XPath evaluation in linear time

Paweł Parys
Warsaw University

We consider a problem of evaluating XPath query in an XML document:

Input: XPath unary query Q, XML document D
Output: document tree nodes,
which satisfy the query

Contribution: The above problem may be solved in time $O(|D| \cdot |Q|^3)$ for Q from a fragment of XPath called FOXPath

Which fragment?

- navigation
- comparing data
  (query $\alpha = \beta$, satisfied in nodes x such that some (x, y_1) is selected by α and some (x, y_2) is selected by β and data value in y_1 and y_2 is the same)
- we do not allow counting and positional arithmetic

Example document:

```
<html>
  <title>Nice document</title>
  <h1>Section</h1>
  <a href="http://www.uw.edu.pl/">University</a>
  <a href="http://www.google.com/">Search</a>
  <table><tr><td>
    <a href="http://www.uw.edu.pl/">
      A link in a table</a>
  </td></tr></table>
</html>
```

Example query – navigation only (CoreXPath):

```
self::"a" and not (ancestor::"table")
```

Example document:

```
<html>
  <title>Nice document</title>
  <h1>Section</h1>
  <a href="http://www.uw.edu.pl/">University</a>
  <a href="http://www.google.com/">Search</a>
  <table><tr><td>
    <a href="http://www.uw.edu.pl/">
      A link in a table</a>
  </td></tr></table>
</html>
```

Example query – comparing data (FOXPath):

```
self::"a" and (self/@href=following::"a"/@href)
```

Example document:

```
<html>
  <title>Nice document</title>
  <h1>Section</h1>
  <a href="http://www.uw.edu.pl/">University</a>
  <a href="http://www.google.com/">Search</a>
  <table><tr><td>
    <a href="http://www.uw.edu.pl/">
      A link in a table</a>
  </td></tr></table>
</html>
```

Example query – counting (AggXPath):

```
count(preceding)+1=count(root/descendant::"a")
```

(not handled by us)

Example document:

```
<html>
  <title>Nice document</title>
  <h1>Section</h1>
  <a href="http://www.uw.edu.pl/">University</a>
  <a href="http://www.google.com/">Search</a>
  <table><tr><td>
    <a href="http://www.uw.edu.pl/">
      A link in a table</a>
  </td></tr></table>
</html>
```

Example query – positional arithmetic (full XPath 1.0):

```
descendant[position()=4 and self::"a"]
```

(not handled by us)

Results summary

CoreXPath (no data)

$O(|D| \cdot |Q|)$ - Gottlob, Koch, Pichler 2002

$O(|D|^{|Q|})$ - real world XPath engines

FOXPath (comparing data)

$O(|D|^2 \cdot |Q|)$ - Gottlob, Koch, Pichler 2003

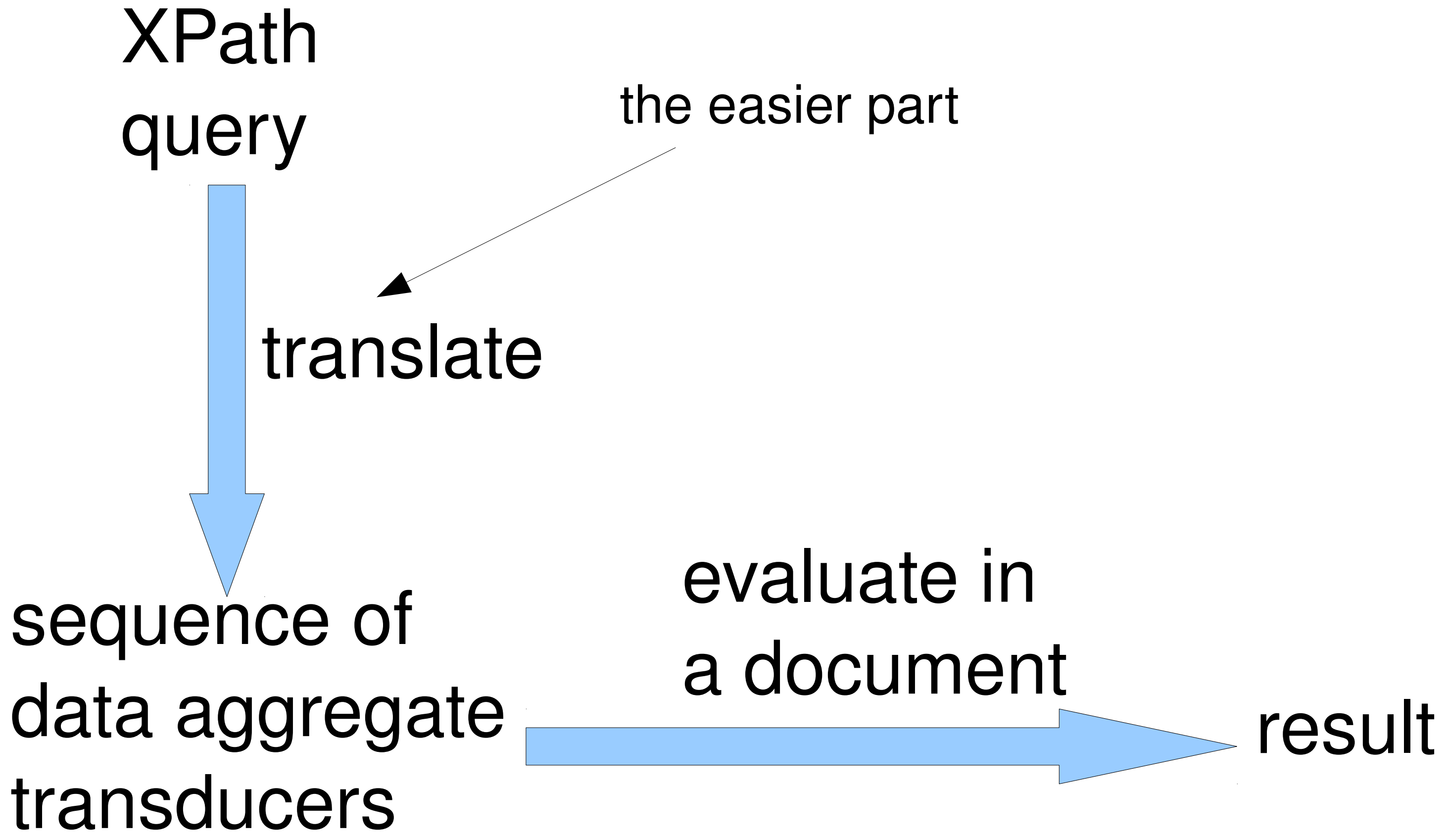
$O(|D| \cdot 2^{|Q|})$, $O(|D| \cdot \log|D| \cdot |Q|^3)$ - Bojańczyk, P. 2008

$O(|D| \cdot |Q|^3)$ - P. 2009

Full XPath (counting, node positions)

$O(|D|^4 \cdot |Q|^2)$ - Gottlob, Koch, Pichler 2003

Two step approach
(new - ICALP 2010)



Aggregate transducer

Assumption: we consider words instead of trees

A - input alphabet
B - output alphabet
$\leq$ - linear order on B
$a \sqcup b = max(a,b)$ - aggregation on B

$\sqcup$ can be any other commutative and associative operation

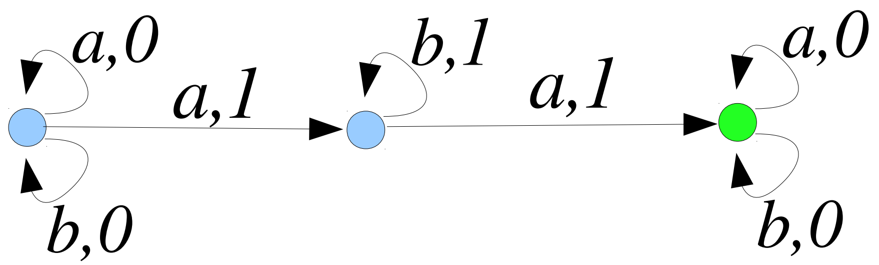
$f : A^* \rightarrow P(B^*)$ - regular letter-to-letter transducer
(nondeterministic automaton over $A \times B$)

We define $\sqcup f : A^* \rightarrow B^*$ as aggregation of f:
to calculate $\sqcup f(w)$ aggregate all results of $f(w)$

Aggregate transducer: example

$A = \{a,b\}$

$B = \{0,1\},\ 0<1$

$f=$ 

$$w = \text{b b b b b a b b a b b b b a b b a b}$$

$$f(w)=\left\{\begin{array}{l} 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0 \\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 0 \end{array}\right.$$

$$\sqcup f(w)=\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0$$

aggregation over all runs of f

Data aggregate transducer

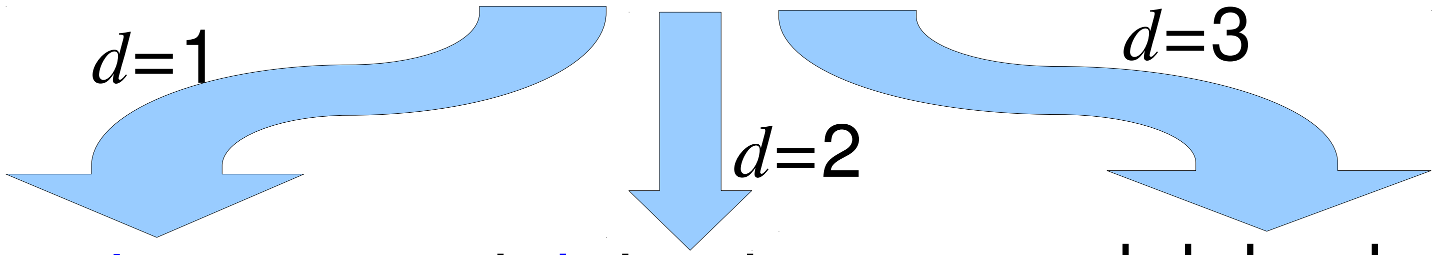
A - input alphabet
B - output alphabet with $\leq$
$\sqcup f : (A\times\{0,1\})^* \to B^*$ - aggregate transducer

Input: data word w over $A\times\mathbb{N}$
Output: word over B

For each data value $d\in\mathbb{N}$ consider the word $w(d)\in(Ax\{0,1\})^*$ which is w with marked positions with data value d.
For each of them calculate $\sqcup f(w_d)$.
Finally, aggregate all the results over all d.

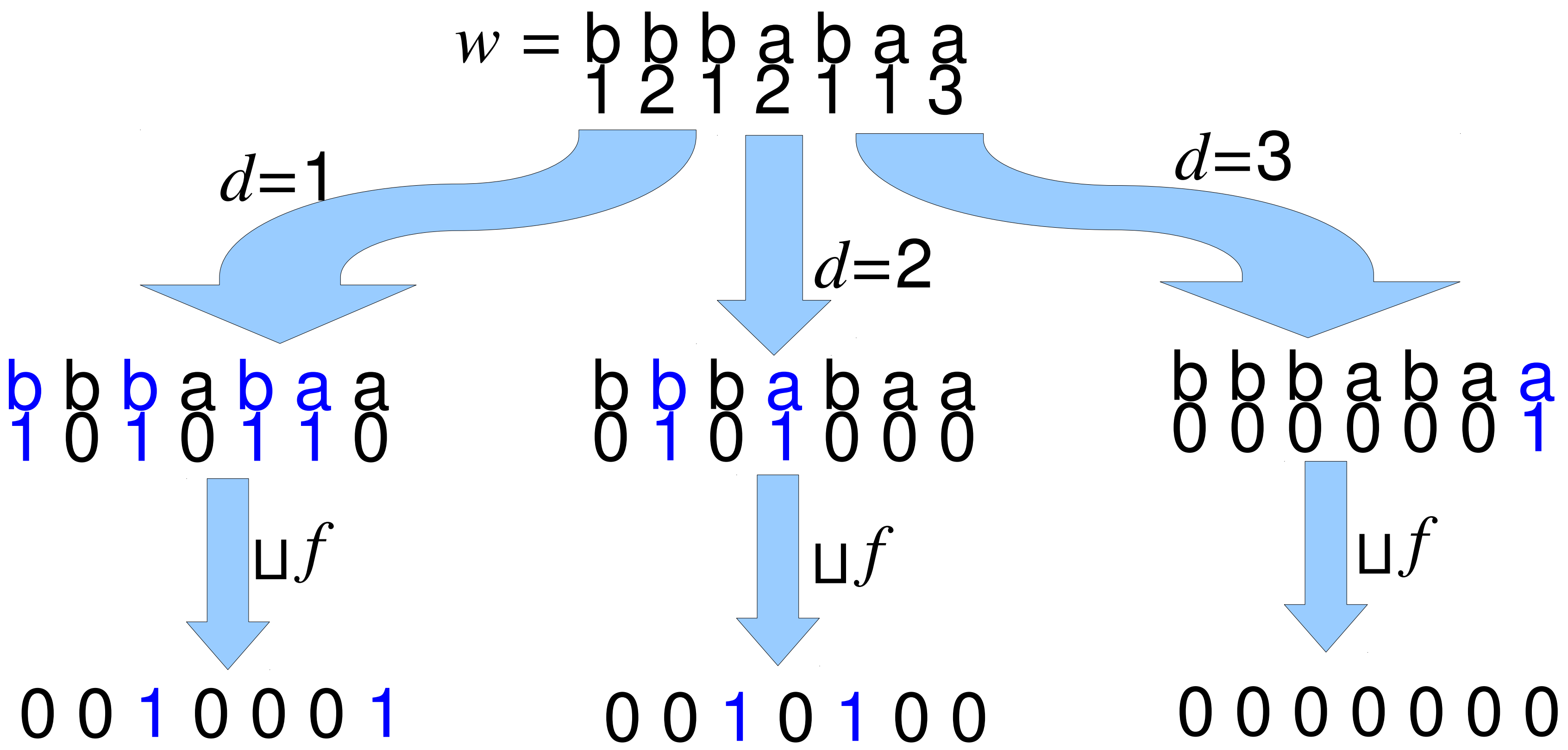
Data aggregate transducer: example

$$w = \begin{array}{ccccccc} \mathrm{b} & \mathrm{b} & \mathrm{b} & \mathrm{a} & \mathrm{b} & \mathrm{a} & \mathrm{a} \\ 1 & 2 & 1 & 2 & 1 & 1 & 3 \end{array}$$

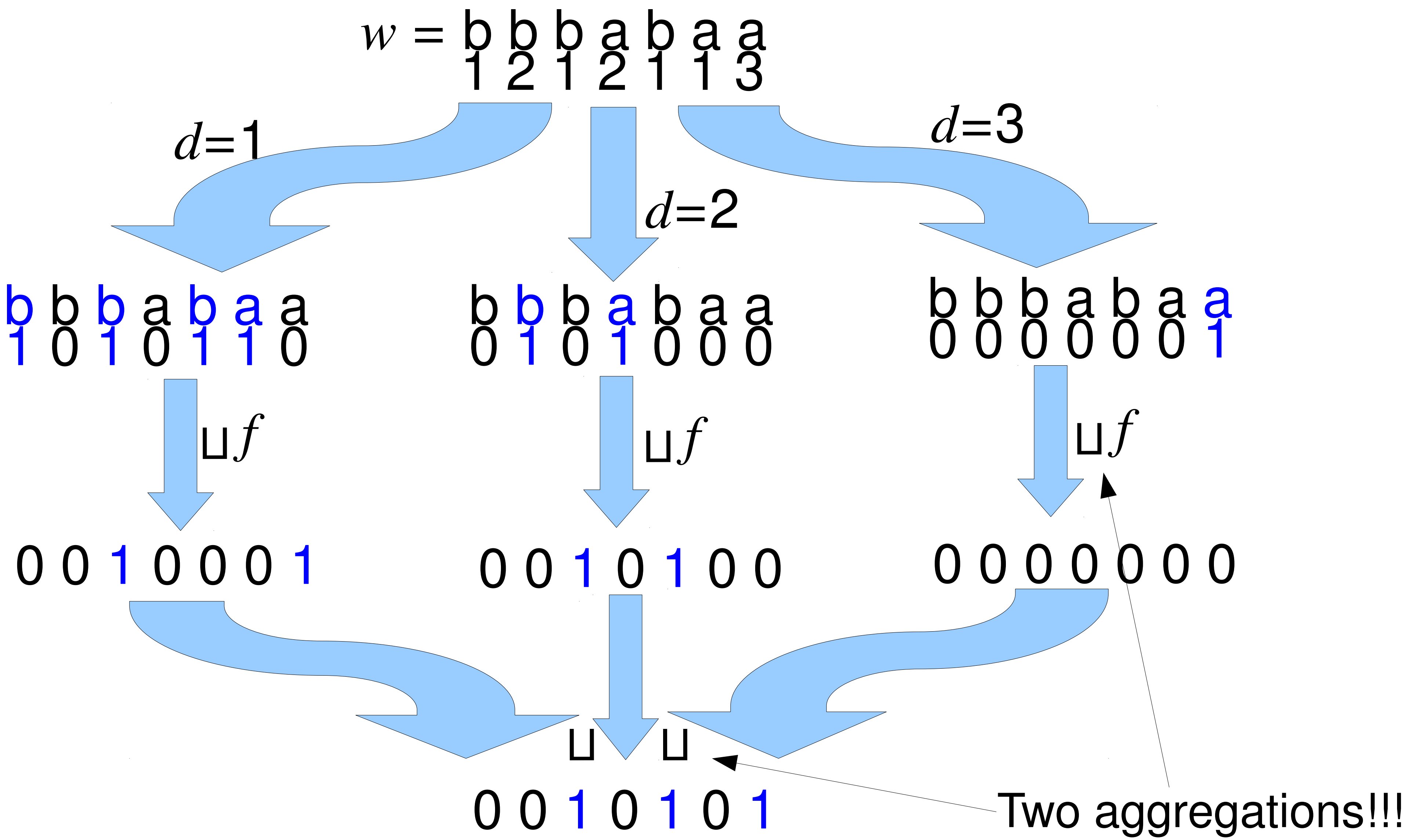


$$\begin{array}{ccccccc} \color{blue}{\mathrm{b}} & \mathrm{b} & \color{blue}{\mathrm{b}} & \mathrm{a} & \color{blue}{\mathrm{b}} & \color{blue}{\mathrm{a}} & \mathrm{a} \\ \color{blue}{1} & 0 & \color{blue}{1} & 0 & \color{blue}{1} & \color{blue}{1} & 0 \end{array} \qquad \begin{array}{ccccccc} \mathrm{b} & \color{blue}{\mathrm{b}} & \mathrm{b} & \color{blue}{\mathrm{a}} & \mathrm{b} & \mathrm{a} & \mathrm{a} \\ 0 & \color{blue}{1} & 0 & \color{blue}{1} & 0 & 0 & 0 \end{array} \qquad \begin{array}{ccccccc} \mathrm{b} & \mathrm{b} & \mathrm{b} & \mathrm{a} & \mathrm{b} & \mathrm{a} & \color{blue}{\mathrm{a}} \\ 0 & 0 & 0 & 0 & 0 & 0 & \color{blue}{1} \end{array}$$

Data aggregate transducer: example



Data aggregate transducer: example



How to evaluate data aggregate transducer on a data word?

Naive approach - from definition

Consider each data value *d* separately: calculate $\sqcup f(w_d)$.

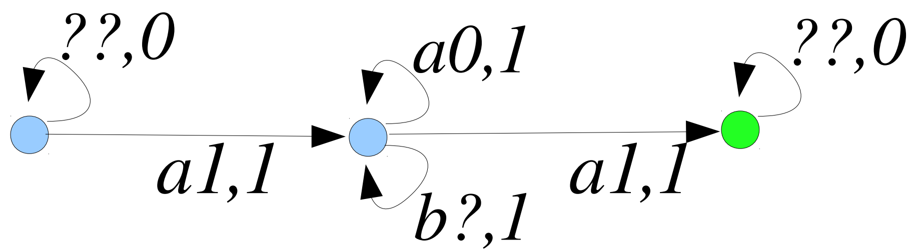
It can be done in time $O(|w|)$, there are $O(|w|)$ data value.
Thus the running time is quadratic.

How?

How to evaluate aggregate transducer in linear time?

$A = \{a0, b0, a1, b1\}$

$B = \{0,1\}$, $0<1$

$f=$ 

$$w_d = \begin{array}{cccccccccccccccccc} b & a & b & b & b & a & b & b & a & b & a & b & b & a & b & b & a & b \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{array}$$

$$f(w_d) = \begin{cases} 0\;0\;0\;0\;0\;1\;1\;1\;1\;0\;0\;0\;0\;0\;0\;0\;0\;0 \\ 0\;0\;0\;0\;0\;0\;0\;0\;1\;1\;1\;1\;1\;1\;0\;0\;0\;0 \\ 0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;1\;1\;1\;1\;0 \end{cases}$$

$$\sqcup f(w_d) = 0\;0\;0\;0\;0\;1\;1\;1\;1\;1\;1\;1\;1\;1\;1\;1\;1\;0$$

aggregation over all runs of f

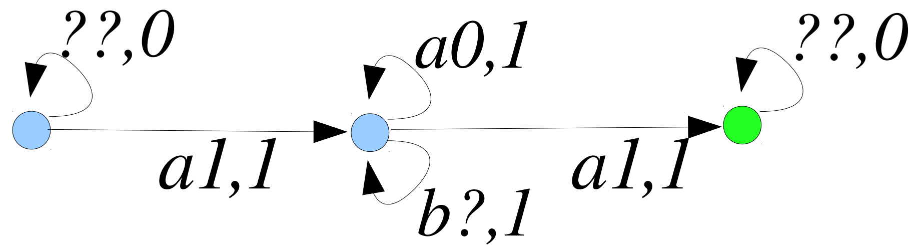
Considering each run - too slow!!

How to evaluate aggregate transducer in linear time?

$A = \{a0,b0,a1,b1\}$

$B = \{0,1\}, 0<1$

$f=$



w_d = b a b b b a b b a b a b b a b b a b

0 0 1 0 0 1 0 0 1 0 0 0 1 1 0 0 1 0

In each place of the word calculate:

- the set of states reachable from the initial state
- the set of states from which we can accept

one left-to-right pass

one right-to-left pass

Using these sets before and after a position, and the letter, we calculate (in constant time) the possible outputs there. We take maximal of them.

How to evaluate data aggregate transducer on a data word?

Better algorithm: $O(|w| log|w|)$

Idea: words w_d are very similar. So, process them together.

Moreover, the total number of ones is $|w|$, not quadratic. So, process a word w_d in time proportional to the number of ones in it.

How to evaluate data aggregate transducer on a data word?

Better algorithm: $O(|w| \log |w|)$

Puzzle 1: quick querying about runs on infixes.

Input: word $w^0 \in (A \times \{0\})^*$, transducer $f : (A \times \{0,1\})^* \to P(B^*)$

Query: positions i, j in w^0

Output: all possible state transitions on the word $w^0[i...j]$

many queries will appear

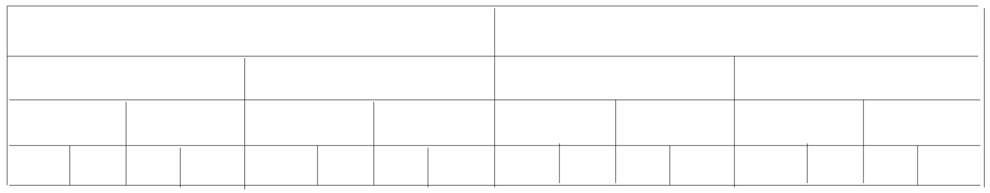
How to evaluate data aggregate transducer on a data word?

Better algorithm: $O(|w| log|w|)$

Puzzle 1: quick querying about runs on infixes.

Input: word $w^0 \in (A \times \{0\})^*$, transducer $f : (A \times \{0,1\})^* \to P(B^*)$

Preprocessing: divide the word into a tree. For each node (fragment of length 2^k) calculate possible transitions.
Running time: $O(|w^0|)$



$w^0 =$ b a b b b a b b a b a b b a b b
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

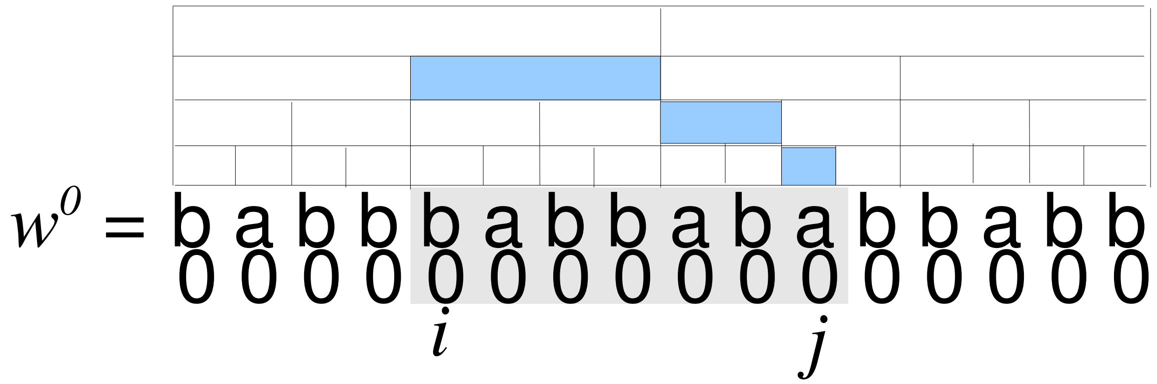
How to evaluate data aggregate transducer on a data word?

Better algorithm: $O(|w| log|w|)$

Puzzle 1: quick querying about runs on infixes.

Input: word $w^0 \in (A \times \{0\})^*$, transducer $f : (A \times \{0,1\})^* \to P(B^*)$
Query: positions i, j in w^0
Output: all possible state transitions on the word $w^0[i...j]$

Divide $w^0[i...j]$ into logarithmically many segments, compose precomputed values. Running time: $O(log|w^0|)$



How to evaluate data aggregate transducer on a data word?

Better algorithm: $O(|w| log |w|)$

Goal: “calculate” $\sqcup f(w_d)$ in time proportional to number of ones.

$$w_d = \begin{array}{cccccccccccccccccc} b & a & b & b & b & a & b & b & a & b & a & b & b & a & b & b & a & b \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{array}$$

In each place before and after 1 calculate:
- the set of states reachable from the initial state
- the set of states from which we can accept

Again: left-to-right or right-to-left pass.
But now we skip the fragments with zeros -
- we use there our magic data structure (puzzle 1)