Theoretical Computer Science 578 (2015) 100–125

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Early nested word automata for XPath query answering on XML streams

Denis Debarbieux [a,c], Olivier Gauwin [d,e], Joachim Niehren [a,c], Tom Sebastian [b,c,*], Mohamed Zergaoui [b]

[a] Inria Lille, France
[b] Innovimax, France
[c] LIFL, France
[d] LaBRI, France
[e] University of Bordeaux, France

ARTICLE INFO

Article history:
Received 2 November 2013
Received in revised form 23 June 2014
Accepted 14 January 2015
Available online 20 January 2015

Keywords:
Automata
Logic
Trees
Nested words
Streams
Databases
Document processing
XML
XPath
XSLT
XQuery

ABSTRACT

Algorithms for answering XPath queries on XML streams have been studied intensively in the last decade. Nevertheless, there still exists no solution with high efficiency and large coverage. In this paper, we introduce early nested word automata in order to approximate earliest query answering algorithms for nested word automata. Our early query answering algorithm is based on stack-and-state sharing for running early nested word automata on all answer candidates with on-the-fly determinization. We prove tight upper complexity bounds on time and space consumption. We have implemented our algorithm in the QuiXPath system and show that it outperforms all previous tools in coverage on the XPathMark benchmark, while obtaining very high time and space efficiency and scaling to huge XML streams. Furthermore, it turns out that our early query answering algorithm is earliest in practice on most queries from the XPathMark benchmark.[1]

1. Introduction

XML is a major format for information exchange besides Json, also for Rdf linked open data and relational data. Therefore, complex event processing for XML streams has been studied for more than a decade [14,7,26,29,5,24,15,10,25]. Query answering for XPath is a basic algorithmic task on XML streams, since XPath is a language hosted by the W3C standards XSLT and XQuery.

Memory efficiency is essential for processing XML documents of several gigabytes that do not fit in main memory, while high time efficiency is even more critical in practice. Nevertheless, so far there exists no solution for XPath query answering on XML streams with high coverage and high efficiency. The best coverage on the usual XPathMark benchmark [8] is reached by Olteanu's Spex [26] with 22% of the use cases. The time efficiency of Spex, however, is only average, for instance

* Corresponding author.
E-mail address: tom.sebastian@inria.fr (T. Sebastian).

[1] Thanks to the QuiXProc project of Inria and Innovimax and the Cnrs Sosp project.

http://dx.doi.org/10.1016/j.tcs.2015.01.017
0304-3975/

compared to Gcx [29] which often runs in time close to the parsing time. We hope that this unsatisfactory situation can be resolved in the near future by pushing existing automata techniques forwards [14,24,10,25].

In contrast to sliding window techniques for monitoring continuous streams [3,19], the usual idea of answering queries on Xml streams is to buffer only *alive* candidates for query answers. These are stream elements which may be selected in some continuation of the stream and rejected in others. All space-optimal algorithms have to remove non-alive elements from the buffer, by outputting them or by discarding them. Unfortunately, this kind of *earliest query answering* is not feasible in polynomial time for XPath queries [6], as first shown by adapting counter examples from online verification [16]. A second argument is that deciding aliveness is more difficult than deciding XPath satisfiability [10], which is coNP-hard even for small fragments of XPath [4]. The situation is different for queries defined by deterministic *nested word automata* (Nwas) [1,2], for which earliest query answering is feasible with polynomial resources [24,11]. Many practical XPath queries (without aggregation, joins, and negation) can be compiled into small Nwas [10], while relying on non-determinism for modeling descendant and following axes. This, however, does not lead to an efficient streaming algorithm. The problem is that a cubic time precomputation in the size of the *deterministic* Nwa is needed for earliest query answering [11], and that the determinization of Nwas raises huge blow-ups in average (in contrast to finite automata).

Most existing algorithms for streaming XPath evaluation approximate earliest query answering, most prominently: Spex's algorithm on the basis of transducer networks [26], Saxon's streaming Xslt engine [15], and Gcx [29] which implements a fragment of XQuery. The recent XSeq tool [25], in contrast, restricts XPath queries by ruling out complex filters all over. In this way, node selection can always be decided with 0-delay [12] once having read the attributes of the node (which follow its opening event). Such queries are called begin-tag determined [5] if not relying on attributes. In this paper, we propose a new algorithm approximating earliest query answering for XPath queries that is based on Nwas. One objective is to improve on the previous approximations, in order to support earliest rejection for XPath queries with negation, such as for instance:

```
//book[not(pub/text()='Springer')][contains(text(),'Lille')]
```

When applied to an Xml document for an electronic library, as below, all books published from `Springer` can be rejected once its publisher was read:

```
<lib>...<book>...<pub> Springer </pub>
              ...<content>...Lille...</content>...</book>...</lib>
```

Spex, however, will check for all books from Springer whether they contain the string `Lille` and detect rejection only when the closing tag `</book>` is met. This requires unnecessary buffering space.

As a first contribution, we provide an approximation of the earliest query answering algorithm for queries defined by Nwa [11,24], while removing the assumption of determinism imposed there. The main idea to gain efficiency is that selection and rejection should depend only on the current state of an Nwa but not on its current stack. Therefore, we propose *early nested word automata* (eNwas) that are Nwas with two kinds of distinguished states: rejection states and selection states. Selection states are final and must always remain final, so that a nested word can be accepted, once one of its prefixes reaches a selection state. Symmetrically, rejection states can never reach a final state, so that a nested word can be rejected, once all non-blocking runs on a prefix reach a rejection state. We then present a new streaming algorithm for answering eNwa queries in an early manner. The basic idea is to run the eNwa for all possible candidates while determinizing on-the-fly, so that one can see easily whether all non-blocking runs of the nondeterministic automaton reach a rejection state, or whether one of them is selecting. The second idea is to share the stacks and states of runs of buffered candidates in the same state, so that the running time does not depend on the number of buffered candidates, but only on the number of states of the deterministic automaton discovered during the on-the-fly determinization. Our streaming algorithm with stack-and-state sharing for answering eNwas queries is original and nontrivial. It enables tight upper bounds for time and space complexity that we prove (Theorem 13).

As a second contribution, we show how to compile XPath expressions to small eNwa descriptors defining the same query. These descriptors allow to represent eNwas with large finite alphabets in a succinct manner, by replacing labels in eNwa rules by label descriptors. The label descriptor $\neg a$, for instance, stands for the set of all finitely many labels different from a. The target of our XPath-compiler is thus an eNwa descriptor. For instance, the eNwa descriptors that our XPath compiler obtains for the XPath expressions $P_n = \texttt{child::}a_1\texttt{/child::}a_2\texttt{/.../child::}a_n$ are of size $O(n)$, while the described eNwa is of size $O(n^2)$. The latter has n states each of which has n transitions, in order to accept children with all possible letters $a_1, \ldots, a_n$. We will prove a tight time bound for our compiler (Theorem 11). It implies the same bound on the size of the generated eNwa descriptors, and in particular that the eNwa descriptors for any XPath expressions without filters, unions, and with no other axes than *child* axes (such as P_n for instance) can be compiled in time $O(n)$. This improves on the previous compiler to dNwas from [10], which required time $O(n^4)$ for P_n. The main idea of the compiler is to adapt the previous translation to dNwas, so that it produces descriptors of eNwas while distinguishing selection and rejection states. We maintain pseudo-completeness (no run can ever block) as an invariant, so that we can compile negations efficiently in the deterministic case. Otherwise, we treat negation based on eNwa determinization after the instantiation of the eNwa descriptors, even though this is costly in theory and often unfeasible in practice. The XPath operators introducing