# Efficient validation and construction of border arrays

J.-P. Duval    T. Lecroq    A. Lefebvre

Laboratoire d'Informatique, de Traitement de l'Information et des Systèmes
University of Rouen, France

JM 2006

30/08/2006 Rennes, France

Laboratoire d'Informatique, de Traitement de l'Information et des Systèmes
LITIS

UNIVERSITÉ DE ROUEN

# Outline

1. Recalls
2. New results
3. Conclusions and perspectives

# Outline

1. Recalls
2. New results
3. Conclusions and perspectives

# Border

**Definition**

A string $u$ is a border of a string $w$ if $u$ is both a prefix and a suffix of $w$ such that $u \neq w$.

**Definition**

The border of a string $w$ is the longest of its borders. It is denoted by $Border(w)$.

# Border array

**Definition**

Given a string $w[1..n]$ of length $n$, the array $f$ defined by

$$f[i] = |Border(w[1..i])|$$

for $1 \leq i \leq n$ is called the border array of $w$.

It constitutes the "failure function" of the Morris-Pratt (1970) string matching algorithm.

# Border array

## Example

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 12 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $w[i]$ | a | b | a | b | a | c | a | a | b | c | a | b | a | b | a |
| $f[i]$ | 0 | 0 | 1 | 2 | 3 | 0 | 1 | 1 | 2 | 0 | 1 | 2 | 3 | 4 | 5 |

# $\mathcal{D}(w)$

## Definition

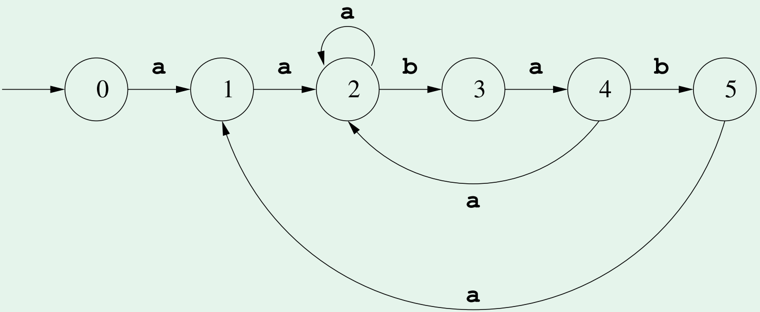
The DFA $\mathcal{D}(w)$ recognizing the language $\Sigma^* w$ is defined by $\mathcal{D}(w[1..n]) = (Q, \Sigma, q_0, T, F)$ where

- $Q = \{0, 1, \ldots, n\}$ is the set of states;
- $\Sigma$ is the alphabet;
- $q_0 = 0$ is the initial state;
- $T = \{n\}$ is the set of accepting states;
- $F = \{(i, w[i+1], i+1) \mid 1 \leq i \leq n\} \cup \{(i, a, |Border(w[1..i]a)|) \mid 0 \leq i < n$ and $a \in \Sigma \setminus \{w[i+1]\}\}$ is the set of transitions.

The underlying unlabeled graph is called the *skeleton* of the automaton.

# DFA

## Example




$\mathcal{D}$(aabab): transitions leading to state 0 are omitted.

# $\delta(i)$ and $\delta'(i)$

## Definition

For $0 \leq i \leq n$:

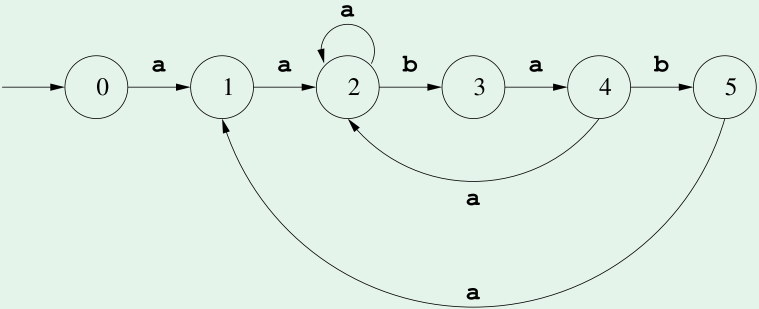
- $\delta(i) = (j \mid (i, a, j) \in F$ with $a \in A$ and $j \neq 0)$;
- $\delta'(i) = (j \mid (i, a, j) \in F$ with $a \in A$ and $j \notin \{0, i+1\})$.

In words:

- $\delta(i)$ is the list of the targets of the significant transitions leaving state $i$;
- $\delta'(i)$ is the list of the targets of the backward significant transitions leaving state $i$.

# DFA

## Example




$\mathcal{D}$(aabab): transitions leading to state 0 are omitted.

$$\delta(4) = (5, 2) \text{ and } \delta'(4) = (2).$$

# Complexity

**Theorem 1 [Simon 1993]**

There are at most $n$ significant backward transitions in $\mathcal{D}(w[1..n])$.

# Valid

**Definition**

An integer array $f[1..n]$ is a valid array (or is valid) if and only if it is the border array of at least one string $w[1..n]$.

# The main problems

### Validation

Given an integer array, is it valid? On which alphabet size?

### Construction of a string

Given a valid array, exhibit a string for which this array is the border array?

### Construction of border arrays

Construct all the distinct border arrays for some length.

# Motivations

Theoretical interest

Generating minimal test sets for various string algorithms

# Previous works


D. Moore, W. F. Smyth, and D. Miller.
Counting distinct strings.
*Algorithmica*, 23(1):1–13, 1999.

F. Franěk, S. Gao, W. Lu, P. J. Ryan, W. F. Smyth, Y. Sun, and L. Yang.
Verifying a border array in linear time.
*Journal on Combinatorial Mathematics and Combinatorial Computing*, 42:223–236, 2002.

J.-P. Duval, T. Lecroq, and A. Lefebvre.
Border array on bounded alphabet.
*Journal of Automata, Languages and Combinatorics*, 10(1):51–60, 2005.

# Previous works

**Web site**

http://al.jalix.org/Baba/Applet/baba.php

# The candidates

## Definition

For $1 \leq i \leq n$, we define

- $f^1[i] = f[i]$; and ,
- $f^\ell[i] = f[f^{\ell-1}[i]]$ for $f[i] > 0$;
- $C(f, i) = (1 + f[i-1], 1 + f^2[i-1], \ldots, 1 + f^m[i-1])$ where $f^m[i-1] = 0$.

# Validation

There are two necessary and sufficient conditions for an integer array $f$ to be valid:

1. $f[1] = 0$ and for $2 \leq i \leq n$, we have $f[i] \in (0) \uplus C(f, i)$;
2. for $i \geq 2$ and for every $j \in C(f, i)$ with $j > f[i]$, we have $f[j] \neq f[i]$.

# Validation

## Example

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 12 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $f[i]$ | | 0 | | 2 | | 0 | 1 | 1 | 2 | 0 | 1 | 2 | 3 | 4 | | ? |
| | 0 | | 1 | | 3 | | | | | | | | | | 5 | |

$C(f,16) = (f[15]+1, f[f[15]]+1, f[f[f[15]]]+1, f^4[15]+1) = (6,4,2,1)$.

The candidates for $f[16]$ are in $C(f,16) \uplus (0) = (6,4,2,1,0)$.

Among these values 2 is not valid since $f[4] = 2$.

# Validation

**Theorem 2 [FGLRSSY 02]**

The validation of an array $f$ of $n$ integers can be done in $O(n)$.

**Theorem 3 [FGLRSSY 02]**

The delay (time spent on one element) is in $O(n)$.

# Validation

## Example

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| $w[i]$ | a | a | a | a | a | a | ? |
| $f[i]$ | 0 | 1 | 2 | 3 | 4 | 5 | 1 |

# Outline

## $f \longrightarrow \delta$

**Proposition 1**

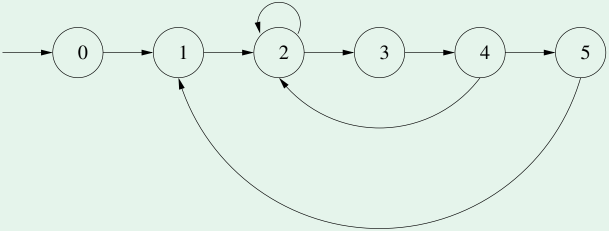
$\delta(0) = (1)$ and

$\delta(j) = (j+1) \uplus \delta(f[j]) \cup (f[j+1])$ for $1 \leq j < n$ and

$\delta(n) = \delta(f[n])$.

## $f \longrightarrow \delta$

### Example




| $j+1$ | $\uplus$ | $\delta(f[j])$ | $\dot\cup$ | $f[j+1]$ | $=$ | $\delta(j)$ | $j$ | $f[j]$ |
|---|---|---|---|---|---|---|---|---|
| (1) | $\uplus$ | | $\dot\cup$ | | $=$ | (1) | 0 | |
| (2) | $\uplus$ | (1) | $\dot\cup$ | (1) | $=$ | (2) | 1 | 0 |
| (3) | $\uplus$ | (2) | $\dot\cup$ | | $=$ | (3,2) | 2 | 1 |
| (4) | $\uplus$ | (1) | $\dot\cup$ | (1) | $=$ | (4) | 3 | 0 |
| (5) | $\uplus$ | (2) | $\dot\cup$ | | $=$ | (5,2) | 4 | 1 |
| | $\uplus$ | (1) | $\dot\cup$ | | $=$ | (1) | 5 | 0 |

# Independence from the alphabet

**Important**

This computation is completely independent from the underlying string(s).

# New validation algorithm

Assuming that $f[1..i]$ is valid, all the values for $f[i+1]$ are in $\delta'(i) \uplus (0)$ and they do not need to be checked.

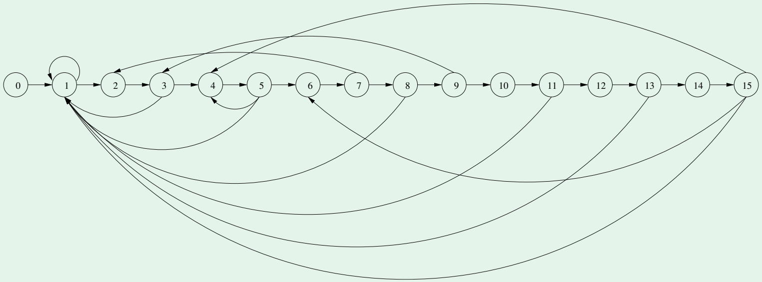
Using Proposition 1, the skeleton of the automaton is build online during the checking of the array $f$.

If $f[i+1]$ is equal to 0, it is enough to check if the cardinality of $\delta'(i)$ is strictly smaller than the alphabet size $s$ to ensure that $f$ is valid up to position $i+1$.

# New algorithm

## Example

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 12 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $f[i]$ | | 0 | | 2 | | 0 | 1 | 1 | 2 | 0 | 1 | 2 | 3 | 4 | | ? |
| | 0 | | 1 | | 3 | | | | | | | | | | 5 | |




The candidates for $f[16]$ are in $\delta'(15) \uplus (0) = (6, 4, 1, 0)$.

# Complexity

**Theorem 4**

The validity of a given integer array $f[1..n]$ can be checked in time and space $O(n)$.
If $f$ is valid, a string for which $w$ is the border array can be computed with the same complexities.

**Theorem 5**

The delay is $O(\min\{n, \textit{card}\ \Sigma\})$.

## Construction of all the distinct border arrays

An algorithm for generating all valid arrays becomes then obvious: all the valid candidates for $f[i]$ are in $\delta'(i-1) \uplus (0)$.

# Counting

| $i$ | $B(i)$ | $B(i,2)$ | $B(i,3)$ | $B(i,4)$ |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 | 2 |
| 3 | 4 | 4 | 4 | 4 |
| 4 | 9 | **8** | 9 | 9 |
| 5 | 20 | 16 | 20 | 20 |
| 6 | 47 | 32 | 47 | 47 |
| 7 | 110 | 64 | 110 | 110 |
| 8 | 263 | 128 | **262** | 263 |
| 9 | 630 | 256 | 626 | 630 |
| 10 | 1525 | 512 | 1509 | 1525 |
| 11 | 3701 | 1024 | 3649 | 3701 |
| 12 | 9039 | 2048 | 8872 | 9039 |
| 13 | 22,140 | 4096 | 21,640 | 22,140 |
| 14 | 54,460 | 8192 | 52,993 | 54,460 |
| 15 | 134,339 | 16,384 | 130,159 | 134,339 |
| 16 | 332,439 | 32,768 | 320,696 | **332,438** |

# Number of distinct border arrays on a binary alphabet

**Proposition 2**

$B(n, 2) = 2^{n-1}$.

## Number of distinct border arrays on an alphabet of size $s$

**Proposition 3**

$B(j, s) = B(j)$ for $j < 2^s$.

## Number of distinct border arrays on an alphabet of size $s$

### Proposition 4

$B(2^s, s) = B(2^s) - 1$.

The missing border array has the following form:
$0..2^0 - 1 \cdot 0..2^1 - 1 \cdots 0..2^{s-1} - 1$.

It corresponds to the string $w_s \cdot \sigma[s+1]$ (of length $2^s$) where $w_s$ is recursively defined by:
$w_1 = a$ and
$w_i = w_{i-1} \cdot \sigma[i] \cdot w_{i-1}$ for $i > 1$.

## Example

The following array $f[1..16]$ if valid on an alphabet of size at least 5:

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $w_4[i]$ | a | b | a | c | a | b | a | d | a | b | a | c | a | b | a | e |
| $f[i]$ | 0 | 0 | 1 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 |

# Outline

# Conclusions

Given an integer array $f$ we can:

- say if $f$ is valid,
  - on an unbounded size alphabet or
  - on a bounded size alphabet;
- exhibit strings for which $f$ is the border array.

$f \longleftrightarrow \delta$

Construct all the distinct border arrays

# Perspectives

Get exact bounds on the number of distinct border arrays.

# Perspectives

Let us recall the "failure function" of the Knuth-Morris-Pratt (1977) string matching algorithm

$g[j] = \max\{i \mid w[1..i-1]$ suffix of $w[1..j-1]$ and $w[i] \neq w[j]\}$.

We know that

$g[j] = \max\{\delta(j-1) - (j)\} = \max\{\delta(f[j-1]) - (f[j])\}$.