A Schema-Based Translation of XQuery Updates

Leonidas Fegaras
University of Texas at Arlington

Query Update Facility (XUF)

Extending XQuery with updates:

- Query Update Facility (XUF)
  - snapshot semantics
  - reorders pending updates

Possible Uses:

- persistent updates
- server-side programming
  - for XML data transformation
  - not necessary ...
  - ... but useful because most web programmers are unfamiliar with functional programming
- client-side programming
  - XQuery on the Web browser
  - to dynamically modify a web page content

Goal

Translate XQuery updates (XUF) to *effective* XQuery expressions

What is *effective*?

- amenable to optimization
- non-recursive
  - naive translation: use recursive XQuery functions to apply the updates to the qualified XML nodes
  - not very useful
- first-order (obviously)
  - cannot use higher-order state transformers during evaluation
    - eg, a state monad in Haskell

Our Approach

- A higher-order compilation scheme that generates first-order XQuery code
- Schema-based
- Based on the novel idea of *state complement (S-complement)*

Why Bother?

Good for:

- XQuery update optimization
  1. XUF → XQuery (this talk)
  2. XQuery optimization (earlier work)
  3. XQuery → XUF (to be explained later)
- incremental maintenance of materialized XML views
  - an XML view V on top of an XML database DB
  - XUF update on DB → XQuery U(DB)
  - $u(V) = view(U(view^{-1}(V)))$
  - XQuery u(V) → XUF
- XUF type-checking and validation
- an alternative XUF semantics

Disadvantages

Disadvantages (compared to in-place updates):

- node identity and document order must be explicitly propagated to new cells
- some space overhead
  - new data cells do not overwrite old ones
  - needs extra garbage collection to recycle old cells
- requires normalization
  - to fuse the generated layers of state transformation
- hard to address aliasing (updates on shared nodes)

Limitations

Current Limitations:

- schema-based
  - need to have the complete knowledge of the state schema
- no reordering of pending updates
  - but can be adjusted, if needed
- partial syntax coverage
  - but we are working on extensions

Mutable State

An XML sequence that consists of all XML data whose nodes are updated by XUF updates

Components: XQuery global variables that are used (directly or indirectly) in update destinations

```
declare variable $v as tp := doc("bib.xml");
```

Type tp is:

```
element bib { element article { element title string,
                                element year string,
                                element author string* }* }
```

```
for $i in $v/article[author="Smith"]
where contains($i/title,"XQuery")
return replace value of node $i/year with 2009
```

mutable state = the variable $v of type tp

Goal

Synthesize a plain XQuery that reconstructs the mutable state reflecting the updated values in the new state

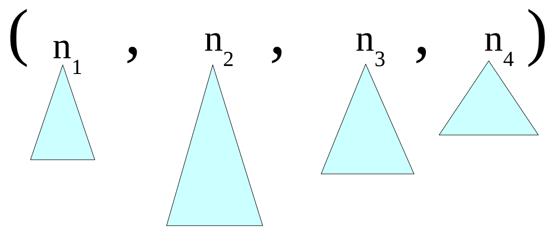
```
for $i in $v/article[author="Smith"]
where contains($i/title,"XQuery")
return replace value of node $i/year with 2009

<bib>{
    for $z in $v/article
    return if $z/author="Smith"
            then if contains($z/title,"XQuery")
                  then <article>{
                              $z/title,
                              <year>2009</year>,
                              $z/author
                        }</article>
                  else $z
            else $z  }</bib>
```

```
Type tp is:
element bib {
    element article {
        element title string,
        element year string,
        element author string* }* }
```

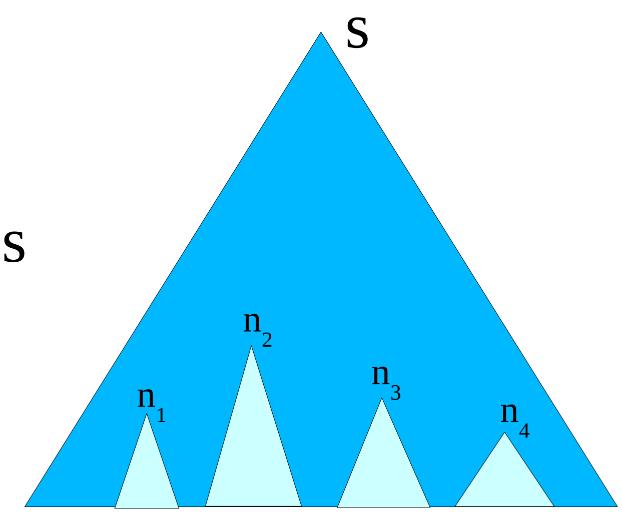
Input state is $v of type tp
Output state is of type tp

The S-Complement

- Let e be the destination of an XUF update (an XQuery expression)
- Let e be of type N* (where N = xs:node) and be evaluated to:



- These nodes n_i must be part of the state s



S-complement of e: <e>

- <e> is a function of type S → (N → N*) → S
- Given a state s: S and a *context mapping* u: N → N*,

<e> s u is equal to s but with each n_i replaced with $u(n_i)$

Updating e is finding the appropriate context mapping u

Example

S-complement:

< $v/article[author="Smith"]/year > s u

```
<bib>{
    for $z in s/article
    return if $z/author="Smith"
            then <article>{   $z/title,
                              u( $z/year ),
                              $z/author
                        }</article>
            else $z
}</bib>
```

For the update:

replace value of node $v/article[author="Smith"]/year **with** 2009

use: s = $v and u(x) = <year>2009</year>

The XUF Compiler

- We provide a compilation scheme that translates XUF to XQuery
  - the resulting XQuery calculates the S-complement of the XQuery result
- It uses an environment σ to bind XQuery variables to their XUF translation
- XUF translation: $C[e]\ \sigma\ s\ u$

  e: XQuery, σ: environment, s: state, u: context mapping
- The XQuery interpreter: $I[e]\ \rho$

  e: XQuery, ρ: environment
- *Lemma:* $\forall v \in \rho : \sigma[v] = < \rho[v] > \quad \Rightarrow \quad C[e]\ \sigma = < I[e]\ \rho >$
- *Theorem:* Our XUF compiler generates XQuery code that transforms the state in way consistent with the standard XUF update semantics (without operation reordering)

Some Rules

- C[**insert node** e_1 **as last into** e_2] σ s u = C[e_2] σ s u'

  assuming that e_2 is a sequence of type: element B t

  u'(x) = u(<A>{ x/node(), e_1 }</A>)

- C[**for** $v **in** e_1 **return** e_2] σ s u = C[e_2] σ' s u

  where σ' is σ extended with the binding from $v to C[e_1] σ

- C[e/A] σ s u = C[e] σ s u'

  assuming that e is a sequence of type: element B t

  if A=B ∨ A=*: u'(x) = u(x/self::A)

  otherwise: u'(x) = x/self::A

- C[(e_1,e_2)] σ s u = C[e_2] σ (C[e_1] σ s u) u

Various Extensions

- Hypothetical Queries
- Handling positional queries
  - The context mapping of the S-Complement must have the same type signature as the XQuery interpreter
  - eval: xquery currentContext position last

    $XQueryAST \rightarrow N \rightarrow Int \rightarrow Int \rightarrow N^*$
  - S-Complement: $S \rightarrow (N \rightarrow Int \rightarrow Int \rightarrow N^*) \rightarrow S$
- Addressing interference caused by aliasing
  - choose one replica as a primary copy
  - redirect updates on replicas to the primary copy
- Reordering the pending updates
  - organize updates into groups $\Rightarrow$ one state transformation per group

Final Thoughts

- Obviously, an XUF implementation based on in-place updates is faster than our approach ...
- ... but XQuery is easier to analyze than XUF
- Is it really necessary to pay the extra overhead?
- We can have our cake and eat it too
  - need an XUF → XUF optimization
  - XQuery → XUF is easier than XUF → XQuery
    - the XQuery must be a T → T function
    - normalize it
    - compare it to the identity function that copies T
    - for each non-matching part, generate an XUF update