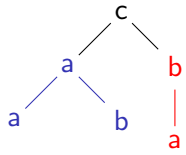
Stackless Processing of Streamed Trees

Corentin Barloy, Filip Murlak, Charles Paperman

PODS 2021

Processing streamed trees

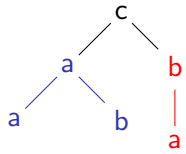
XML encoding of trees:



$\langle c \rangle$
  $\langle a \rangle$
    $\langle a \rangle \langle / a \rangle$
    $\langle b \rangle \langle / b \rangle$
  $\langle / a \rangle$
  $\langle b \rangle$
    $\langle a \rangle \langle / a \rangle$
  $\langle / b \rangle$
$\langle / c \rangle$

Processing streamed trees

XML encoding of trees:

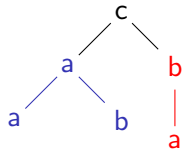


$\langle c \rangle$
  $\langle a \rangle$
    $\langle a \rangle \langle / a \rangle$
    $\langle b \rangle \langle / b \rangle$
  $\langle / a \rangle$
  $\langle b \rangle$
    $\langle a \rangle \langle / a \rangle$
  $\langle / b \rangle$
$\langle / c \rangle$

Two problems: validation and querying.

Processing streamed trees

XML encoding of trees:



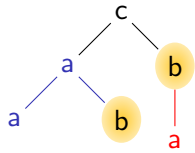
$\langle c \rangle$
$\langle a \rangle$
$\langle a \rangle \langle / a \rangle$
$\langle b \rangle \langle / b \rangle$
$\langle / a \rangle$
$\langle b \rangle$
$\langle a \rangle \langle / a \rangle$
$\langle / b \rangle$
$\langle / c \rangle$

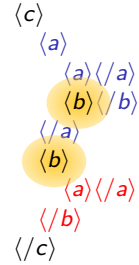
Two problems: validation and querying.

RPQs: the path from the root belongs to a given regular language.

Processing streamed trees

XML encoding of trees:





Two problems: validation and querying.

RPQs: the path from the root belongs to a given regular language.
For instance, the RPQ associated to ca^*b.

Evaluation in constant memory

Which RPQs can be evaluated in constant memory?

Theorem (Effective characterisation)

RPQ L can be evaluated in constant memory $\Leftrightarrow$ L is (almost) reversible

Which RPQs can be evaluated in constant memory?

Theorem (Effective characterisation)

RPQ L can be evaluated in constant memory $\Leftrightarrow$ L is (almost) reversible

The minimal automaton is co-deterministic, that is, after reversing the arrows it is deterministic.

Which RPQs can be evaluated in constant memory?

Theorem (Effective characterisation)

RPQ L can be evaluated in constant memory $\Leftrightarrow$ L is (almost) reversible

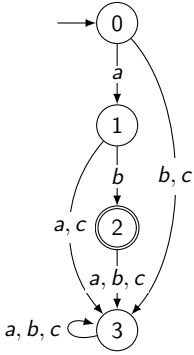
The minimal automaton is co-deterministic, that is, after reversing the arrows it is deterministic.

Algorithm:

- When an opening tag is read, follow the transition in the automaton.
- When a closing tag is read, follow the transition in the reverse automaton.

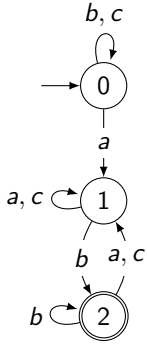
Limitations of constant memory evaluation

Many queries cannot be evaluated in constant memory.



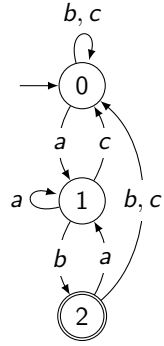
ab

$/a/b$



$(a+b+c)^*a(a+b+c)^*b$

$//a//b$

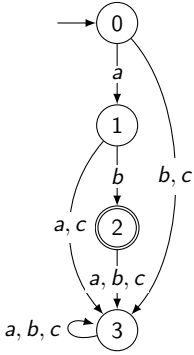


$(a+b+c)^*ab$

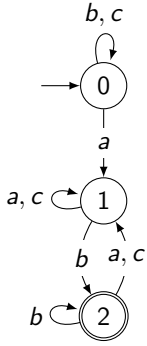
$//a/b$

Limitations of constant memory evaluation

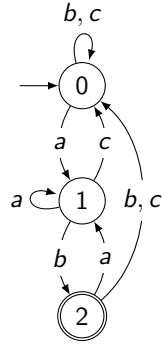
Many queries cannot be evaluated in constant memory.



ab
$/a/b$



$(a+b+c)^*a(a+b+c)^*b$
$//a//b$



$(a+b+c)^*ab$
$//a/b$

They can be evaluated using a stack, but this is costly (memory linear in the depth).

Stackless queries
(Evaluation in logarithmic memory)

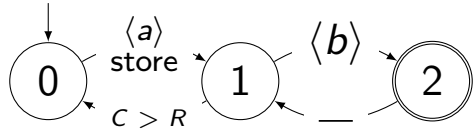
Stackless automata

Main ingredients:
- a finite state machine,
- a counter that stores the current depth in the tree,
- a finite number of registers where the counter values can be stored,
- can compare register values with the current depth.

Stackless automata

Main ingredients:
- a finite state machine,
- a counter that stores the current depth in the tree,
- a finite number of registers where the counter values can be stored,
- can compare register values with the current depth.

Evaluating $(a+b+c)^*a(a+b+c)^*b$
$//a//b$



Effective characterisation of stackless RPQs

Theorem

The RPQ L is stackless $\Leftrightarrow$ L is hierarchically (almost) reversible

Effective characterisation of stackless RPQs

Theorem

The RPQ L is stackless $\Leftrightarrow$ L is hierarchically (almost) reversible

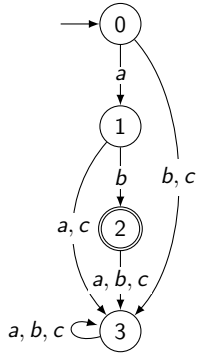
Each strongly connected component is (almost) reversible:

Effective characterisation of stackless RPQs

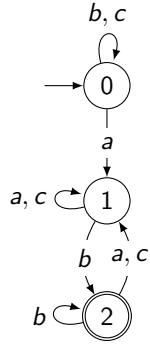
Theorem

The RPQ L is stackless $\Leftrightarrow$ L is hierarchically (almost) reversible

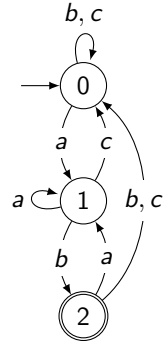
Each strongly connected component is (almost) reversible:



$/a/b$
is HAR



$//a//b$
is HAR



$//a/b$
is not HAR

When can we validate a document in constant memory?

Validation

Check if the tree conforms to the given schema, modelled as a regular tree language.

When can we validate a document in constant memory?

Validation

Check if the tree conforms to the given schema, modelled as a regular tree language.

Weak validation [Segoufin, Vianu PODS'02]

Assume that the input is a correct encoding of some tree.

When can we validate a document in constant memory?

Validation

Check if the tree conforms to the given schema, modelled as a regular tree language.

Weak validation [Segoufin, Vianu PODS'02]

Assume that the input is a correct encoding of some tree.

Characterize the schemas that can be weakly validated in constant memory.

When can we validate a document in constant memory?

Validation

Check if the tree conforms to the given schema, modelled as a regular tree language.

Weak validation [Segoufin, Vianu PODS'02]

Assume that the input is a correct encoding of some tree.

Characterize the schemas that can be weakly validated in constant memory.

Segoufin & Vianu solved it for fully recursive DTDs.

We solve it for tree languages of the form: each branch is in language L.

General problem still open, both for constant-memory and our stackless model.

Conclusion

- Similar characterisations hold for JSON-like encoding, where closing tags carry no information on the letters.

Conclusion

- Similar characterisations hold for JSON-like encoding, where closing tags carry no information on the letters.
- Ongoing work on leveraging schemas for querying streamed trees.

Conclusion

- Similar characterisations hold for JSON-like encoding, where closing tags carry no information on the letters.
- Ongoing work on leveraging schemas for querying streamed trees.
- Ongoing work on vectorization.