **border** of u is a word different from u which is both prefix and suffix of u. For example, the word $u = \texttt{abacaba}$ has the three borders ϵ, a and aba.

Let us note $\textsc{Border}(x)$ the **longest border** of a non-empty word x.

For a given word x, let us define a function β_x on all its prefixes as

$$\beta_x(i) = |\textsc{Border}(x[1 \ldots i])| \qquad 1 \leqslant i \leqslant |x|$$

or, equivalently

$$\beta_x(|y|) = |\textsc{Border}(y)| \qquad y \leqslant x$$

Morris-Pratt algorithm/Pattern preprocessing (cont)

Here is the table of the maximum borders for the prefixes of the word abacabac:

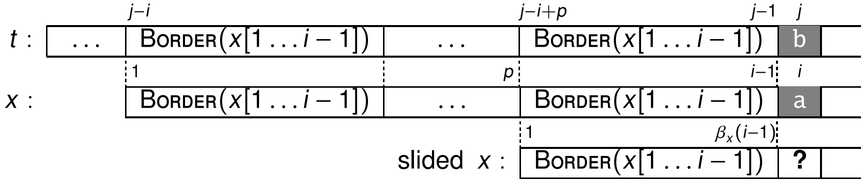
x	a	b	a	c	a	b	a	c
i	1	2	3	4	5	6	7	8
$\textsc{Border}(y)$	ϵ	ϵ	a	ϵ	a	ab	aba	abac
$\beta_x(i)$	0	0	1	0	1	2	3	4

Each row corresponds to a prefix y of x, not just to a letter. So, at index 7, you should read $\textsc{Border}(\underline{\text{aba}}\text{c}\underline{\text{aba}}) = \text{aba}$, at index 4, $\textsc{Border}(\text{abac}) = \varepsilon$ because $\text{abac} = \underline{\varepsilon}\text{abac}\underline{\varepsilon}$.

Also, note that borders can overlap, e.g.,. $\text{aaaa} = \underline{\text{aaa}}\text{a} = \text{a}\underline{\text{aaa}}$, so $\textsc{Border}(\text{aaaa}) = \text{aaa}$.

Morris-Pratt algorithm/Pattern preprocessing (cont)

Coming back to the naive search:



the difference here is that, in case of failure (in grey), we do **not** decrement the position in t and we slide x of $p = i - 1 - \beta_x(i - 1)$ positions with respect to t, i.e., the current position in x does not decrement to 1 but becomes $1 + \beta_x(i - 1)$.

Morris-Pratt algorithm/Pattern preprocessing (cont)

What happens in the extreme case $i = 1$, that is, when the comparison fails on the first letter of the pattern?

In this case, there is no optimisation left, we just can try our luck by comparing the next position in t to the first letter of x (again): this is exactly what the naive algorithm would do.

This means that the sliding offset must equal one when $i = 1$:

$$i - 1 - \beta_x(i - 1) = 1 \quad \text{where } i = 1$$

which is equivalent to

$$\beta_x(0) = -1$$

Therefore we need to extend β to be defined on 0 as $\beta_x(0) = -1$ for all words x.

Morris-Pratt algorithm/Pattern preprocessing (cont)

In practice, it is convenient to use a function $s_x : \{1, \ldots, |x|\} \to \{0, \ldots, |x|\}$ defined as

$$s_x(i) = 1 + \beta_x(i - 1) \qquad 1 \leqslant i \leqslant |x|$$

Then the sliding consists simply in replacing i by $s_x(i)$.

This function s_x is called the **failure function** of pattern x. For the word abacabac, it is

x		a	b	a	c	a	b	a	c
i	0	1	2	3	4	5	6	7	8
$\beta_x(i)$	−1	0	0	1	0	1	2	3	4
$s_x(i)$	$\square$	0	1	1	2	1	2	3	4

Morris-Pratt algorithm/Imperative pseudo-code

```
  MP($x, t$)
1 $s \leftarrow$ MP-FAIL($x$)                    ▷ The failure function as an array.
2 $i \leftarrow 1; j \leftarrow 1$
3 **while** $i \leqslant |x|$ **and** $j \leqslant |t|$
4      **do if** $i = 0$ **or** $x[i] = t[j]$       ▷ Test on $i$ because now $i$ can be 0.
5            **then** $i \leftarrow i + 1; j \leftarrow j + 1$   ▷ Schedule next letter in $x$.
6            **else** $i \leftarrow s[i]$          ▷ Failed: we slide $x$ on $t$.
7 **if** $|x| < i$
8    **then** ...                              ▷ Occurrence of $x$ in $t$ at $j - |x|$.
9    **else** ...                              ▷ No occurrences.
```

Morris-Pratt algorithm/Single-assignment pseudo-code

```
   MP($x, t$)
1  $s \leftarrow$ MP-FAIL($x$)
2  $(i, j) \leftarrow$ OFFSETS($s, x, t, 1, 1$)
3  if $|x| < i$
4     then ... else ...
```

```
   OFFSETS($s, x, t, i, j$)
1  if $i \leqslant |x|$ and $j \leqslant |t|$
2     then if $i = 0$ or $x[i] = t[j]$
3             then OFFSETS $\leftarrow$ OFFSETS($s, x, t, i + 1, j + 1$)
4             else OFFSETS $\leftarrow$ OFFSETS($s, x, t, s[i], j$)
5     else OFFSETS $\leftarrow (i, j)$
```

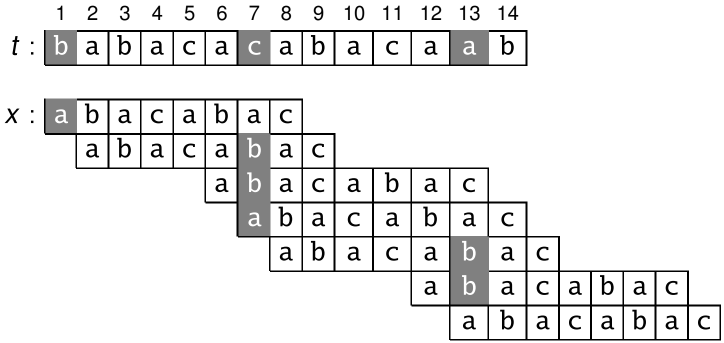
Morris-Pratt algorithm/Analysis and example

The Morris-Pratt algorithm finds an occurrence of a word x in a text t, or fail to do so, in at most $2|t| - 1$ letter comparisons, if we already have the failure function on x.

Example. Consider the following table for the search of the word $x = \mathtt{abacabac}$ in the text $t = \mathtt{babacacabacaab}$.

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	b	a	b	a	c	a	c	a	b	a	c	a	a	b
i	1	1	2	3	4	5	6	1	2	3	4	5	6	2
	0						2						2	
							1						1	
							0							

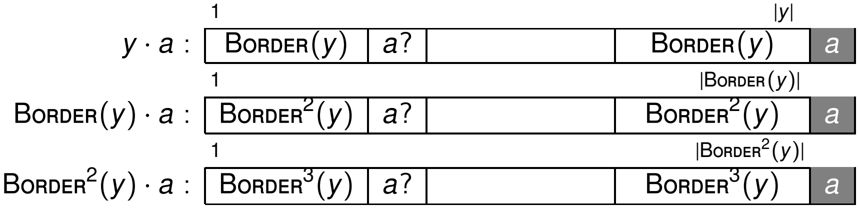
Morris-Pratt algorithm/Example



Morris-Pratt algorithm/Maximum borders (cont)

We need to show how to compute the function β or, equivalently, the failure function s. Both rely on the computation of maximum-length borders.

How do we compute the longest borders?



Morris-Pratt algorithm/Maximum borders (cont)

Let us find the longest border of a word ya, i.e., $\textsc{Border}(ya)$, where a is a letter.

The idea is to, recursively, consider $\textsc{Border}(y) \cdot a$.

If $\textsc{Border}(y) \cdot a$ is a prefix of y, then $\textsc{Border}(ya) = \textsc{Border}(y) \cdot a$.

Otherwise, we must consider the border of the border of y, i.e., $\textsc{Border}^2(y) \cdot a$, instead of $\textsc{Border}(y) \cdot a$ and repeat this process, until $\textsc{Border}^k(y) \cdot a$ is a prefix of y of $\textsc{Border}^k(y)$ is empty.

Morris-Pratt algorithm/Maximum borders (cont)

This can be formally summarised as follows. For all word $y \neq \epsilon$ and all letter a:

$$\begin{aligned}
\textsc{Border}(a) &= \epsilon \\
\textsc{Border}(y \cdot a) &= \begin{cases} \textsc{Border}(y) \cdot a & \text{if } \textsc{Border}(y) \cdot a \preccurlyeq y \\ \textsc{Border}(\textsc{Border}(y) \cdot a) & \text{otherwise} \end{cases}
\end{aligned}$$

For example, let $y = \mathtt{ababbb}$ and assume $\textsc{Border}(y) = \epsilon$.
Then $\textsc{Border}(y \cdot \mathtt{a}) = \textsc{Border}(y) \cdot \mathtt{a} = \mathtt{a}$ because $\mathtt{a} \preccurlyeq y$. Other examples:

$$\begin{array}{lll}
y = \mathtt{babbaa} & \textsc{Border}(y) = \epsilon & \textsc{Border}(y \cdot \mathtt{a}) = \textsc{Border}(\epsilon \cdot \mathtt{a}) = \epsilon \\
y = \mathtt{abaaab} & \textsc{Border}(y) = \mathtt{ab} & \textsc{Border}(y \cdot \mathtt{a}) = \textsc{Border}(y) \cdot \mathtt{a} = \mathtt{aba} \\
y = \mathtt{abbaab} & \textsc{Border}(y) = \mathtt{ab} & \textsc{Border}(y \cdot \mathtt{a}) = \textsc{Border}(\mathtt{ab} \cdot \mathtt{a}) = \mathtt{a}
\end{array}$$

Morris-Pratt algorithm/Maximum borders (cont)

Let us prove that, when we take the border of a word, we get a prefix of this word, i.e.,

$$\textsc{Border}(y) \prec y \qquad y \neq \varepsilon$$

First, the property holds on words of length 1 since, by definition,

$$\textsc{Border}(a) = \varepsilon \prec a$$

Assume now that the property holds for all words of length lower or equal to n and let us prove that it holds for words of length $n + 1$.

Morris-Pratt algorithm/Maximum borders (cont)

Let y be a word of length n, $1 \leqslant n$, and a letter a.

By definition, we have

$$\textsc{Border}(ya) = \begin{cases} \textsc{Border}(y) \cdot a & \text{if } \textsc{Border}(y) \cdot a \preccurlyeq y \\ \textsc{Border}(\textsc{Border}(y) \cdot a) & \text{otherwise} \end{cases}$$

So, if $\textsc{Border}(y) \cdot a \preccurlyeq y$, then

$$\textsc{Border}(ya) = \textsc{Border}(y) \cdot a \preccurlyeq y \prec ya$$

This is the property for words of length $n+1$ such that $\textsc{Border}(y) \cdot a \preccurlyeq y$.

Morris-Pratt algorithm/Maximum borders (cont)

Otherwise, if $\textsc{Border}(y) \cdot a \not\preccurlyeq y$ then

$$\textsc{Border}(ya) = \textsc{Border}(\textsc{Border}(y) \cdot a).$$

Since $\textsc{Border}(y) \prec y$ and $|y| = n$ then

$$\begin{aligned}|\textsc{Border}(y)| < |y| \Leftrightarrow |\textsc{Border}(y)| < n \Leftrightarrow & \; |\textsc{Border}(y)| + 1 < n + 1 \\ \Leftrightarrow & \; |\textsc{Border}(y)| + 1 \leqslant n\end{aligned}$$

Thus $|\textsc{Border}(y) \cdot a| = |\textsc{Border}(y)| + 1 \leqslant n$, and the inductive assumption holds:

$$\begin{aligned}\textsc{Border}(\textsc{Border}(y) \cdot a) & \prec \textsc{Border}(y) \cdot a \\ \textsc{Border}(ya) & \preccurlyeq \textsc{Border}(y) \prec y \prec ya\end{aligned}$$

This is the property for words of length $n + 1$ such that $\textsc{Border}(y) \cdot a \not\preccurlyeq y$, so this achieves the proof.

Morris-Pratt algorithm/Maximum borders (cont)

Let us try now the following.

$\textsc{Border}(ya)$

$$\begin{aligned}
&= \textsc{Border}(\textsc{Border}(y)\cdot a) && \textsc{Border}(y)\cdot a \not\leqslant y\\
&= \textsc{Border}(\textsc{Border}^2(y)\cdot a) && \textsc{Border}^2(y)\cdot a \not\leqslant \textsc{Border}(y)\\
&= \ldots && \ldots\\
&= \textsc{Border}(\textsc{Border}^{k-1}(y)\cdot a) && \textsc{Border}^{k-1}(y)\cdot a \not\leqslant \textsc{Border}^{k-2}(y)
\end{aligned}$$

and $\forall p \leqslant k-2.\textsc{Border}^p(y) \neq \varepsilon$.

Then there is a smallest k such that $\textsc{Border}^k(y)\cdot a \leqslant \textsc{Border}^{k-1}(y)$ or $\textsc{Border}^{k-1}(y) = \varepsilon$.

Morris-Pratt algorithm/Pattern preprocessing (resumed)

We proved page 29, that

$$\textsc{Border}(y) \prec y \qquad y \neq \varepsilon$$

Therefore

$$\textsc{Border}^{k}(y) \prec \textsc{Border}^{k-1}(y) \prec y \qquad 2 \leqslant k$$

Then,

$$\textsc{Border}^{k}(y) \cdot a \preccurlyeq \textsc{Border}^{k-1}(y) \Longleftrightarrow \textsc{Border}^{k}(y) \cdot a \preccurlyeq y \qquad 1 \leqslant k$$

(Make a picture to convince yourself.)

Morris-Pratt algorithm/Pattern preprocessing (resumed)

If $\textsc{Border}^k(y) \cdot a \preccurlyeq y$ and $\textsc{Border}^{k-1}(y) \neq \varepsilon$ then, using the definition

$$\textsc{Border}(ya) = \textsc{Border}(\textsc{Border}^{k-1}(y) \cdot a) = \textsc{Border}^k(y) \cdot a$$

Otherwise, if $\textsc{Border}^{k-1}(y) = \varepsilon$ then

$$\textsc{Border}(ya) = \textsc{Border}(\textsc{Border}^{k-1}(y) \cdot a) = \textsc{Border}(a) = \varepsilon$$

and we cannot have $\textsc{Border}^k(y) \cdot a \preccurlyeq y$ since $\textsc{Border}^k(y) = \textsc{Border}(\textsc{Border}^{k-1}(y))$ is then undefined (the empty word has no border). Finally, we can summarise all cases:

$$\textsc{Border}(ya) = \begin{cases} \textsc{Border}^k(y) \cdot a & \text{if } \textsc{Border}^k(y) \cdot a \preccurlyeq y \\ \varepsilon & \text{if } \textsc{Border}^{k-1}(y) = \varepsilon \end{cases}$$

Morris-Pratt algorithm/Pattern preprocessing (resumed)

Let us give a computational definition of function β_x. Let $ya \preccurlyeq x$.

$$\begin{aligned}\beta_x(|ya|) &= |\textsc{Border}(ya)| && \text{by substitution of } y \text{ by } ya \text{ in definition}\\ \beta_x(|y|+1) &= |\textsc{Border}(ya)| && \text{by property of length} \qquad (1)\end{aligned}$$

From the definition of β_x page 17, we deduce the corollary

$$\beta_x^k(|y|) = |\textsc{Border}^k(y)| \quad y \preccurlyeq x \qquad (2)$$

which we can prove by induction on k. The property is true for $k = 1$ (even $k = 0$), by definition. Then assume $\beta_x^k(|y|) = |\textsc{Border}^k(y)|$ and do

$$\begin{aligned}\beta_x^{k+1}(|y|) &= \beta_x^k(\beta_x(|y|)) = \beta_x^k(|\textsc{Border}(y)|) = |\textsc{Border}^k(\textsc{Border}(y))|\\ &= |\textsc{Border}^{k+1}(y)|\end{aligned}$$

which proves that the property holds for all k such that $1 \preccurlyeq k$.

Morris-Pratt algorithm/Pattern preprocessing (resumed)

For the sake of clarity, let us mimic the setting of figure page 19 by letting $|ya| = i$.

$\textsc{Border}^k(y) \cdot a \preccurlyeq y$

$$\begin{aligned}
&\iff y[|\textsc{Border}^k(y)| + 1] = a && \text{since } \textsc{Border}(y) \prec y\\
&\iff y[\beta_x^k(|y|) + 1] = a && \text{by property (2)}\\
&\iff x[\beta_x^k(|y|) + 1] = x[|y| + 1] && \text{since } ya \preccurlyeq x\\
&\iff x[\beta_x^k(i-1) + 1] = x[i] && \text{since } |y| = i - 1 \qquad (3)
\end{aligned}$$

Pattern preprocessing (cont)

Let us consider again the recursive definition of Border at page 34:

$$\textsc{Border}(ya) = \begin{cases} \textsc{Border}^k(y)\cdot a & \text{if } \textsc{Border}^k(y)\cdot a \preccurlyeq y \\ \varepsilon & \text{if } \textsc{Border}^{k-1}(y) = \varepsilon \end{cases}$$

where k is the smallest non-zero integer such that $\textsc{Border}^k(y)\cdot a \preccurlyeq y$ or $\textsc{Border}^{k-1}(y) = \varepsilon$.

By taking the lengths of each side of the equations, it comes

$$|\textsc{Border}(ya)| = \begin{cases} |\textsc{Border}^k(y)\cdot a| & \text{if } \textsc{Border}^k(y)\cdot a \preccurlyeq y \\ |\varepsilon| & \text{if } \textsc{Border}^{k-1}(y) = \varepsilon \end{cases}$$

Pattern preprocessing (cont)

By equation (1) we get

$$\beta_x(|y|+1) = \begin{cases} |\textsc{Border}^k(y)|+1 & \text{if } \textsc{Border}^k(y)\cdot a \preccurlyeq y \\ 0 & \text{if } \textsc{Border}^{k-1}(y) = \varepsilon \end{cases}$$

Using equations (2) and (3):

$$\beta_x(|y|+1) = \begin{cases} \beta_x^k(|y|)+1 & \text{if } x[\beta_x^k(i-1)+1] = x[i] \\ 0 & \text{if } \beta_x^{k-1}(i-1) = 0 \end{cases}$$

Pattern preprocessing (cont)

Finally, taking into account the value in 0 and 1, we have, for $|x| = i \geqslant 2$

$$\beta_x(0) = -1 \qquad \beta_x(1) = 0$$

$$\beta_x(i) = \begin{cases} 1 + \beta_x^k(i-1) & \text{if } x[1 + \beta_x^k(i-1)] = x[i] \\ 0 & \text{if } \beta_x^{k-1}(i-1) = 0 \end{cases}$$

where k is the smallest nonzero integer such that $x[1 + \beta_x^k(i-1)] = x[i]$ or $\beta_x^{k-1}(i-1) = 0$.

Pattern preprocessing (cont)

From that definition, it is clear that β returns the value -1 only on 0.

Thus

$$\begin{aligned}\beta_x^{k-1}(i-1) = 0 &\iff \beta_x(\beta_x^{k-1}(i-1)) = \beta_x(0) \iff \beta_x^k(i-1) = -1\\ &\iff 1 + \beta_x^k(i-1) = 0\end{aligned}$$

Then we can rewrite the previous definition of β_x as

$$\beta_x(0) = -1 \qquad \beta_x(i) = 1 + \beta_x^k(i-1) \quad 1 \leqslant i$$

where k is the smallest non-zero integer such that

- either $1 + \beta_x^k(i-1) = 0$
- or $x[1 + \beta_x^k(i-1)] = x[i]$

Pattern preprocessing/Imperative

It is then straightforward to write an algorithm for β_x, assuming $0 \leqslant i \leqslant |x|$:

```
   BETA($x, i$)
1  if $i = 0$
2     then BETA ← $-1$                                  ▷ $\beta_x(0) = -1$
3     else offset ← $i - 1$
4          repeat
5                offset ← BETA($x$, offset)
6            until offset $= -1$ or $x[1 + \textit{offset}] = x[i]$
7          BETA ← $1 + \textit{offset}$                 ▷ $\beta_x(i) = 1 + \beta_x^k(i-1)$
```

Pattern preprocessing/Single-assignment pseudo-code

```
BETA($x, i$)
1  if $i = 0$
2      then BETA $\leftarrow -1$
3      else BETA $\leftarrow 1 +$ OFFSET($x$, BETA($x, i - 1$))

OFFSET($x, offset$)
1  if $offset = -1$ or $x[1 + offset] = x[i]$
2      then OFFSET $\leftarrow offset$
3      else OFFSET $\leftarrow$ OFFSET($x$, BETA($x, offset$))
```

Pattern preprocessing (cont)

The problem here is inefficiency:

1. successive calls with the same arguments trigger again the same computations,
2. the recursive calls are expensive.

It is much preferable to have β as an array:

1. the values of β are computed only once each
2. and are accessed in constant time instead of relying on function calls.

Pattern preprocessing/Imperative

The idea is to compute all the values of $\beta_x(i)$ for increasing values of i:

```
    MAKE-BETA($x$)
1   $\beta_x[0] \leftarrow -1$
2   for $i \leftarrow 1$ to $|x|$
3       do $offset \leftarrow i - 1$
4          repeat
5                $offset \leftarrow \beta_x[offset]$
6            until $offset = -1$ or $x[1 + offset] = x[i]$
7          $\beta_x[i] \leftarrow 1 + offset$          $\triangleright \beta_x(i) = 1 + \beta_x^k(i - 1)$
8   MAKE-BETA $\leftarrow \beta_x$
```

Pattern preprocessing / Single-assignment pseudo-code

```
   MAKE-BETA($x$)
1  $\beta_x[0] \leftarrow -1$
2  **for** $i \leftarrow 1$ **to** $|x|$
3        **do** $\beta_x[i] \leftarrow 1 + \text{OFFSET}(x, \beta_x, \beta_x[i-1])$
4  MAKE-BETA $\leftarrow \beta_x$

   OFFSET($x, \beta_x, offset$)
1  **if** $offset = -1$ **or** $x[1 + offset] = x[i]$
2     **then** OFFSET $\leftarrow offset$
3     **else** OFFSET $\leftarrow \text{OFFSET}(x, \beta_x, \beta_x[offset])$
```

Pattern preprocessing (cont)

In order to finish, we need to give the algorithm for the failure function s_x (see page 21). Recall that

$$s_x(i) = 1 + \beta_x(i-1) \quad 1 \leqslant i$$

We could then keep Beta and create a separate function MP-Fail simply by sticking to the above definition:

```
   MP-Fail($\beta_x, i$)
1  MP-Fail $\leftarrow 1 + \beta_x[i-1]$        $\triangleright\ s_x[i] \leftarrow 1 + \beta_x[i-1]$
```

But, again, this is not efficient: it would be better to precompute and store all the values of MP-Fail in an array.

Pattern preprocessing (cont)

Instead, let us prove by induction on k that

$$s_x^k(i) = 1 + \beta_x^k(i-1) \quad 1 \leqslant i$$

The case $k = 1$ is the definition of s. Let us then assume that the property is true for all values from 1 to k. We have

$$s_x^{k+1}(i) = s_x^k(s_x(i)) = 1+\beta_x^k(s_x(i)-1) = 1+\beta_x^k(\beta_x(i-1)) = 1+\beta_x^{k+1}(i-1)$$

which proves the property for all $1 \leqslant k$.

Pattern preprocessing (cont)

Let us recall the definition of β_x (page 40):

$$\beta_x(0) = -1 \qquad \beta_x(i) = 1 + \beta_x^k(i-1) \qquad 1 \leqslant i$$

where k is the smallest non-zero integer such that

- either $1 + \beta_x^k(i-1) = 0$ or $x[1 + \beta_x^k(i-1)] = x[i]$

This becomes

$$s_x(1) = 0 \qquad s_x(1+i) = 1 + s_x^k(i) \qquad 1 \leqslant i$$

where k is the smallest non-zero integer such that

- either $s_x^k(i) = 0$ or $x[s_x^k(i)] = x[i]$

Pattern preprocessing/Imperative

```
MP-FAIL($x$)
$s_x[1] \leftarrow 0$
for $i \leftarrow 1$ to $|x| - 1$
    do $offset \leftarrow i$
        repeat
                $offset \leftarrow s_x[offset]$
          until $offset = 0$ or $x[offset] = x[i]$
        $s_x[1 + i] \leftarrow 1 + offset$
MP-FAIL $\leftarrow s$
```

Pattern preprocessing/Single-assignment pseudo-code

```
   MP-FAIL($x$)
1  $s_x[1] \leftarrow 0$
2  **for** $i \leftarrow 1$ **to** $|x| - 1$
3          **do** $s_x[1+i] \leftarrow 1 + \textsc{Offset}(x, s_x, s_x[i])$
4  MP-FAIL $\leftarrow s_x$

   OFFSET($x, s_x, offset$)
1  **if** $offset = 0$ **or** $x[offset] = x[i]$
2     **then** OFFSET $\leftarrow offset$
3     **else** OFFSET $\leftarrow \textsc{Offset}(x, s_x, s_x[offset])$
```

Analysis

The Morris-Pratt algorithm makes at most $2(|t| + |x| - 2)$ comparisons to find a word x in a text t or to fail.

This is much better than the naive algorithm whose cost was $(|t| - |x| + 1) \times |x|$.