View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Dagstuhl Research Online Publication Server

Compression vs Queryability - A Case Study

Siva Anantharaman
LIFO, Université d'Orléans (France)
e-mail: siva@univ-orleans.fr

Abstract

Some compromise on compression is known to be necessary, if the relative positions of the information stored by semi-structured documents are to remain accessible under queries. With this in view, we compare, on an example, the 'query-friendliness' of XML documents, when compressed into straightline tree grammars which are either regular or context-free. The queries considered are in a limited fragment of XPath, corresponding to a type of patterns; each such query defines naturally a non-deterministic, bottom-up 'query automaton' that runs just as well on a tree as on its compressed dag.

Keywords: Tree automata, Tree Grammars, Dags, XML documents, Queries.

1 Introduction

Structures over dags instead of over trees have been widely used in order to optimize algorithms. Tree automata (TA) are among the basic tools employed for querying XML documents (e.g., [10, 11, 16, 17]); on the other hand, the notion of a compressed XML document has been introduced in [2, 9, 14], and a possible advantage of using dag structures for the manipulation of such documents has been brought out in [14]. It is legitimate then to investigate the possibility of using automata running directly over dags instead of over trees, for querying compressed XML documents. Unfortunately however, the Dag Automata (DA), defined as a natural extension of tree automata in [5] as bottom-up tree automata running on dags, cannot directly serve such a purpose. The reason is that the class of their languages – defined as the set of dags accepted under their bottom-up runs – is algebraically ill-behaved ([1]): although a *deterministic* bottom-up TA runs exactly alike on a dag or on its uncompressed tree, the set of dags accepted by a *non-deterministic* DA does not represent, in general, a regular tree language. Thus – if the notion of acceptance is not 'adapted appropriately' –, the languages of DAs (would) form a *strict* superclass of the class of regular tree languages. Note, on the other hand, that the answers to MSO-definable queries on semi-structured trees are known to be regular tree languages, cf. [17, 19]. So, for a suitable definition of acceptance. a bottom-up

Dagstuhl Seminar Proceedings 08261
Structure-Based Compression of Complex Massive Data
http://drops.dagstuhl.de/opus/volltexte/2008/1676

run of a non-deterministic TA, *on a dag*, has to be coupled in general with an "on the fly determinization" of the TA along the run.

Several observations are in order at this point, before we proceed. First, we are concerned with queries on XML documents, and the trees modeling such documents are unranked, i.e., the symbols at the nodes can have arbitrarily many arguments; so the automata we want to employ for querying should also be unranked. A second observation is that, fully or partially compressing an *unranked* tree into a DAG is not a major issue: suffices to represent it as a *straightline regular* tree grammar (SLR), e.g., as in [3]. In the sequel, therefore, we shall consider the terms 'document' and 'dag' to be synonymous; and by a 'tree', we shall mean a dag with 'copying fully allowed', i.e., with no compression at all. It must be pointed out here that a higher degree of compression of a *ranked* tree can be achieved by representing it as a *straightline, context-free,* tree grammar (SLCF), cf. e.g., [3, 4]; but it is not clear if the requirement on the base alphabet to be ranked can be dropped. Now, patterns (cf. [15], and Section 3 below) are often conveniently used to specify queries on XML documents; and they can be naturally visualized as bottom-up, non-deterministic, query automata; the full evaluation of such queries – i.e., *access to the information stored, along with their respective relative positions* – is quite possible on documents compressed under SLR, without any need for decompression; while on a document compressed under SLCF, checking for query satisfiability can be checked efficiently, but full query evaluation appears to be more intricate.

Several mechanisms have been proposed in the literature for querying (compressed, unranked) documents. The one proposed in [2] is a priori for query satisfiability, and employs 'tree-like' automata; it can be either top-down, or bottom-up, on dags. That of [7] is a mixture of top-down and bottom-up analysis, based on the SLR vision, suitable for query evaluation on dags, for a fragment of Core XPath, but it is not hard to extend it a little farther. [17] proposes a very general query mechanism named *query automaton* (QA), which is a *2-way tree automaton with specified selecting states*: a selection label '1' is attributed to some specified states, and '0' or '$\perp$' to the others; this works only for trees, however.

For the view presented below, our concern will be limited to a class of queries that are expressible inside a restricted XPath format; they correspond to the *patterns* using the / and // of XPath, as defined in [15], possibly with some filters (or 'branches', as was called there). Besides being simple, this limitation will also have the advantage that such queries can be visualized as usual, bottom-up, non-deterministic tree automata with specified selecting states (Note: this, incidentally, will also allow us to consider n-ary queries).

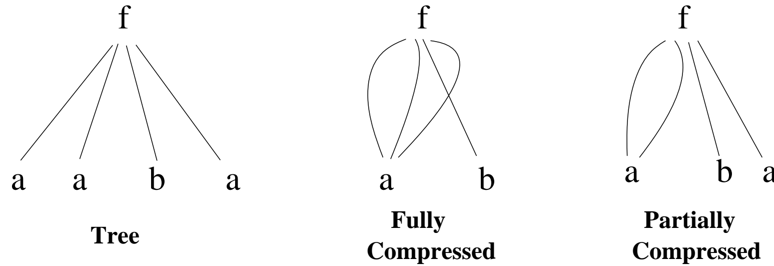
This paper is structured as follows. The notation and the preliminaries are given in Section 2. In Section 3, we define the notion of patterns (as in [15]), and show how to visualize them, naturally, as bottom-up query automata (QA); we also show, on an example, how to evaluate the query corresponding to a given pattern p on a given document t, via a bottom-up run on t of the QA associated with p, if the document is compressed under SLR; on the same example, we will

also show that, if the document gets compressed as SLCF, the query is not that easy to evaluate (although it can be tested for satisfiability).

A final remark before closing this section: although we noted above that bottom-up runs of non-deterministic TA on *dags* can be 'problematic', no such complication arises actually for the QAs defined by the patterns we consider here; so, we do not need to determinize the QA on the fly, along its evaluating runs. Moreover, it can be shown that the answer to the query defined by any given pattern on a given SLR-compressed document t, will be the 'same' as when the query is evaluated on the uncompressed tree equivalent of t.

2 Notation and Preliminaries

We assume given an unranked base alphabet Σ. Trees and dags over Σ are defined as usual, each of their nodes baring a name that is a symbol from Σ. (Formal definitions do not seem needed.) A first example to show what we mean by fully or partially compressed dags is the following:



The most elegant and efficient way to distinguish between these 3 formats of the 'same document' (same information, but stored with more or less optimization) is to associate to each format t, *canonically*, a straightline regular tree grammar (SLR) G_t: canonical in the sense that G_t recognizes exactly t, and nothing else. The three respective SLR for the above 3 formats are as follows:

$X_0 \to f(X_1, X_2, X_3, X_4)$
$X_1 \to a$
$X_2 \to a$
$X_3 \to b$
$X_4 \to a$

$X_0 \to f(Y_1, Y_1, Y_2, Y_1)$
$Y_1 \to a$
$Y_2 \to b$

$X_0 \to f(Z_1, Z_1, Z_2, Z_3)$
$Z_1 \to a$
$Z_2 \to b$
$Z_3 \to a$

The notions of *child, parent, descendent, ancestor* are all defined, in a natural manner, on the set of nodes of any given dag. A node is a *root* (resp. *leaf*) for a dag, iff it has no parent (resp. child). All our dags will be assumed rooted, i.e., to have a unique root node. It is then easy to associate a *set of positions* to any given node on any given dag, such that the following holds: A dag is a tree iff the set of positions of any of its nodes is singleton. (Note: a node on a dag can have more than one parents.)

We also need the the notion of a bottom-up tree automaton over an unranked alphabet; to facilitate understanding, we first recall the notion over a ranked alphabet; the definition is easily extended to the unranked case.

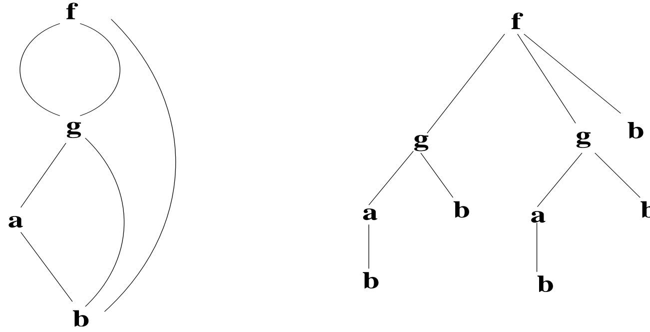
Definition 1 *A bottom-up tree automaton (TA) over a ranked alphabet* Σ *is a tuple* (Σ, Q, F, Δ)*, where* Q *is a finite non-empty set of states,* $F \subseteq Q$ *is the set of final (or accepting) states, and* Δ *is a set of transition rules of the form:* $f(q_1, ..., q_k) \to q$*, where* $f \in \Sigma$ *is of rank* k*, and* $q_1, \ldots, q_k, q \in Q$*.*

Now, a transition $f(q_1, \ldots, q_k) \to q$ can also be written as $f(q_1 \ldots q_k) \to q$, where $q_1 \ldots q_k$ is seen as a word in Q^*, that has to be of length $= rank(f)$ in the ranked case. So the extension is easy to the unranked case: suffices to define the transitions to be of the form $f(\omega) \to q$, where $\omega \in Q^*$, and $f \in \Sigma$. A TA is said to be bottom-up *deterministic* iff whenever there are two transition rules of the form $f(\omega) \to q$, $f(\omega') \to q'$, with $q \neq q'$, we have necessarily $\omega \cap \omega' = \emptyset$; otherwise it is said to be *non-deterministic*. We also agree to denote the transitions of the form $f(\emptyset) \to q$ simply as $f \to q$, and refer to them as *initial* transitions.

Example. We come now to our second example: to the right of the figure below is the tree format of a document, of which the fully compressed dag format is to the left. Over the unranked signature $\{a, f, g\}$ we consider the bottom-up TA **A**, with the following transitions:

$a \to p, \quad b \to p, \quad b \to q,$
$a(p) \to q, \quad a(q) \to p,$
$g(q\,Q^*) \to q, \quad g(p\,q) \to p,$
$f(q\,p\,q) \to q_{fin}, \quad f(p\,Q^*) \to q_{fin},$

with $Q = \{p, q, q_{fin}\}$, q_{fin} being the unique accepting state.



It is not hard to check that there is an accepting run of the TA on the tree to the right. And if we want to run the TA directly on the dag to the left, we may *have* to determinize the run on the fly, as and when we move up, in general. This can be done more elegantly, and more naturally, by seeing the dag as its SLR, and by seeing its productions as well as the transitions of the TA as rewrite rules, and using innermost rewriting. For instance, the SLR for

the dag to the left above is as follows:

$X_0 \to f(X_1, X_1, X_2)$
$X_1 \to g(X_3, X2)$
$X_3 \to a(X_2), \quad X_2 \to b$

We get by innermost rewriting:

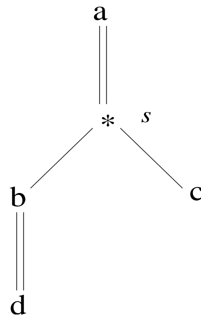
$X_2 = b \to \{p, q\}, \quad X_3 = a(X_2) \to \{p, q\}, \quad X_1 \to g(\{p, q\}\{p, q\}),$
and finally: $X_0 \to f(\{p, q\}\{p, q\}\{p, q\})$.

And we end up by checking that q_{fin} is in the set $f(\{p, q\}\{p, q\}\{p, q\})$. □

This is how one would proceed, in general, for deciding acceptance under the bottom-up runs of non-deterministic TAs on dags. However, to any query defined by a pattern of a limited format that we shall be considering below, we shall naturally associate a bottom-up non-deterministic TA, called its query automaton (QA); for such QA, the selecting runs can be computed more directly (without on the fly determinizations).

3 Query Automata for Patterns

We henceforth assume known the usual notions and terminology of XML, and of XPath which is a language generally employed for querying XML documents. For simplicity, we only consider unary queries, i.e., a queries that return a set of nodes and/or the data attached to these nodes. We are interested here in a limited sub-class of XPath expressions which can also be seen naturally as a non-deterministic bottom-up QA. These expressions are all representable as 'patterns with selection labels' with the help of the symbols $/, //, *$ of XPath, along with those from the (unranked) base alphabet Σ, and the filter expressions of XPath. For instance, the XPath expression $a//*[b//d][c]$ is represented as the following pattern, where a, b, c, d are in the base alphabet, '$*$' is the wildcard, and s is the selection symbol:



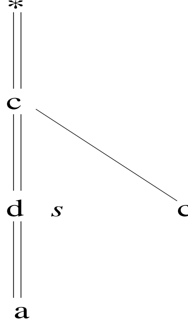
A node u on any given document t is an answer to the query of this pattern, iff:
- the root node of t bears the name a, and u is a descendant of the root,
- and u has two children nodes b and c, the child named b itself having a descendant d.

The XPath expressions that we are interested in here, can be defined formally in terms of simple grammars; for instance, as in [15], one can define them by:

$$P ::= \quad \sigma \mid * \mid . \mid P/P \mid P//P \mid P[P]$$

where $\sigma \in \Sigma$, '$*$' is the wildcard of XPath, and '.' stands for the current node position. To each such expression, one can associate a *pattern tree* (pattern, for short), as in the above example (cf. [15] for details). A pattern is essentially a (rooted) tree with usual edges, plus some distinguished "double-edges" (as in the above example), referred to as descendant-edges; the nodes are named from $\Sigma \cup \{*\}$; in addition, to exactly one of the nodes is assigned a selection symbol: 1 or s. (Note: we are only concerned here with unary queries.) An essential difference between the usual trees and patterns is that the "set of outgoing nodes" from a node, on a pattern, are *not ordered*. (Thus, in the above pattern example, it is *not* required that the b-child, of the node u to be selected, be to the left of its c-child.)

A point we want to drive in now, is that to any pattern can be associated a QA in a natural manner. We shall do that only on an example, with which we will continue in the next section. Consider, for instance, the following pattern:



This pattern represents the following XPath expression: $//c[c]//d[.//a]$, which is short for: $//c[child : c]//d[decendant : a]$. The QA associated with this pattern is the bottom-up *non-deterministic* TA over the alphabet $\{a, c, d, *\}$, with $Q = \{q_{in}, q_0, q_1, q_2, q_3 q_{acc}\}$ as its set of states, q_{acc} as the accepting state, q_1 as the *selecting* state, and the following transitions:

$* \to q_{in}$	$c(Q^* q_{in} Q^*) \to q_2$
$*(Q^* q_{in} Q^*) \to q_{in}$	$c(Q^* q_1 Q^* q_2 Q^*) \to q_3$
$a \to q_0$	$c(Q^* q_2 Q^* q_1 Q^*) \to q_3$
$a(Q^* q_{in} Q^*) \to q_0$	$*(Q^* q_3 Q^*) \to q_3$
$*(Q^* q_0 Q^*) \to q_0$	$*(Q^* q_3 Q^*) \to q_{acc}$
$d(Q^* q_0 Q^*) \to q_1$	$*(Q^* q_{acc} Q^*) \to q_{acc}$
$*(Q^* q_1 Q^*) \to q_1$	

(The Q^* stands for any string over the set Q.) The semantics of the transitions are as follows: at any leaf node other than a-named we would be in state q_{in}; when we reach an a-node we would be in state q_0; when we reach a d-node above

a we would be in state q_1; at a c-node, 'in general' we would be in q_2, but if it is above a d-node, we would be in q_3; finally, at any node strictly above a "c-node reached at state q_3", we would be in the accepting state q_{acc}. (Note: a query automaton returns a selected node only on an accepted document.)

4 Querying under SLR or under SLCF

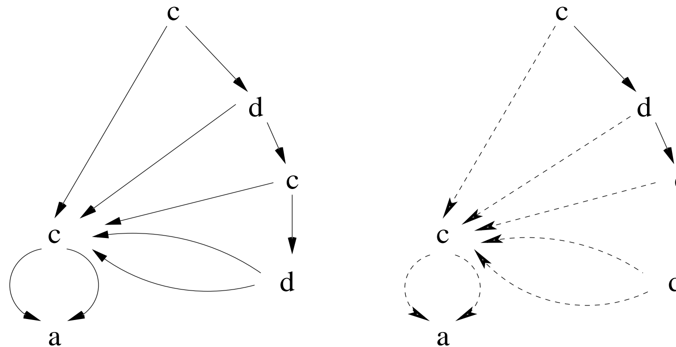
We show here, briefly, how the query $//c[child : c]//d[decendant : a]$ can be evaluated on the following 'document':

$$t = c(c(a, a), d(c(a, a), c(c(a, a), d(c(a, a), c(a, a))))),$$

under the runs of the QA defined in the previous section. The example has been borrowed from [3]; as was shown there, the following SLCF – with $S, A, B(y), C$ as non-terminals, S being the axiom – gives a 'nice' compression of this tree:

$$\begin{array}{l} S \to B(B(C)) \\ B(y) \to c(C, d(C, y)) \\ C \to c(A, A) \\ A \to a \end{array}$$

Observe first that our QA, if run on the tree t, would select the subterm $d(c(a, a), c(a, a))$, the position of which (on t) can be deduced by following the path from the d-node to which is assigned the selecting state q_1, to the root.



Now, the dag to the left of the figure above is the fully compressed dag format for t; and it is not hard to check that our QA can run bottom-up just as well on this dag (*without needing* any determinization on the fly), as on the tree t; the dag will be accepted, and the d-node to the bottom-right will be assigned the selecting state q_1; the set of positions on a dag being well-defined, one deduces easily the position(s) of the selected node: it corresponds here to the path formed of the full arrows in the dag to the right. It is easy then to 'output' the sub-dag at that position (or its SLR), as the answer to our query.

On the other hand, our QA can also 'run' on the SLCF compressed format given above for the document t; we will then get the following:

- there is an innermost rewrite derivation from the axiom S to the state q_{acc};
- and *one* of the intermediary terms thus derived gets selected.

In other words, we can conclude that the query defined by the pattern does have an answer, 'somewhere'. It could also be possible to say 'exactly where', using some additional and more intricate syntax, based on the BPLEX algorithm of [3, 4]. But, although BPLEX is 'bottom-up multiplex', it is not clear if it can *output the exact* SLCF grammar *corresponding to the answer* of the query; more precisely, it does not seem simple to relate the SLCF grammar of the answer to the SLCF of the given document.

Before we close this section, a few observations are perhaps in order:

i) Several of the usual XPath based queries on XML documents – or their skeletons – can be visualized as pattern trees.

ii) Bottom-up *unranked* query automata can be derived naturally, from such patterns.

iii) These unranked query automata are (not easy to determinize, but they) run bottom-up on compressed dags, just as easily as on their corresponding uncompressed trees; and the answer sets obtained correspond.

iv) Compression based on SLR, and the bottom-up evaluation technique described above for queries defined by our patterns, can both continue to function – without any major modifications – on documents that are subject to rule-based access control policies (RBAC). It is not clear if the same holds for SLCF-based compression.

5 Conclusion

We have tried to show that although compression under SLR ensures at best only a single exponential space optimization, it has a more query-friendly behavior on several fronts, than compression based on SLCF. We have also shown that non-deterministic, bottom-up, tree automata are natural and intuitive candidates for fully evaluating a certain class of queries in the XPath format, visualizable as patterns, even on documents compressed as DAGs (it is actually the formulation as a pattern that gives the clue for deriving an automaton corresponding to the query). The pattern-based view for queries seems to have other advantages as well: for instance, n-ary queries can be handled quite naturally in that set up. Moreover, the patterns of the format studied here can also be directly defined in terms of a suitable grammar; this is of help in handling other problems, such as that of pattern containment, via rewrite techniques (cf. e.g., [8]).

References

[1] S. Anantharaman, P. Narendran, M. Rusinowitch, *Closure Properties and Decision Problems of Dag Automata*, In Information Processing Letters, 94(5):231–240, 2005.

[2] P. Buneman, M. Grohe, C. Koch, *Path queries on compressed XML.* In Proc. of the 29*th* Conf. on VLDB, 2003, pp. 141–152, Ed. Morgan Kaufmann.

[3] G. Busatto, M. Lohrey, S. Maneth, *Grammar-Based Tree Compression*. EPFL Tech. Report IC/2004/80, `http://icwww.epfl.ch/publications`.

[4] G. Busatto, M. Lohrey, S. Maneth, *Efficient Memory Representation of XML Documents*. In Proc. DBPL'05, LNCS 3774, pp. 199–216, Springer-Verlag, 2005.

[5] W. Charatonik, *Automata on DAG Representations of Finite Trees*, Technical Report MPI-I-99-2-001, Max-Planck-Institut für Informatik, Saarbrücken, Germany.

[6] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi, *Tree Automata Techniques and Applications*, `http://www.grappa.univ-lille3.fr/tata/`

[7] B. Fila, S. Anantharaman, *Automata for Positive Core XPath Queries on Compressed Documents*, In Proc. of the Int. Conf. LPAR-13, pp. 467–481, LNAI 4246, Springer-Verlag, November 2006.

[8] B. Fila-Kordy, *A Rewrite Approach for Pattern Containment*, In Proc. WADT'2008, Pisa, June 2008.

[9] M. Frick, M. Grohe, C. Koch, *Query Evaluation of Compressed Trees*, In Proc. of LICS'03, pp. 188–197, IEEE,

[10] G. Gottlob, C. Koch, *Monadic Queries over Tree-Structured Data*, In Proc. of LICS'02, pp. 189–202, IEEE.

[11] G. Gottlob, C. Koch, *Monadic Datalog and the Expressive Power of Languages for Web Information Extraction*, In Journal of the ACM, 51(1):12–28, 2004.

[12] G. Gottlob, C. Koch, R. Pichler, L. Segoufin, *The complexity of XPath query evaluation and XML typing* In Journal of the ACM 52(2):284-335, 2005.

[13] W. Martens, F. Neven, *On the complexity of typechecking top-down XML transformations*, In Theoretical Computer Sc., 336(1): 153–180, 2005.

[14] M. Marx, *XPath and Modal Logics for Finite DAGs*. In Proc. of TABLEAUX'03, pp. 150–164, LNAI 2796, 2003.

[15] G. Miklau, D. Suciu, *Containment and Equivalence for a Fragment of XPath*, In Journal of the ACM, 51(1):2-45, 2004.

[16] F. Neven, *Automata Theory for XML Researchers*, In SIGMOD Record 31(3), September 2002.

[17] F. Neven, T. Schwentick, *Query automata over finite trees*, In Theoretical Computer Science, 275(1–2):633–674, 2002.

[18] A. Potthof, S. Seibert, W. Thomas, *Nondeterminism versus determinism of finite automata over directed acyclic graphs*, In Bull. Belgian Math. Society, 1:285–298, 1994.

[19] J.W. Thatcher, J.B. Wright, *Generalized finite automata theory with an application to a decision problem of second-order logic*, In Math. Syst. Theory, 2(1):57–81, 1968.