An Evaluation of Regular Path Expressions with Qualifiers against XML Streams

Dan Olteanu, Tobias Kiesling, François Bry
Institute for Computer Science, University of Munich, Germany
{olteanu,kiesling,bry}@informatik.uni-muenchen.de

Abstract

This paper presents SPEX, a streamed and progressive evaluation of regular path expressions with XPath-like qualifiers against XML streams. SPEX proceeds as follows. An expression is translated in linear time into a network of transducers, most of them having 1-DPDT equivalents. Every stream message is then processed once by the entire network and result fragments are output on the fly. In most practical cases SPEX needs a time linear in the stream size and for transducer stacks a memory quadratic in the stream depth. Experiments with a prototype implementation point to a very good efficiency of the SPEX approach.

1 Motivation

Querying data streams is motivated by applications like real time measurements and continuous services which select informations from continuous streams of data, e.g. stock exchange or meteorology data. For a selective dissemination of information, streams have to be filtered according to complex queries before being distributed to subscribers [2]. To integrate data over the Internet, particularly from sources with low throughput, it is desirable to progressively process the data before the full stream is retrieved [5]. Furthermore, the data streams considered in such applications can be infinite. Thus, traditional querying approaches based on parsing and buffering are not applicable. The messages of a data stream are conveniently modeled with XML and message selection is naturally expressed using regular path expressions with qualifiers.

2 Overview

SPEX stands for a **s**treamed and **p**rogressive **e**valuation of regular path e**x**pressions against wellformed XML streams. Streamed evaluation means that a data stream is not completely buffered, progressive processing means that results are streamed and delivered on the fly.

XML Streams and Query Language. Streaming an XML document corresponds to a traversal of the XML document in document order, i.e. a preorder traversal of the document tree. The document tree nodes correspond to stream messages. SPEX provides support for querying XML streams by means of regular path expressions [1] with qualifiers like those of XPath [9]. More precisely, the query language processed by SPEX subsumes the XPath fragment represented by `child` and `descendant` forward steps, union and intersection set operations and multiple and nested qualifiers. The qualifiers, i.e. value comparisons and structural conditions, do not create result, but rather condition the result. Any expression is allowed as a structural condition. E.g. the expression `root._*.a[a][b[c='text']].c`, where `_*` is a wilcard closure step, selects all `c` messages that are children of `a` messages that have (at least) an `a` child and a `b` child with a `c` child that has the value `text`. Furthermore, the backward steps `ancestor` and `parent` are treated. As established in [8], they are expressible in the aforementioned query language fragment. The addition of variables proves to be straightforward [7].

Translation to SPEX Networks. For each regular path expression construct, e.g. a step or a structural condition, a SPEX pushdown transducer is defined. A SPEX transducer is similar to a conventional **d**eterministic **p**ush**d**own **t**ransducer (DPDT), except that it does not have accepting states and that it has two stacks, i.e. it is a 2-DPDT. However, both stacks are updated in a synchronized manner, and most SPEX transducers can be reduced to 1-DPDT [7].

A regular path expression is translated into a network of interconnected SPEX transducers. A SPEX network is a directed acyclic graph (DAG), where each node consists in a SPEX transducer and an edge relates an output to an input tape of two successive transducers. The communication between transducers is done by having a transducer writing a message on its output tape, the next transducer reading that message from its input tape. The messages that stream into

Proceedings of the 19th International Conference on Data Engineering (ICDE'03)
1063-6382/03 $ 17.00 © 2003 IEEE

the network are on one hand the messages constituting the input stream, and on the other hand internal control messages needed e.g. for changing transducer states.

SPEX Network Evaluation. The evaluation of a SPEX network assumes a pipeline execution model, where each transducer processes a stream message before forwarding it to the next transducers. The message processing in some transducers can result in the creation of control messages. This is the case when specialized transducers, that convey the semantics of different regular path expression constructs, *match* the incoming stream message. E.g. an `a` child transducer `CH(a)` (corresponding to an `a` child step) can match an `a` message at a certain nesting level in the stream, as provided by one of its stacks, the *depth stack*. Then, a special control message, an *activation message*, is forwarded to the immediate next transducers, activating them, i.e. instructing them to try also to match. When such activation messages reach the last transducer in the network, then result candidates are considered.

The support for qualifiers enhance the aforementioned evaluation model. That is, the result candidates can be constrained by the fulfillment of certain conditions, expressed in the supported query language as qualifiers. During the evaluation, a condition is modeled as a boolean variable. A variable is set to `true` via a *determination message*, when its related condition is fulfilled. The composition of such variables using `and` and `or` connectors results in boolean formulas. Keeping track of formulas is done individually for each transducer by means of a *condition stack*. When a transducer matches, then it activates its successor transducers with an activation message containing the topmost formula from its condition stack. The newly activated transducers push the received formula on their condition stacks. Therefore, result candidates depend on the topmost formulas from the condition stack of the last-but-one transducer. The dependencies between candidates and boolean formulas can not be overcome, as there are situations in which result candidates depend on conditions that can be evaluated only based on characteristics of future stream messages.

The move from transducer chains to DAGs is supported by specialized transducers, that have two input tapes and two output tapes, respectively. In this way, branching and joining in the evaluation flow are enabled. Branching is needed e.g. for evaluating two expressions in parallel. Consider the regular path expression `root.a[b].c`. While trying to satisfy the `[b]` qualifier, one should look also for `c` messages, as under an `a` either `b` or `c` might be found. The joining of evaluation flows is convenient e.g. for compacting the network, avoiding redundancy, and eliminating duplicates. With the expression `root.(a*._* | b).d` a `d` message can be selected via an `a*._*`, or a `b`, or both, therefore making the evaluation flow joining desirable.

Example. Consider the regular path expression with qualifiers `root.a[b].c` which selects the `c` messages that are children of `a` messages, such that the `a`'s are children of the root message and each of them has at least a `b` child message. Evaluated against the XML stream `<$><a><c></c><b></b></a></$>`, the expression selects the `c` message. Figure 1 shows the corresponding SPEX network, where each box represents a SPEX transducer. The input transducer `IN` forwards into the network one message at a time and when it encounters the root message `$`, it activates with a `true` formula the next transducer. The child transducer `CH(a)` is activated and it tries to match `a` messages only at the level of the direct children of the root. Thus, it matches the next `a` message, and activates the next transducer `VC([b])` with the formula from the top of its condition stack. The variable creator transducer `VC([b])` creates condition variables for every activation it receives, here co_1, and sends a conjunction of the received formula and the new created variable: $co_1 \wedge$ `true`. The split transducer `SP` forwards the received activation to the child transducer `CH(b)` and to the join transducer `JO`. The join transducer lets the activation through and, for forwarding also the current stream message `a`, it waits for it to arrive on both input tapes, hence avoiding message duplication. The child transducer `CH(c)` is also activated. After having the current message `a` also processed by the other transducers following `CH(b)` in the branch, `JO` forwards the `a` to `CH(c)`. The next stream message `c` is not matched by any other transducer but `CH(c)` which activates the output transducer `OU` with the formula from the top of its condition stack, i.e. co_1. In this way, `OU` is notified that a result candidate is considered and this depends on the value of co_1, which is undetermined at this time. `CH(b)` tries to match without success and waits to reach the same nested level for a new matching attempt. The message corresponding to the closing tag of `c` is also buffered by `OU` and gives again the opportunity to `CH(b)` and `CH(c)` to match. Only `CH(b)` matches the new stream message `b` and activates the variable filter transducer `VF([b])` and then a variable determinant `VD` with the formula co_1. At this point, a determination message is generated, which sets the variable co_1 to `true`. This determination message reaches incrementally all the transducers positioned after `VD` and at `OU` the only candidate is considered result and is output. The left messages empty the transducer stacks.

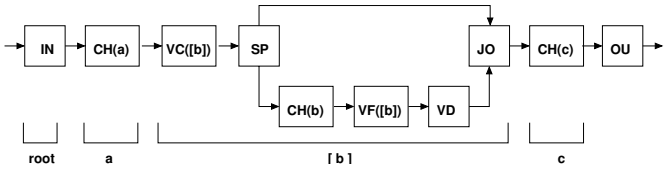

Figure 1. SPEX network for `root.a[b].c`

Proceedings of the 19th International Conference on Data Engineering (ICDE'03)
1063-6382/03 $ 17.00 © 2003 IEEE

3 Salient Features

A detailed description of SPEX is given in [7]. Some of its salient features are:

1. The translation of a regular path expression with qualifiers into a SPEX transducer network takes a time linear in the size of the expression. A SPEX transducer has a fixed number of states, its depth stack alphabet consists of a fixed number of symbols and its condition stack alphabet consists of run-time generated symbols, i.e. formulas. The condition stack is needed for evaluating expressions with qualifiers.

2. The evaluation of a SPEX network is performed in one pass over the stream and requires for every transducer stack a number of entries bounded in the depth d of the stream. In most practical cases it takes a time linear in the stream size and uses formulas with a size bounded in d. Without qualifiers or closure steps the size of a formula is constant. The evaluation of expressions with qualifiers on n wildcard closure steps can require formulas with size exponential in the number of such steps, i.e. d^n. The last transducer takes care of outputting ordered result and in the worst case it needs a memory linear in the stream size. However, it buffers messages only if their membership in the result can not be decided based on the stream fragment already processed.

3. The computational power needed by a transducer from a SPEX network, except its last transducer, is within the 1-DPDT class. Note that this is the lowest bound for the computational power needed for querying XML streams with regular path expressions [7]. The output transducer needs the computational power of a Turing machine.

4. Experiments with a prototype implementation point to a very good efficiency of SPEX. The prototype supports also other XPath navigational capabilities, i.e. following and preceding, and node-identity joins. Compared with existing XPath processors, SPEX overcomes them in most of the medium-sized scenarios and scales acceptable in cases when the other processors can not handle huge data, e.g. $\geq$ 1 GB [7]. The prototype was tested also against application-generated infinite streams and proved stable in cases where the depth of the tree conveyed in the stream is bounded.

4 Related Work

There are by now several proposals towards an efficient streamed evaluation of XML queries [5, 2, 4, 3]. SPEX adds to this common effort a formal framework and reasonable features, by keeping in the same time the expressiveness of the supported XML query language (at least) as powerful as of the above-cited proposals. To the best of our knowledge, SPEX is the only current approach that accomodates XPath qualifiers, closure and backward navigation.

The query operator X-Scan from the Tukwila data integration system [5] compiles regular path expressions into deterministic finite automata (DFA). The DFA reports to a host application when a node is reached in a final state. An improved version [4] considers an evaluation model based on the on-demand (lazy) creation of DFA. Both approaches use also stacks for keeping track of previous states. In [4] some expressions can be considered qualifiers, but their relations to the other expressions are left to a host application. SPEX is based on the evaluation of these connections between expressions inside and outside qualifiers and does not need extra logic for providing correct result. The XFilter [2] and YFilter [3] engines are used for deciding if entire XML documents are matched by XPath expressions that represent user profiles. Therefore, they are not focused on answering XPath expressions. YFilter [3] proposes also a basic multi-query optimization technique for reusing common prefixes of several queries. XSM [6] was developed in parallel to SPEX. Although both use a novel approach for a query execution plan based on transducer networks, their evaluation models, transducer types and query language features are quite different. SPEX is designed for low computational power and memory usage, which we consider essential in a stream-based context. It uses strongly coupled (i.e. without in-between queues) pushdown transducers and supports closure steps and qualifiers. XSM uses more general loosely coupled transducers with unbounded buffers and (therefore) supports value-based joins and element creation constructs.

References

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web. From Relations to Semistructured Data and XML.* Morgan Kaufmann, 2000.

[2] M. Altinel and M. Franklin. Efficient Filtering of XML Documents for Selective Dissemination of Information. In *Proc. 26th Int. Conf. on Very Large Data Bases*, 2000.

[3] Y. Diao, P. Fischer, M. J. Franklin, and R. To. YFilter: Efficient and Scalable Filtering of XML Documents. In *Proc. 18th Int. Conf. on Data Engineering*, 2002.

[4] T. J. Green, G. Miklau, M. Onizuka, and D. Suciu. Processing XML Streams with Deterministic Automata. Technical report, Univ. of Washington, 2002.

[5] A. Levy, Z. Ives, and D. Weld. Efficient Evaluation of Regular Path Expressions on Streaming XML Data. Technical report, Univ. of Washington, 2000.

[6] B. Ludäscher, P. Mukhopadhyay, and Y. Papakonstantinou. A Transducer-Based XML Query Processor. In *Proc. 28th Int. Conf. on Very Large Data Bases*, 2002.

[7] D. Olteanu, T. Kiesling, and F. Bry. An Evaluation of Regular Path Expressions with Qualifiers against XML Streams. Technical Report PMS-FB-2002-12, Univ. of Munich, 2002.

[8] D. Olteanu, H. Meuss, T. Furche, and F. Bry. XPath: Looking Forward. In *Workshop on XML-Based Data Management*, 2002. Springer LNCS 2490.

[9] W3C. XML Path Language (XPath) Version 1.0. W3C Recommendation, 1999. http://www.w3.org/TR/xpath.

Proceedings of the 19th International Conference on Data Engineering (ICDE'03)
1063-6382/03 $ 17.00 © 2003 IEEE