CTL Model Checking for Processing Simple XPath Queries

Loredana Afanasiev[1] Massimo Franceschet[1,2] Maarten Marx[1] Maarten de Rijke[1]
[1]*Informatics Institute, University of Amsterdam*
lafanasi,francesc,marx,mdr@science.uva.nl
[2]*Department of Sciences, University "G. d'Annunzio" of Chieti-Pescara*
m.franceschet@unich.it

Abstract

The Extensible Markup Language (XML) was designed to describe the content of a document and its hierarchical structure, and the XML Path language (XPath) is a language for selecting elements from XML documents. There is a close connection between the query processing problem for XPath and the model checking problem for temporal logics. Both boil down to checking which nodes of a graph satisfy a property. We investigate the potential *of a technique based on Computation Tree Logic (CTL) model checking for evaluating queries expressed in (a subset of) XPath. To this aim, we isolate a simple fragment of XPath that is naturally embeddable into CTL. We report on experiments based on the model checker NuSMV, and compare our results with alternative academic XPath processors. We comment on the advantages and drawbacks of the application of our model checking-based approach to XPath processing.*

1. Introduction

Much of the data on the Web is unstructured and unorganized, and regular schemas underlying conventional databases are simply inappropriate for this type of data. The Extensible Markup Language (XML) [26] is a textual representation of information that was designed by the World Wide Web Consortium. It is able to represent missing or duplicated data as well as nested information [1], and, hence, provides a suitable data model for semistructured data.

The XML Path Language (XPath) is a query language for selecting elements from XML documents [27]. XPath and its query evaluation problem have become an active research area that has proved particularly appealing for computational logicians. Query evaluation and containment problems have been recast in terms of reasoning problems in suitable logical languages (cf. Section 2 for details). Our objective is to investigate the *potential* of a technique for processing queries expressed in XPath that is based on model checking Computation Tree Logic formulas. To this end, we focus on a simple fragment of Core XPath, the logical core of XPath defined in [16]. This fragment, called *Simple XPath*, allows queries in which only child and descendant axes are admitted and such that the root of the tree is selected if, and only if, the query is successful. Such queries have been investigated in the context of XML data dissemination, where they are referred to as *tree pattern queries* [11]. In this context, consumers specify their subscriptions, indicating the type of XML content that is of interest, using some XML pattern specification language like XPath. For each incoming XML document, a router matches the document contents against the set of subscriptions to identify the interested consumers, and then routes the document to them. Thus, the result of matching a document with respect to a subscription is either 'yes' (the document matches the subscription and should be posted to the interested consumer) or 'no' (the document does not match the subscription and hence is not interesting for the consumer).

We provide an embedding of Simple XPath into Computation Tree Logic (CTL) [13], a specification language in the model checking framework [14]. The existential fragment of CTL featuring temporal operators **EX** and **EF** turns out to be sufficient to embed Simple XPath. A CTL model checker, such as NuSMV [12], can then be exploited to check a Simple XPath query against an XML document. The model checker answers 'yes' (the formula representing the query matches the model representing the document) or 'no' (the formula does not match the model). In the nega-

tive case, NuSMV provides an explanation for the failure of the matching.

Our main contributions are the following:

1. We devise a linear translation of the query evaluation problem for Simple XPath into the model checking problem for CTL. The translation is sound: a query matches an XML document if, and only if, the translated formula matches the translated model.
2. We describe an implementation of a Simple XPath processor that captures the above translation and invokes the NuSMV model checker to check whether the XPath query matches the XML file.
3. We report on tests on different XML files and queries, and on comparisons between our outcomes and those of alternative implementations of XPath. Our experiments use XML files generated with XMark [23], an XML document generator.
4. We extend the translation to *relative* Simple XPath queries. These are Simple XPath queries that need to be checked at *each* node of the XML tree, selecting a node if, and only if, the query is successful at that node. Hence, the result of a relative query is an arbitrary set of nodes, not just the root as for absolute queries. We extend the implementation on top of NuSMV and run experiments on relative queries as well.

Our approach to XPath query processing is of interest both to researchers in semistructured data and to researchers in temporal logic, yielding many new questions and challenges, some of which we point out below.

The rest of the paper is organized as follows. Section 2 discusses related work. In Section 3 we introduce background knowledge about XPath, and we give the embedding of Simple XPath into CTL. We test the feasibility of the method in Section 4 and conclude in Section 5.

2. Related Work

Since the mid-1990s there has been a lot of work on the interface of computational logic and semistructured data, making use of a wide variety of logical tools and techniques. For instance, simulations and morphisms [7], regular expressions [2], and connections with description logic [10] have all been used. Early work in the area was (necessarily) mostly concerned with proposals for data models and query languages for semistructured data. After the introduction of XML and XPath, this is where much of the research converged.

The relation between model checking and query processing has been extensively explored in the setting of *structured* data. We refer to [17] as a good entry-point to the area. The relation between model checking and query processing for *semistructured* data goes back at least to [3], where it was formulated in terms of suitable modal-like logics. Quintarelli [22] embeds a fragment of the graphical query language G-Log into CTL, and sketches a mapping for subsets of other semistructured query languages, like Lorel, GraphLog and UnQL. De Alfaro [5] proposes the use of model checking for detecting errors in the structure and connectivity of web pages. Alechina et al. [4] discuss the use of Propositional Dynamic Logic (rather than CTL) to obtain decidability and complexity results for checking path constraints on semistructured data (rather than query evaluation). Calvanese et al. [10] use description logics for similar purposes. Finally, Miklau and Suciu [20] and Gottlob et al. [15] sketch an embedding of the forward looking fragment of XPath into CTL, but they do not test the practical effectiveness of this approach.

As noticed by Gottlob et al. [16], many commercial engines implement XPath processing by adopting a naive exponential-time strategy even though the query processing problem for XPath admits a polynomial-time algorithm, and it can be solved in linear time in case of Core XPath, the navigational fragment of XPath. A number of Core XPath processors have so far been implemented. In [16] the authors propose an algorithm that embeds a Core XPath query into an algebraic expression over sets of nodes of the tree representing an XML document, and then evaluates the algebraic expression in order to process the query. Buneman et al. [8] propose an algorithm for Core XPath processing on compressed XML files; the idea is to compress XML trees into directed acyclic graphs sharing common subtrees, and to evaluate the query directly on compressed XML documents. Koch [18] embeds a query into a tree automaton and runs the resulting tree automaton in order to process the query. A alternative approach is to embed XML documents into *relational* databases, to rewrite XPath queries as SQL ones, and to run an SQL engine to retrieve the answer set of the original XPath query [24]); the advantage of this approach is clear: it taps into sophisticated relational database technology.

Tree pattern queries, similar to the ones we use in this paper, have already been investigated in [6], where the authors propose query minimization algorithms, in [11], where the authors devise algorithms to aggregate an input set of queries into a smaller set such that a given space constraint is met and the loss of precision is minimal, and in [20], where the authors study the corresponding containment and equivalence problems.

3. Embedding Simple XPath Queries into CTL

We start by providing background on XML, XPath, and Simple XPath. After that we define an embedding from Simple XPath into CTL. We assume that the reader is familiar with the syntax and semantics of Computation Tree Logic (CTL) (see, e.g., [14]). We will only need the fragment of CTL that contains the temporal operators **EX** and **EF**. Given a CTL model M and formula α, the *truth set* of formula α with respect to model M is $\{s \in M \mid M, s \models_{\mathrm{CTL}} \alpha\}$.

An XML document can be easily represented as a tree with tag labels attached to the nodes [1]. We define the tree representation of an XML document as follows. Let Σ be a set of labels corresponding to the XML tags containing the special label $*$. An *XML tree* is a node-labeled tree $T = (N, R_\downarrow, L)$, where N is the finite set of nodes containing the root node r, $R_\downarrow \subseteq N \times N$ is the set of tree edges, and $L : \Sigma \to 2^N$ is a node-labeling function associating to each label a set of nodes such that $L(*) = N$.

The *XML path language* (XPath) [27] is a language proposed by the World Wide Web Consortium (W3C) for selecting elements (or nodes) from XML documents. *Core XPath* [16] is the logical core of XPath, often referred to as the *navigational fragment* of XPath, since it maintains its navigational power in the four directions of the tree (forward, backward, right, and left), while omitting the arithmetical and string operations. Simple XPath is the following fragment of Core XPath. Let l be a label in Σ. *Predicates* $\mathtt{p}$ are defined by:

$$\begin{array}{rcl} \mathtt{p} & = & \mathtt{p\ and\ p} \mid \mathtt{p\ or\ p} \mid \mathtt{not\ p} \mid \mathtt{path} \\ \mathtt{path} & = & \mathtt{step} \mid \mathtt{step/path} \\ \mathtt{step} & = & \mathtt{axis} :: l \mid \mathtt{axis} :: l[\mathtt{p}] \\ \mathtt{axis} & = & \mathtt{self} \mid \mathtt{child} \mid \mathtt{descendant} \mid \\ & & \mathtt{descendant\_or\_self} \end{array}$$

A Simple XPath query is either an *absolute* query of the form `/[p]`, or a *relative* XPath query of the form `//[p]`. Intuitively, absolute queries are evaluated at the root of the tree, while relative ones are evaluated at each node of the tree.

We give the semantics of Simple XPath queries. The semantics is as in [16], which is in line with the standard XPath semantics from [25]. The semantics of a predicate $\mathtt{p}$ is specified by a function $[\![\cdot]\!]$, that, given an XML tree $T = (N, R_\downarrow, L)$ and a node $n \in N$, returns the set of nodes $[\![\mathtt{p}]\!]_{T,n}$ recursively defined as follows, where $(R_\downarrow)^+$ denotes the transitive closure of $R_\downarrow$ and $(R_\downarrow)^*$ is the reflexive and transitive closure of $R_\downarrow$:

$$\begin{array}{l} [\![\mathtt{axis} :: \mathtt{l}]\!]_{T,n} = \{m \mid (n,m) \in \{\!|\mathtt{axis}|\!\}_T \text{ and } m \in L(l)\} \\ [\![\mathtt{axis} :: \mathtt{l[p]}]\!]_{T,n} = \{m \mid (n,m) \in \{\!|\mathtt{axis}|\!\}_T \text{ and } m \in L(l) \text{ and } [\![\mathtt{p}]\!]_{T,m} \neq \emptyset\} \\ [\![\mathtt{step/path}]\!]_{T,n} = \{m \mid \exists k.\ k \in [\![\mathtt{step}]\!]_{T,n} \text{ and } m \in [\![\mathtt{path}]\!]_{T,k}\} \\ [\![\mathtt{p_1\ and\ p_2}]\!]_{T,n} = [\![\mathtt{p_1}]\!]_{T,n} \cap [\![\mathtt{p_2}]\!]_{T,n} \\ [\![\mathtt{p_1\ or\ p_2}]\!]_{T,n} = [\![\mathtt{p_1}]\!]_{T,n} \cup [\![\mathtt{p_2}]\!]_{T,n} \\ [\![\mathtt{not\ p}]\!]_{T,n} = N \setminus [\![\mathtt{p}]\!]_{T,n} \end{array}$$

$$\begin{array}{l} \{\!|\mathtt{self}|\!\}_T = \{(n,n) \mid n \in N\} \\ \{\!|\mathtt{child}|\!\}_T = R_\downarrow \\ \{\!|\mathtt{descendant}|\!\}_T = (R_\downarrow)^+ \\ \{\!|\mathtt{descendant\_or\_self}|\!\}_T = (R_\downarrow)^* \end{array}$$

Define the *answer set* of the absolute query `/[p]` with respect to a tree T to be the singleton $\{r\}$ containing the tree root if $[\![\mathtt{p}]\!]_{T,r} \neq \emptyset$, and the empty set otherwise. The query `/[p]` is *successful* if its answer set is non-empty. The answer set of the relative query `//[p]` with respect to a tree T is $\{n \in N \mid [\![\mathtt{p}]\!]_{T,n} \neq \emptyset\}$.

The translation from Simple XPath queries to CTL formulas is as follows. For $l \in \Sigma$, let $\hat{l} = \mathit{true}$ if $l = *$ and $\hat{l} = l$ otherwise. We define a linear embedding ω of predicates into CTL formulas as follows, where ϵ is the empty string:

$$\begin{array}{rcl} \omega(\mathtt{axis} :: \mathtt{l}) & = & \langle \mathtt{axis} \rangle \hat{\mathtt{l}} \\ \omega(\mathtt{axis} :: \mathtt{l[p]}) & = & \langle \mathtt{axis} \rangle (\hat{\mathtt{l}} \wedge \omega(\mathtt{p})) \\ \omega(\mathtt{axis} :: \mathtt{l/path}) & = & \langle \mathtt{axis} \rangle (\hat{\mathtt{l}} \wedge \omega(\mathtt{path})) \\ \omega(\mathtt{axis} :: \mathtt{l[p]/path}) & = & \langle \mathtt{axis} \rangle (\hat{\mathtt{l}} \wedge \omega(\mathtt{p}) \wedge \omega(\mathtt{path})) \\ \omega(\mathtt{p_1\ and\ p_2}) & = & \omega(\mathtt{p_1}) \wedge \omega(\mathtt{p_2}) \\ \omega(\mathtt{p_1\ or\ p_2}) & = & \omega(\mathtt{p_1}) \vee \omega(\mathtt{p_2}) \\ \omega(\mathtt{not\ p}) & = & \neg\omega(\mathtt{p}) \end{array}$$

$$\begin{array}{rcl} \langle \mathtt{self} \rangle & = & \epsilon \\ \langle \mathtt{child} \rangle & = & \mathbf{EX} \\ \langle \mathtt{descendant} \rangle & = & \mathbf{EXEF} \\ \langle \mathtt{descendant\_or\_self} \rangle & = & \mathbf{EF} \end{array}$$

The following result links XPath query processing to CTL model checking. The proof is by a straightforward induction.

Theorem 1 *Let $T = (N, R_\downarrow, L)$ be an XML tree with root node r and p a predicate. Then,*

1. *the absolute query $/[\mathrm{p}]$ is successful if, and only if, $T, r \models_{\mathrm{CTL}} \omega(p)$;*
2. *the answer set of the relative query $//[\mathrm{p}]$ with respect to T equals the truth set of the* CTL *formula $\omega(p)$ with respect to T.*

Given the link between XPath query processing and CTL model checking provided by Theorem 1, many questions arise, both regarding the CTL side of the link and regarding the XPath side. For instance, what *exactly* is the CTL fragment determined by the range of the translation ω? What is its expressive power? Its complexity? Because of space limitations, we have to leave these questions for future research. Instead, we want to devote much of the rest of the paper to the following question: how feasible is it to use CTL model checking for evaluating Simple XPath queries? More concretely, given Theorem 1 we can solve the Simple XPath query evaluation problem by taking an existing model checker for CTL, feeding it (the translations of) an XML document and a Simple XPath query to be evaluated against the document, and obtaining the answer set by computing the truth set for the corresponding model checking problem. How effective is this strategy in practice?

4. Evaluation

To address the question of how feasible it is in practice to deploy a CTL model checker as a query evaluation engine for Simple XPath, we carried out a number of experiments. The specific questions that we aimed to address with our experiments were:

1. What are the time requirements for translating XML documents into CTL models, suitable for processing by a model checker?
2. Does XMChecker perform similarly on absolute and relative queries? Note that, strictly speaking, relative queries are outside standard (local) CTL model checking.
3. How does model checking-based processing of translated Simple XPath queries compare to other tools for XPath query evaluation?

In this section we report on experiments carried out to address these questions. We describe our implementation, the test set used, and the results.

4.1. The Implementation

We implement our approach to Simple XPath query evaluation by using an existing CTL model checker, thus reducing our task to implementing the document and query translations presented in Section 3 and the translation from the model checker output back to the world of XML. The resulting tool, called XMChecker (XML Model Checker), uses NuSMV [12] as the CTL model checker. Besides the above translations, XMChecker implements a subroutine that runs NuSMV, and one that interprets the truth set of a CTL formula, returned by NuSMV, as a set of elements of the original XML file.

We used NuSMV because it is a state-of-the-art tool implementing many optimization techniques, including symbolic model checking, cone of influence reduction, and conjunctive partitioning of the transition relation [14]. Moreover, NuSMV is open source, structured modularly, and well documented [21]. However, designed for the verification of computer hardware and software, NuSMV only allows for *local* model checking, i.e., it checks whether a given formula holds at all initial states of a given model. For our purposes, we need to solve a slightly more general task, viz. *global* model checking: given a formula and a model, retrieve all states of the model that satisfy the formula. This functionality was easily implemented by making a small modification to NuSMV's source code.

The source code for XMChecker and for NuSMV, modified to perform global model checking, can be found at the XMChecker website [28].

4.2. Experimental Setting

Our experiments were run on a Pentium IV, 1.60GHz, with 1.5GB RAM, running Redhat Linux version 2.4.21-ict1. We ran tests using a variety of XML documents and (Simple XPath) queries. The documents and queries are generated using the XML benchmarking program XMark [23].

The XMark generated documents are modeled after a database as deployed by an Internet auction site, a typical e-commerce application. They allow for the formulation of queries that both feel natural and present concise challenges. The generated documents make the behavior of queries predictable. XMark provides an accurate scaling of the XML document size using a user defined scaling factor f. The numbers are calibrated to match

a total XML document size of approximately 100 MB when f assumes the value 1.0.

We used the following Simple XPath queries:

Q1 /[child::site/child::regions/child::africa/ child::item/ child::description/child::parlist/ child::listitem/child::text]

Q2 /[descendant::item/child::description/ child::parlist/ child::listitem/child::text]

Q3 /[descendant::item/descendant::text]

Q4 //[self::open_auction and child::bidder]

Q5 //[self::item and child::payment and child::mailbox]

Q6 //[self::person and descendant::payment]

Observe that Q1, Q2, and Q3 are absolute queries, while Q4, Q5, and Q6 are relative. For an absolute query we apply local model checking, which gives us a positive answer if the query is successful and a negative answer, otherwise. We apply the global model checking in order to obtain the answer set for a relative query. All the queries are designed to have non-empty answer sets.

We measured the CPU times (in seconds) needed to execute the following tasks: model checking ("query evaluation"), and converting the model into NuSMV's internal, symbolic representation. On the data just described we compared XMChecker's query evaluation time against the run-times of the following alternative XPath processors: `XMLTaskForce Engine`, the first polynomial-time XPath engine, proposed in [16], and `MacMill` [8], a fast XPath processor that works on compressed XML documents. To the best of our knowledge `MacMill` is the fastest implemented navigational XPath processor currently available.

4.3. Results

We first look at the model building times used by XMChecker. The time that XMChecker takes to translate the XML documents into NuSMV input models can be neglected for two reasons: the translation takes little time and can be done off-line.

Figure 1 contains the time taken by NuSMV to process the input model into its internal format. Observe that the numbers are extremely high and that the process takes non-linear (quadratic) time with respect to the size of the input.

Figure 2 contains the outcomes of the comparison between XMChecker, MacMill, and XMLTaskForce. Each figure contains, for a single query,

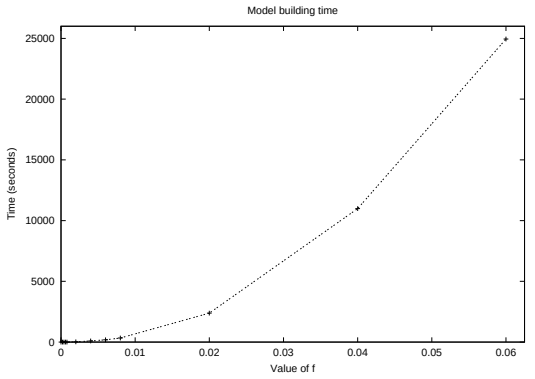

Figure 1. Processing CTL models into NuSMV's internal format. The XML document sizes corresponding to the values of f range from 26 KB to 6.6 MB.

the CPU times for the three query evaluation engines against increasing values of the scaling factor f. For these experiments, we used the tree representation of the XML documents in the signature of the six queries, i.e., labels that do not appear in the given queries were ignored during the translation. This was done to obtain the smallest possible model relative to the given queries. The same projection technique is implemented in MacMill [8].

All six queries Q1, ..., Q6 yield the same pattern: The time that XMChecker takes for query evaluation when the model and the query are in main memory are of the same order of magnitude as the time that MacMill takes to process the queries under the same conditions, while the XMLTaskForce engine takes one order of magnitude more time for the same task. Furthermore, for each of the engines, the relative queries Q4, Q5, Q6 are easier to process than the absolute queries Q1, Q2, Q3; for the values of f considered here, the differences between MacMill and XMChecker on the relative queries are very small.

Let us go back to the research questions formulated at the start of this section. As regards the first question, we found out that the model compilation time is rather high and hence the *run-time* translation of the model is not feasible in practice. We also carried out other experiments in which the complete tree representation of the document was translated, with no query projection optimization. In this case, the models grow larger since they contain all the tag information carried by the XML documents. However, as this processing is query

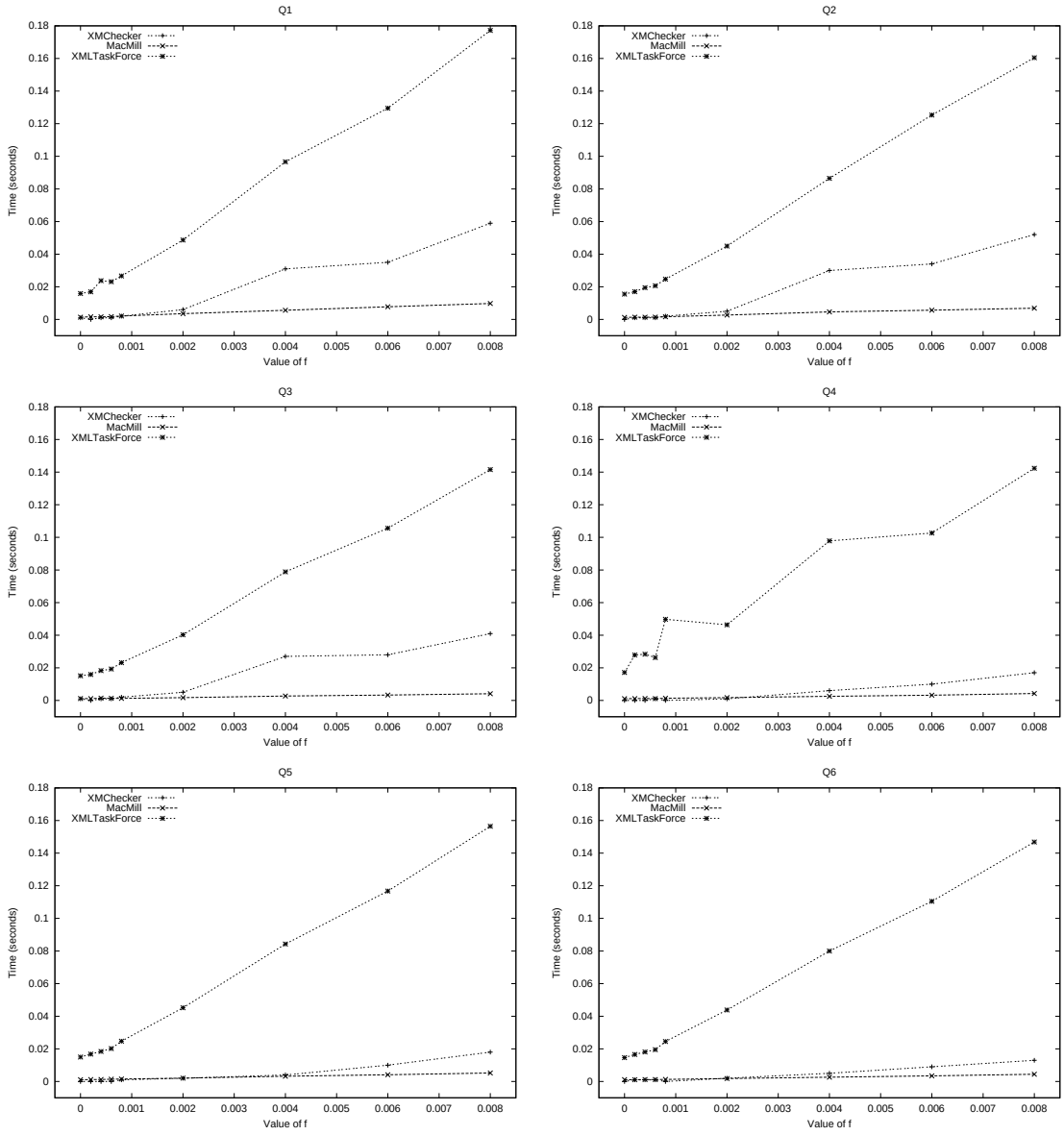

Figure 2. Run times for evaluating queries Q1, . . ., Q6 against documents with increasing scaling factor f, using XMChecker, MacMill, and XMLTaskForce.

independent, it can be done off-line. We found that the time needed by XMChecker to perform query evaluation is in the same order of magnitude as XMLTaskForce, that is, one order of magnitude larger than in the query dependent translation.

As to our second question, concerning the difference in behaviour of XMChecker on absolute and relative queries, the experiments show that our addition to NuSMV to do global model checking has no negative effect on performance. This is not surprising, since the original NuSMV implementation first computes the truth set of a formula, and then intersects it with the set of initial states. Our modification basically avoids the intersection and it prints the states in the truth set of the formula.

With respect to our third and final question, our experiments show that the XMChecker query evaluation times (when the model is in main memory) can compete both with the MacMill query evaluation engine (with document translation relative to the given queries) and with the XMLTaskForce query evaluation engine (with full translation of

the tree representation of XML documents).

Ignoring the model processing times, these results are very encouraging. In the next section we discuss possible improvements of the representation of the document trees.

5. Conclusions and Future Work

We tested the potential of currently available off-the-shelf CTL model checking technology for processing XPath queries on XML documents. To this aim, we implemented XMChecker, a processor for Simple XPath queries based on the NuSMV model checker, and we compared its performance with alternative XPath processors.

If we only consider the pure model checking time, XMChecker works very well: in several cases its performance is in the same league as that of MacMill, one of the fastest XPath processors currently available. XMChecker's real bottleneck is the long compilation time, i.e., the time taken by NuSMV to encode the high-level description of the XML document into a low-level OBDD-based one. The cause is not NuSMV itself, but the unconventional way in which we used NuSMV. NuSMV is a model checker designed to verify hardware and software systems. Their behaviour is usually specified as the parallel composition of the behaviours of components. The costly parallel composition is never explicitly performed. Hence, the implicit representation of the system is usually much smaller than the explicit one, sometimes even logarithmic in the size of its explicit representation. This has made it possible to verify systems whose explicit representation contains up to 10^{20} states [9]. In contrast, we provided NuSMV with explicit XML trees whose size is proportional to the size of the XML file. Encoding such trees into OBDD format takes a large amount of time, and is not manageable by NuSMV for relatively large XML documents.

We see many directions for future work, but list only some of them. First, there are questions on issues related to CTL, in particular to CTL model checking. By adding past operators to CTL language, that is, the backward looking analogues of **EX** and **EF**, it is possible to capture *all* relative forward XPath queries. How can we extend NuSMV with past temporal operators? For our application, we must do model checking on Kripke models that are given explicitly as labelled graphs. Is it possible to turn them directly into OBDD format in a much more efficient manner than currently done by NuSMV? It seems that NuSMV gets lost in optimizing the OBDD format for explicitly given Kripke structures. This may be because it is tuned to a very different input format. Preliminary investigations support this conjecture, as the OBDD format created by NuSMV is often much too large for the given Kripke structure. A different algorithm here could have a double positive effect: reducing the model compilation time, and — as the models get smaller — reducing the query evaluation times.

Next, there is a variety of questions related to XML and model checking. For a start XML documents are often redundant; in particular, the skeletal information contained in XML files is often repeated [8]. Our aim is to investigate alternative compact representations of XML files. Next, most of the queries only access a limited portion of the XML file [19]. Our goal is to study a systematic way to avoid processing XML regions that are irrelevant for the query, and to integrate it with XML compression. Finally, real XML documents (like XMark generated files) often are graphs (not trees) because of the use of ID/IDREF attributes. Query evaluation on graphs is not as easy to optimize as on trees, and has, in fact, hardly been addressed so far. Since model checkers are designed to work on cyclic structures, this seems a good area in which a model checking-based approach has potential.

Acknowledgments

Massimo Franceschet was supported by the Netherlands Organization for Scientific Research (NWO) under project number 612.000.207. Maarten Marx was supported by NWO under grant number 612.000.106. Maarten de Rijke was supported by NWO under project numbers 365-20-005, 220-80-001, 612.069.006, 612.000.106, 612.000.207, and 612.066.302.

References

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, Los Altos, CA 94022, USA, 2000.

[2] S. Abiteboul and V. Vianu. Regular path queries with constraints. *Journal of Computer and System Sciences*, 58(3):428–452, 1999.

[3] N. Alechina and M. de Rijke. Describing and querying semistructured data: Some expressiveness results. In S. Embury, N. Fiddian, W. Gray, and A. Jones, editors, *Advances in Databases*, LNCS. Springer, 1998.

[4] N. Alechina, S. Demri, and M. de Rijke. A modal perspective on path constraints. *Journal of Logic and Computation*, 2003.

[5] L. D. Alfaro. Model checking the world wide web. In G. Berry, H. Comon, and A. Finkel, editors, *Computer Aided Verification*, pages 337–349. Springer, 2001.

[6] S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, and D. Srivastava. Minimization of tree pattern queries. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 30(2):497–508, June 2001.

[7] P. Buneman, W. Fan, and S. Weinstein. Path constraints on semistructured and structured data. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 129–138, 1998.

[8] P. Buneman, M. Grohe, and C. Koch. Path queries on compressed XML. In *Proceedings of the International Conference on Very Large Data Bases (VLBD)*, 2003.

[9] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[10] D. Calvanese, G. D. Giacomo, and M. Lenzerini. Representing and reasoning on XML documents: A description logic approach. *Journal of Logic and Computation*, 9(3):295–318, 1999.

[11] C. Y. Chan, W. Fan, P. Felber, M. N. Garofalakis, and R. Rastogi. Tree pattern aggregation for scalable XML data dissemination. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 826–837. Morgan Kaufmann Publishers, 2002.

[12] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In *Proceedings of the International Conference on Computer-Aided Verification*, 2002.

[13] E. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal-logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[14] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[15] G. Gottlob and C. Koch. Monadic Queries over Tree-Structured Data. In *Logic in Computer Science*, pages 189–202, Los Alamitos, CA, USA, July 22–25 2002. IEEE Computer Society.

[16] G. Gottlob, C. Koch, and R. Pichler. Efficient Algorithms for Processing XPath Queries. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2002.

[17] J. Y. Halpern, R. Harper, N. Immerman, P. G. Kolaitis, M. Vardi, and V. Vianu. On the unusual effectiveness of logic in computer science. *The Bulletin of Symbolic Logic*, 7(2):213–236, 2001.

[18] C. Koch. Efficient processing of expressive node-selecting queries on XML data in secondary storage: A tree automata-based approach. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2003.

[19] A. Marian and J. Siméon. Projecting XML documents. In J. C. Freytag, P. C. Lockemann, S. Abiteboul, M. J. Carey, P. G. Selinger, and A. Heuer, editors, *Proceedings of International Conference on Very Large Data Bases VLDB*, pages 213–224. Morgan Kaufmann Publishers, 2003.

[20] G. Miklau and D. Suciu. Containment and equivalence for an XPath fragment. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 65–76, 2002.

[21] NuSMV: a new symbolic model checker. URL: `http://nusmv.irst.itc.it`.

[22] E. Quintarelli. *Model-Checking Based Data Retrieval: an application to semistructured and temporal data*. PhD thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano, 2002.

[23] A. R. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, and R. Busse. XMark: A Benchmark for XML Data Management. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 974–985, 2002. URL: `http://monetdb.cwi.nl/xml/`.

[24] J. Shanmugasundaram, H. Gang, K. Tufte, C. Zhang, D. J. DeWitt, and J. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 302–314. Morgan Kaufmann, 1999.

[25] P. Wadler. Two semantics for XPath. Technical report, Bell Labs, 2000.

[26] World Wide Web Consortium. Extensible markup language (XML). URL: `http://www.w3.org/XML`, 1998.

[27] World Wide Web Consortium. XML path language (XPath) version 1.0 – W3C recommendation. URL: `http://www.w3.org/TR/xpath.html`, 2000.

[28] XMChecker: an XML model checker. URL: `http://lit.science.uva.nl/Research/XMChecker`.