

Incremental View Maintenance for Active Documents

Serge Abiteboul Pierre Bourhis Bogdan Marinoiu

INRIA Futurs, Université de Paris Sud 11, ENS Cachan

4/10/2007

DocFlow Meeting



Overview

- 1 Introduction
- 2 Model
- 3 Incremental Maintenance
- 4 Query satisfiability
- 5 Useless Data
- 6 Conclusion

Plan

- 1 Introduction
- 2 Model
- 3 Incremental Maintenance
- 4 Query satisfiability
- 5 Useless Data
- 6 Conclusion

Motivation

Monitoring data evolving in the time

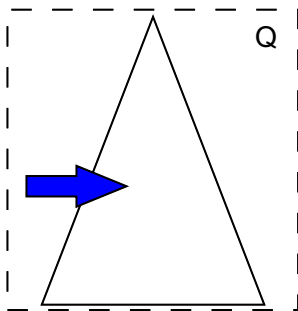
A user subscribes to a continuous query over data and receives results in order to monitor the document

- Active databases
- Aggregation of several streams : web page of newspapers e.g. “Le Monde”, iGoogle
- Monitoring of distributed systems
- Computing distributed queries

Goal

Incremental View Maintenance

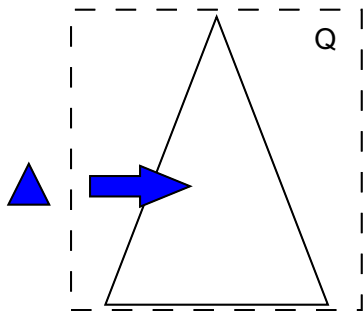
To compute $q(I_2(d)) - q(I_1(d))$ without computing $q(I_2(d))$ but by evaluating $q'(I_1(d), \Delta)$, where $q(I(d))$ is the view of the query q



Goal

Incremental View Maintenance

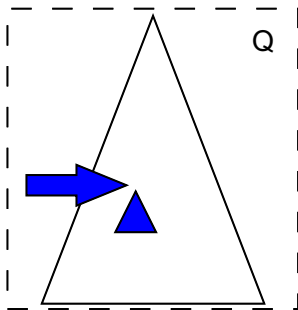
To compute $q(l_2(d)) - q(l_1(d))$ without computing $q(l_2(d))$ but by evaluating $q'(l_1(d), \Delta)$, where $q(l(d))$ is the view of the query q



Goal

Incremental View Maintenance

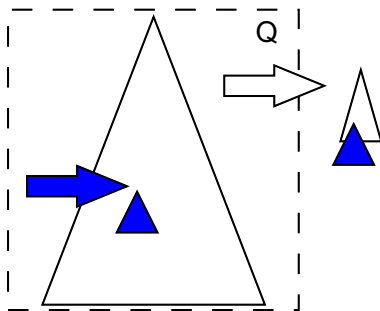
To compute $q(I_2(d)) - q(I_1(d))$ without computing $q(I_2(d))$ but by evaluating $q'(I_1(d), \Delta)$, where $q(I(d))$ is the view of the query q



Goal

Incremental View Maintenance

To compute $q(I_2(d)) - q(I_1(d))$ without computing $q(I_2(d))$ but by evaluating $q'(I_1(d), \Delta)$, where $q(I(d))$ is the view of the query q



Our approach

- Data structured in tree
- The evolution is restricted to adding subtrees
- Queries based on tree patterns

Plan

- 1 Introduction
- 2 Model**
- 3 Incremental Maintenance
- 4 Query satisfiability
- 5 Useless Data
- 6 Conclusion

Active Documents

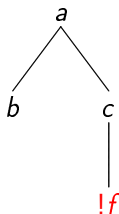
Active Document

- Tree where nodes are labeled by data (labels and XML PCDATA) or by functions (only the leaves for the functions)
- Update of an active document when a message comes from a stream
- Sequence of updates by integrating messages from several streams/data sources

Active Documents

Active Document

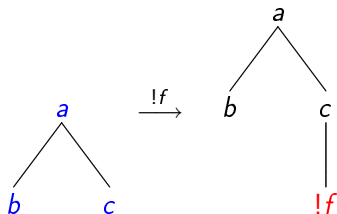
- Tree where nodes are labeled by data (labels and XML PCDATA) or by functions (only the leaves for the functions)
- Update of an active document when a message comes from a stream
- Sequence of updates by integrating messages from several streams/data sources



Active Documents

Active Document

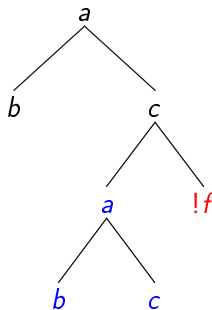
- Tree where nodes are labeled by data (labels and XML PCDATA) or by functions (only the leaves for the functions)
- Update of an active document when a message comes from a stream
- Sequence of updates by integrating messages from several streams/data sources



Active Documents

Active Document

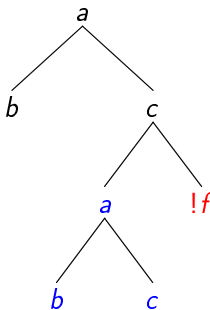
- Tree where nodes are labeled by data (labels and XML PCDATA) or by functions (only the leaves for the functions)
- Update of an active document when a message comes from a stream
- Sequence of updates by integrating messages from several streams/data sources



Active Documents

Active Document

- Tree where nodes are labeled by data (labels and XML PCDATA) or by functions (only the leaves for the functions)
- Update of an active document when a message comes from a stream
- Sequence of updates by integrating messages from several streams/data sources



Active Documents

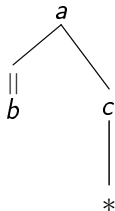
Some studied extensions :

- Typed documents
- Typed streams

Tree Pattern queries

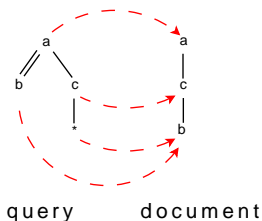
Boolean Tree Pattern query

is a tree where edges belong to relation $child (E_1)$ and $descendant E_{//}$. The nodes are labeled by data or $*$.



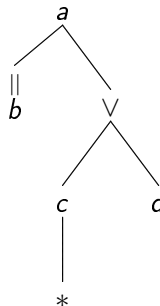
Query satisfaction

Query satisfaction : There exists a (non injective) function from the query q to the instance of the document respecting root, child, descendant relations and the labeling



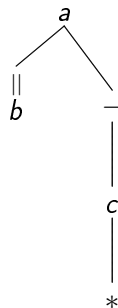
Other queries

- Disjunction :
- Negation :
- Return values
- Joins



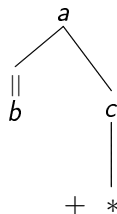
Other queries

- Disjunction :
- Negation :
- Return values
- Joins



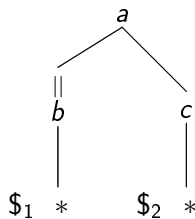
Other queries

- Disjunction :
- Negation :
- Return values
- Joins



Other queries

- Disjunction :
- Negation :
- Return values
- Joins



$$\$1 = \$2$$

Plan

- 1 Introduction
- 2 Model
- 3 Incremental Maintenance**
- 4 Query satisfiability
- 5 Useless Data
- 6 Conclusion

Our approach

Problem : some new data arrives on a stream

Optimization 1

Use incremental maintenance

Optimization 2

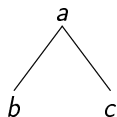
Use an optimization in the spirit of Magic Set.

Our approach

A tree pattern is translated into a nr-datalog program using the base relations :

- $root(n)$: The node n is root of the tree
- $par(n, n')$: The node n is the parent of the node n'
- $anc(n, n')$: The node n is an ancestor of the node n'
- $label_a(n)$: The node n is labeled a
- $function(n)$: The node n is labeled by a function

A simple instance



$$q() \leftarrow a(n)$$

$$\widehat{a}(n) \leftarrow \text{root}(n), \text{lab}_a(n)$$

$$\widehat{b}(n) \leftarrow \widehat{a}(n'), \text{par}(n', n), \text{lab}_b(n)$$

$$\widehat{c}(n) \leftarrow \widehat{a}(n'), \text{par}(n', n), \text{lab}_c(n), (b(n''), \text{par}(n', n''))$$

$$b(n) \leftarrow \widehat{b}(n)$$

$$c(n) \leftarrow \widehat{c}(n)$$

$$a(n) \leftarrow \widehat{a}(n), b(n'), c(n''), \text{par}(n, n'), \text{par}(n, n'')$$

Incremental Maintenance

- All sets computed for the evaluation of q are stored
- Rewriting of the nr-datalog program to propagate only the new updates $\tilde{p} = p \uplus \Delta(p)$

$$\begin{array}{ll}
 p(n) & \leftarrow q(n, n'), r(n', n'') \\
 \tilde{p}(n) & \leftarrow q(n, n'), r(n', n'') \\
 \tilde{p}(n) & \leftarrow \Delta(q)(n, n'), r(n', n'') \\
 \tilde{p}(n) & \leftarrow q(n, n'), \Delta(r)(n', n'') \\
 \tilde{p}(n) & \leftarrow \Delta(q)(n, n'), \Delta(r)(n', n'') \\
 \Delta p(n) & \leftarrow \Delta(q)(n, n'), r(n', n'') \\
 \Delta p(n) & \leftarrow \tilde{q}(n, n'), \Delta(r)(n', n'')
 \end{array}$$

Plan

- 1 Introduction
- 2 Model
- 3 Incremental Maintenance
- 4 Query satisfiability**
- 5 Useless Data
- 6 Conclusion

Goal

In the context of monotone queries and monotone streams, three cases are possible :

- 1 the document satisfies the query
- 2 the document does not never satisfy the query
- 3 the document can satisfy the query (**satisfiability**)

The incremental maintenance is needed only in the third case

Definition

Query satisfiability

An instance of document $l(d)$ can satisfy the query q , denoted $l(d) \models_{\diamond} q$ iff there exists an sequence of update ω such that $\omega(l)(d) \models q$.

Evaluation

In the case of monotone queries :

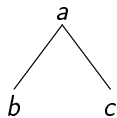
We use a 3 values based logic nr-datalog program $\{0, \frac{1}{2}, 1\}$:

$X \wedge Y = \text{Min}(X, Y)$, $X \vee Y = \text{Max}(X, Y)$, $\neg X = 1 - X$

The semantics of our program are

- True (1) The document satisfies the query
- May be ($\frac{1}{2}$) The document does not satisfy the query but may satisfy it in the future.
- False (0) The document does not and will never satisfy the query.

A simple instance



$$q() \leftarrow a(n)$$

$$b(n) \leftarrow \text{label}_b(n)$$

$$b(n) \leftarrow \text{func}(n), \frac{1}{2}$$

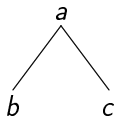
$$c(n) \leftarrow \text{label}_c(n)$$

$$c(n) \leftarrow \text{func}(n), \frac{1}{2}$$

$$a(n) \leftarrow \text{label}_a(n), b(n'), c(n''), \text{par}(n, n'), \text{par}(n, n'')$$

Optimization

The Magic Set optimisation is adapted to our context of logic with three values.



$$q() \leftarrow a(n)$$

$$\widehat{a}(n) \leftarrow \text{root}(n), \text{lab}_a(n)$$

$$\widehat{b}(n) \leftarrow \widehat{a}(n'), \text{par}(n', n), \text{lab}_b(n)$$

$$\widehat{b}(n) \leftarrow \widehat{a}(n'), \text{par}(n', n), \text{fun}(n), \frac{1}{2}$$

$$\widehat{c}(n) \leftarrow \widehat{a}(n'), \text{par}(n', n), \text{lab}_c(n),$$

$$(b(n'') \geq \frac{1}{2}), \text{par}(n', n'')$$

$$\widehat{c}(n) \leftarrow \widehat{a}(n'), \text{par}(n', n), \text{fun}(n), \frac{1}{2},$$

$$(b(n'') \geq \frac{1}{2}), \text{par}(n', n'')$$

$$b(n) \leftarrow \widehat{b}(n)$$

$$b(n) \leftarrow \widehat{b}(n), \text{fun}(n)$$

$$c(n) \leftarrow \widehat{c}(n)$$

$$c(n) \leftarrow \widehat{c}(n), \text{fun}(n)$$

$$a(n) \leftarrow \widehat{a}(n), b(n'), c(n''), \text{par}(n, n'), \text{par}(n, n'')$$

Complexity (1/2)

In the case of a simple active document :

Query	Complexity in the data	Complexity in the query
Boolean	LINEAR	LINEAR
Boolean with negation	PTIME	EXPTIME-Complete
Boolean with Joins	PTIME	NP-Hard
Boolean with Joins and Negation	Undecidable	Undecidable
Non-Boolean	PTIME	EXPTIME-Complete
Non-Boolean with Joins	PTIME	EXPTIME-Complete

Complexity(2/2)

Only with Boolean queries :

Type	Complexity in the data	Complexity in the query
typed Document	?	NP-Hard
non streams functions	PTIME	NP-Complete
Typed functions	LINEAR	NP-Complete

Plan

- 1 Introduction
- 2 Model
- 3 Incremental Maintenance
- 4 Query satisfiability
- 5 Useless Data**
- 6 Conclusion

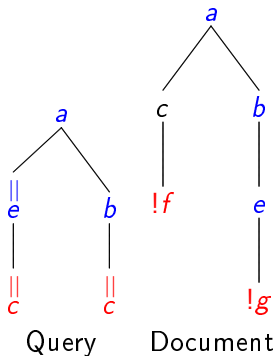
Objectif

Our goal

Checking the data that is not useful to answer the query

We focus here on useless functions

Instance



- The sequence $(!f, e[c]), (!g, c)$ transforms the instance in one satisfying the query
 - But the sequence $(!g, c)$ is enough for the satisfaction of the query
- Thus $!f$ is useless for this query.

Complexity et Algorithm

Complexity

Deciding usefulness is **NP-Complete** in the size of the query and **PTIME** in the size of the instance.

Algorithm : To find a symbolic representation of all sequences of update transforming the instance in an instance satisfying the query based on boolean formulae of tree patterns over streams.

- A first sufficient formula is built
- It is refined using checks of containment of tree patterns queries
- Only the functions appearing in the refined formula are useful

Plan

- 1 Introduction
- 2 Model
- 3 Incremental Maintenance
- 4 Query satisfiability
- 5 Useless Data
- 6 Conclusion

Implementation

Our theoretical approach is being implemented. There are some prototypes

- Transforms (d, q) into a P2PML plan, possible combinatorial explosion.
- Incremental evaluation in datalog is implemented for testing in storing all relations in memory

Other works

- The satisfiability in more complex case : non monotone streams, typed documents ...
- The useless data in a more general case

Thank you

Any question ?

