XML and Databases
XPath evaluation (2)

Kim.Nguyen@nicta.com.au

Week 9
Recap from last week

1. Node Selection algorithm: can be used for full XPath (with filters, ancestor and parent axes, ...), but is not very efficient and cannot work in streaming.
Recap from last week

1. Node Selection algorithm: can be used for full XPath (with filters, ancestor and parent axes, ...), but is not very efficient and cannot work in streaming.

2. Automata-based algorithm: efficient, can be used for streaming XPath but only handles / and //, no filters.
Recap from last week

1. Node Selection algorithm: can be used for full XPath (with filters, ancestor and parent axes, ...), but is not very efficient and cannot work in streaming.

2. Automata-based algorithm: efficient, can be used for streaming XPath but only handles / and //, no filters.
Recap from last week

1. Node Selection algorithm: can be used for full XPath (with filters, ancestor and parent axes, ...), but is not very efficient and cannot work in streaming.

2. Automata-based algorithm: efficient, can be used for streaming XPath but only handles / and //, no filters.

Today

Automata algorithm for XPath with backward filters
To answer a query in streaming, you are only allowed to use memory proportional to **the depth** of the tree.

In practice you might need a stack whose size is at most the depth of the tree. You are not allowed to buffer the whole document, load it into memory with DOM or precompute another data-structure using a SAX parser (DAG, tables,...).
For the XPath query:

```
//a//b//d
```
Automata and XPath

For the XPath query:

```
//a/b///d
```

We can execute the NFA:
XPath with backward filters

What about the query:

//a[./ancestor::c/parent::b]/b//d[./parent::e]
XPath with backward filters

What about the query:

```
//a[./ancestor::c/parent::b]/b//d[./parent::e]
```

when is this true?
XPath with backward filters

What about the query:

```
//a[./ancestor::c/parent::b]/b//d[./parent::e]
```

when is this true?

1. The node must be an `a`
XPath with backward filters

What about the query:

```xml
//a[./ancestor::c/parent::b]/b//d[./parent::e]
```

when is this true?

1. The node must be an `a`
2. With an ancestor `c`
XPath with backward filters

What about the query:

```xml
//a[./ancestor::c/parent::b]/b//d[./parent::e]
```

when is this true?

1. The node must be an `a`
2. With an ancestor `c`
3. Whose parent is a `b`
XPath with backward filters

What about the query:

```
//a[./ancestor::c/parent::b]/b//d[./parent::e]
```

when is this true?

1. The node must be an `a`  ⇒  The descendant of
2. With an ancestor `c`
3. Whose parent is a `b`
XPath with backward filters

What about the query:

```
//a[./ancestor::c/parent::b]/b//d[./parent::e]
```

when is this true?

1. The node must be an `a` ➔ The descendant of
2. With an ancestor `c` ➔ a node `c`
3. Whose parent is a `b`
XPath with backward filters

What about the query:

```
//a[./ancestor::c/parent::b]/b//d[./parent::e]
```

when is this true?

1. The node must be an **a** ⇒ The descendant of
2. With an ancestor **c** ⇒ a node **c**
3. Whose parent is a **b** ⇒ which is the child of a **b**
XPath with backward filters

What about the query:

//a[./ancestor::c/parent::b]/b//d[./parent::e]

when is this true?

1. The node must be an a ⇒ The descendant of
2. With an ancestor c ⇒ a node c
3. Whose parent is a b ⇒ which is the child of a b
   ⇒ which can occur anywhere
XPath with backward filters

What about the query:

```
//a[./ancestor::c/parent::b]/b//d[./parent::e]
```

when is this true?

1. The node must be an `a`  ⇒  The descendant of
2. With an ancestor `c`  ⇒  a node `c`
3. Whose parent is a `b`  ⇒  which is the child of a `b`
   ⇒  which can occur anywhere
XPath with backward filters

What about the query:

```
//a[./ancestor::c/parent::b]/b//d[./parent::e]
```

when is this true?

1. The node must be an `a` ➞ The descendant of
2. With an ancestor `c` ➞ a node `c`
3. Whose parent is a `b` ➞ which is the child of a `b`
   ➞ which can occur anywhere

⇒ //b/c//*
XPath with backward filters

What about the query:

```
//a[ ./ancestor::c/parent::b]/b//d[./parent::e]
```

when is this true?

1. The node must be an `a` ⇒ The descendant of
2. With an ancestor `c` ⇒ a node `c`
3. Whose parent is a `b` ⇒ which is the child of a `b`
⇒ which can occur anywhere

⇒ `//b//c/*`

A node matches the first step of the original query if it’s an `a`-node which would be selected by the second query
XPath with backward filters

What about the query:

```
//a[ ./ancestor::c/parent::b ]/b//d[./parent::e]
```

when is this true?

1. The node must be an a
   ⇒ The descendant of
2. With an ancestor c
   ⇒ a node c
3. Whose parent is a b
   ⇒ which is the child of a b
   ⇒ which can occur anywhere

⇒ `//b/c//*`

A node matches the first step of the original query if it’s an a-node which would be selected by the second query.
XPath and backward filters

//b/c/*
XPath and backward filters

This is a simple query! We can use the automaton:

```
//b/c/**
```

And also:

```
[[.//parent::e] becmes //e/** for which we can use:
```

```
* b c *
```

```
5 6 7 8
```
This is a simple query! We can use the automaton:

```
//b/c/**
```

And also: [ ./parent::e ] becomes //e/* for which we can use:

```
* e *
```
Running the automata

From

```
//a[./ancestor::c/parent::b]/b//d[./parent::e]
```

we get

\[ A_0 = \begin{array}{cccc}
1 & \rightarrow & a & \rightarrow & 2 \\
& \rightarrow & b & \rightarrow & 3 \\
& & d & \rightarrow & 4 \\
\end{array} \]

\[ A_2 = \begin{array}{cccc}
5 & \rightarrow & b & \rightarrow & 6 \\
& \rightarrow & c & \rightarrow & 7 \\
& & * & \rightarrow & 8 \\
\end{array} \]

\[ A_4 = \begin{array}{cccc}
9 & \rightarrow & e & \rightarrow & 10 \\
& & * & \rightarrow & 11 \\
\end{array} \]

\( A_0 \) is the automaton for the “main” XPath expression. The other \( A_i \) automata correspond to the filter which must be checked for state \( i \) of automaton \( A_0 \).
Query transformation algorithm

1. Split the query into a “main” downward query and its filters.
   e.g.:
   
   //a[./ancestor::c/parent::b]/b//d[./parent::e]
   becomes:
   //a/b//d, ./ancestor::c/parent::b, ./parent::e
Query transformation algorithm

1. Split the query into a “main” downward query and its filters.
   e.g.:
   
   ```xml
   //a[./ancestor::c/parent::b]/b//d[./parent::e]
   becomes:
   //a/b//d, ./ancestor::c/parent::b, ./parent::e
   ```

2. The main query is unchanged. Transform the backward queries into forward ones. e.g.:
   ```xml
   //a/b//d, ./ancestor::c/parent::b, ./parent::e
   becomes:
   //a/b//d, //b/c//*, //e/*
   ```
Query transformation algorithm

1. Split the query into a “main” downward query and its filters.
   e.g.:
   \[//a[./ancestor::c/parent::b]/b//d[./parent::e]\]
   becomes:
   \[//a/b//d, ./ancestor::c/parent::b, ./parent::e\]

2. The main query is unchanged. Transform the backward queries into forward ones. e.g.:
   \[//a/b//d, ./ancestor::c/parent::b, ./parent::e\]
   becomes:
   \[//a/b//d, //b/c/*, //e/*\]

3. Transform each query obtained in step 2 into an NFA.
Remember, to evaluate an NFA, you keep track of the current states that you have reached, in a set of states $S$.

When you read a label, for each state in $S$, you compute the destination states according to the transitions and put them in a state $S'$.

For instance with:  

```
\* \rightarrow a \rightarrow b \rightarrow \* \rightarrow d
```

if your current set of states is $\{1, 3\}$ and you see an a-node, you go into the states $\{1, 2, 3\}$. Now, we just have several automata, so we keep several sets of states.

“Reading a label” corresponds to seeing a `startElement(...)`
Assume you have an automaton class: Auto with the following methods:

- `StateSet transition(String label, StateSet S)`: Computes the set of states reachable from the states in S, with a given label.
Assume you have an automaton class: `Auto` with the following methods:

- `StateSet transition(String label, StateSet S)`: Computes the set of states reachable from the states in `S`, with a given label.
- `bool isFinal(StateSet S)`: returns true if a final state of the automaton is in `S`. 

Assume that your main query has states: \( \{1, \ldots, N\} \) You need:

- An array \( \text{AAutos}[N+1] \) containing Auto objects. \( \text{AAutos}[0] \) contains the automaton for the main query, \( \text{AAutos}[i] \) contains the automaton for state \( i \) of the main query (can be null if there is no automaton for that state)
- A counter the preorder number
- A Stack which will contain arrays of set of states. Each array has size \( N+1 \) the cell \( i \) of such an array contains the current set of states for automaton \( \text{AAutos}[i] \).
XPath evaluation algorithm over SAX events

**Initialisation**: Create an array States[\(N+1\)] and put in States[\(i\)] the set containing the initial state of AAutos[\(i\)] (put null if AAutos[\(i\)] == null). Push States on the Stack.
XPath evaluation algorithm over SAX events

**Initialisation** : Create an array States[N+1] and put in States[i] the set containing the initial state of AAutos[i] (put null if AAutos[i] == null). Push States on the Stack.

**startElement(String label)** : States = top of the Stack.
XPath evaluation algorithm over SAX events

**Initialisation** : Create an array States\[N+1\] and put in States\[i\] the set containing the initial state of AAutos\[i\] (put null if AAutos\[i\] == null). Push States on the Stack.

**startElement(String label)** : States = top of the Stack.
Initialisation : Create an array States[N+1] and put in States[i] the set containing the initial state of AAutos[i] (put null if AAutos[i] == null). Push States on the Stack.

startElement(String label) : States = top of the Stack.

Create a new array of sets of states NextStates[N+1].
XPath evaluation algorithm over SAX events

**Initialisation** : Create an array States[N+1] and put in States[i] the set containing the initial state of AAutos[i] (put null if AAutos[i] == null). Push States on the Stack.

**startElement(String label)** : States = top of the Stack.

Create a new array of sets of states NextStates[N+1]. For \( i = 1 \) to \( N \)

if (AAutos[i] != null)

NextStates[i] = AAutos[i].transition(label,States[i])
**XPath evaluation algorithm over SAX events**

**Initialisation**: Create an array States\([N+1]\) and put in States\([i]\) the set containing the initial state of AAutos\([i]\) (put null if AAutos\([i]\) == null). Push States on the Stack.

**startElement(String label)**: States = top of the Stack. Create a new array of sets of states NextStates\([N+1]\).

For \(i = 1\) to \(N\)

- if (AAutos\([i]\) != null)
  - NextStates\([i]\) = AAutos\([i]\).transition(label, States\([i]\))
- Compute Stemp = AAutos\([0]\).transition(label, States\([0]\))
XPath evaluation algorithm over SAX events

**Initialisation**: Create an array States[N+1] and put in States[i] the set containing the initial state of AAutos[i] (put null if AAutos[i] == null). Push States on the Stack.

**startElement(String label)**: States = top of the Stack. Create a new array of sets of states NextStates[N+1]. For i = 1 to N
  if (AAutos[i] != null)
    NextStates[i] = AAutos[i].transition(label,States[i])
  Compute Stemp = AAutos[0].transition(label,States[0])
  For q ∈ Stemp
    if (AAutos[i] == null || AAutos[i].isFinal(NextStates[i]))
      leave q in Stemp, otherwise remove it.
**XPath evaluation algorithm over SAX events**

**Initialisation** : Create an array $\text{States}[N+1]$ and put in $\text{States}[i]$ the set containing the initial state of $\text{AAutos}[i]$ (put null if $\text{AAutos}[i] == \text{null}$). Push $\text{States}$ on the Stack.

**startElement(String label)** : $\text{States} = \text{top of the Stack}$. Create a new array of sets of states $\text{NextStates}[N+1]$. For $i = 1$ to $N$

- if ($\text{AAutos}[i] != \text{null}$)
  
  $\text{NextStates}[i] = \text{AAutos}[i].\text{transition(label,States}[i]])$

Compute $\text{Stemp} = \text{AAutos}[0].\text{transition(label,States}[0]])$

For $q \in \text{Stemp}$

- if ($\text{AAutos}[i] == \text{null} \|\| \text{AAutos}[i].\text{isFinal(NextStates}[i]))$
  
  leave $q$ in $\text{Stemp}$, otherwise remove it.
```java
startElement(String label) (continued) :
    NextStates[0] = Stemp
```
startElement(String label) (continued):

NextStates[0] = Stemp
XPath evaluation algorithm over SAX events

startElement(String label) (continued) :
  
  NextStates[0] = Stemp
  
  if (AAutos[0].isFinal(NextStates[0]))
    
    print the current preorder
XPath evaluation algorithm over SAX events

**startElement(String label) (continued)**:

NextStates[0] = Stemp
if (AAutos[0].isFinal(NextStates[0]))
   print the current preorder
Push NextStates on the stack
XPATH evaluation algorithm over SAX events

**startElement(String label) (continued)**:

NextStates[0] = Stemp
if (AAutos[0].isFinal(NextStates[0]))
    print the current preorder
Push NextStates on the stack
Increment preorder counter
XPath evaluation algorithm over SAX events

**startElement(String label) (continued)**:

NextStates[0] = Stemp

if (AAutos[0].isFinal(NextStates[0]))
    print the current preorder
Push NextStates on the stack
Increment preorder counter

**endElement(String label)**: Just pop the stack!
Next week

- Adding following-sibling/preceding siblings
- More hints/pseudo code on how to implement automata