NC STATE UNIVERSITY

ENGINEERING ONLINE

Lecture Notes

Course Number: CSC 513

Instructor: Dr. Singh

Lecture Number: 26

XQuery Quantification: 2

weakness of the language reduces portability

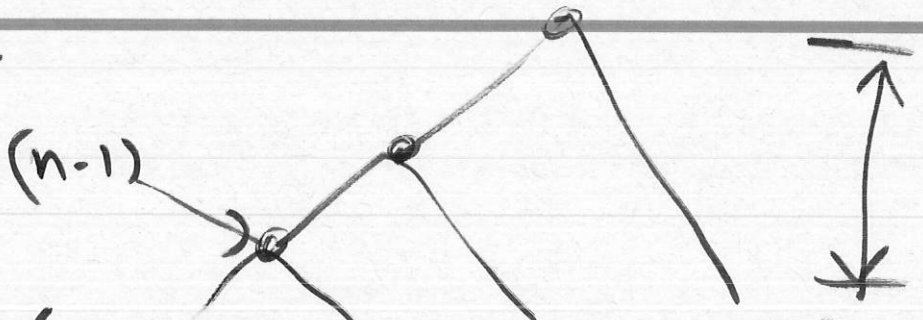
A typical useful quantified expression would use variables that were introduced outside of its scope

- The order of evaluation is implementation-dependent: enables optimization
- If some bindings produce errors, this can matter
- **some**: trivially false if no variable bindings are found that satisfy it
- **every**: trivially true if no variable bindings are found

Some : $\vee$ on a list

every : $\wedge$

Σ : +



$\Sigma_{\vee}(a_1\ a_2 \cdots a_n) \equiv (((a_1 \vee a_2) \vee a_3) \cdots a_n)$

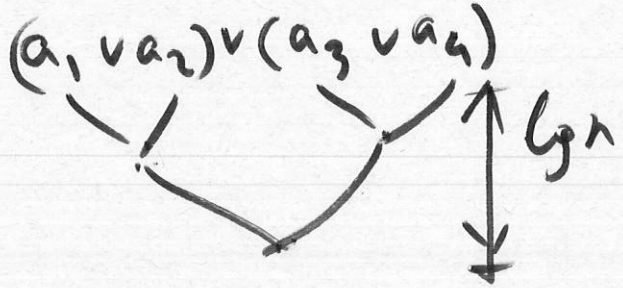
↳ REDUCE a list to a value

for $a in ($a_1, a_2, \cdots a_n$)
return f($a)

MAP

f(a_1),
f(a_2),
⋮
f(a_n)

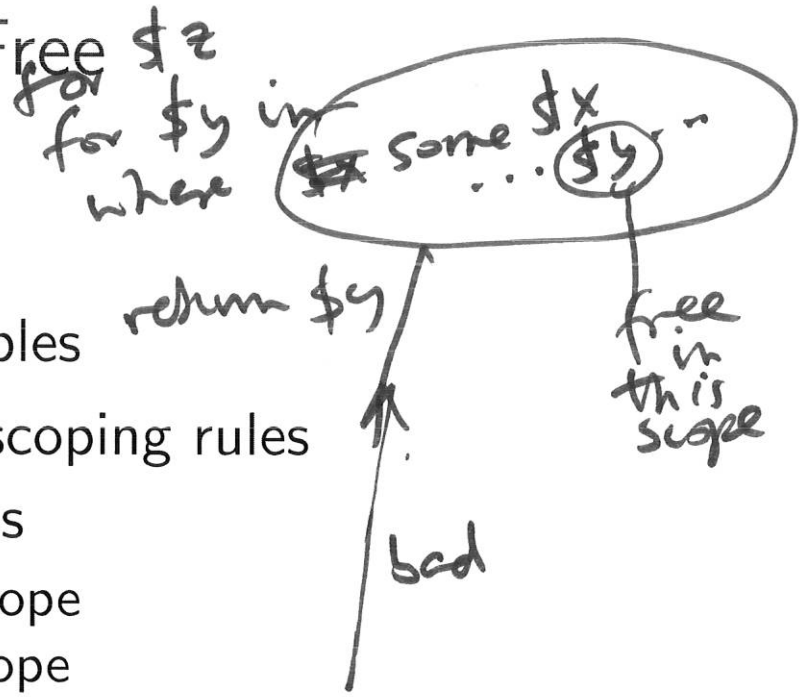
MAP REDUCE



$\vee, \wedge, +, *, \min, \max$

assoc
identity
commutative

Variables: Scoping, Bound, and Free



for, **let**, **some**, and **every** introduce variables

- The visibility variable follows typical scoping rules
- A variable referenced within a scope is
  - *Bound* if it is declared within the scope
  - *Free* if it not declared within the scope

```
for $x in ...
where (some $x in ...
satisfies ...
)
return ...$x
```

replace $x by $y uniformly - no effect

Here the two $x refer to *different* variables

XQuery Conditionals

Like a classical **if-then-else** clause

- The **else** is not optional
- Empty sequences or node sets, written ( ), indicate that nothing is returned

XQuery Constructors

Braces { } to delimit expressions that are evaluated to generate the content to be included; analogous to macros

- document { }: to create a document node with the specified contents
- element { } { }: to create an element
  - element foo { 'bar' }: creates <foo>Bar</foo>
  - element { 'foo' } { 'bar' }: also evaluates the name expression
- attribute { } { }: likewise
- text { body}: simpler, because anonymous

XQuery Effective Boolean Value

Analogous to Lisp, a general value can be treated as if it were a Boolean

- A **xs:boolean** value maps to itself
- An empty sequence maps to **false**
- A sequence whose first member is a node maps to **true**
- A numeric that is 0 or NaN maps to **false**, else to **true**
- An empty string maps to **false**, others to **true**

Quirks in XPath

XQuery versus XSLT: 2

- XQuery is geared for querying databases
  - Supported by major relational DBMS vendors in their XML offerings
  - Supported by native XML DBMSs
  - Offers superior coverage of processing joins
  - Is more logical (like SQL) and potentially more optimizable
- XSLT is geared for transforming documents
  - Is functional rather than declarative
  - Based on template matching

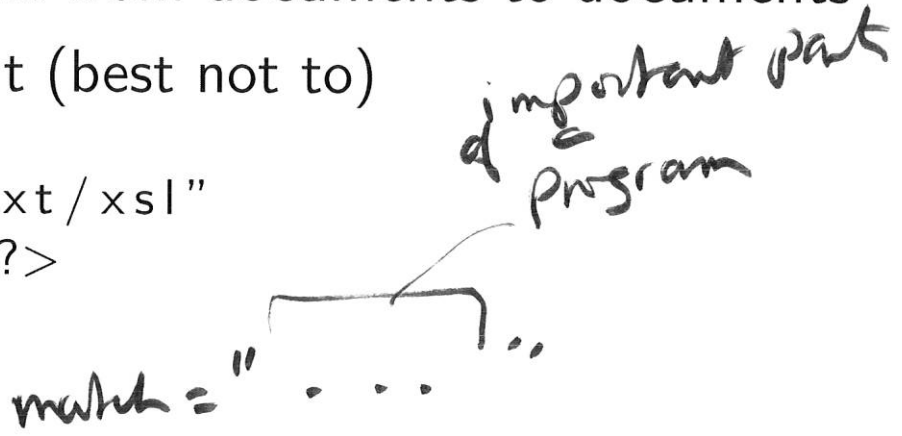
T jumps of control flow

XSLT

A programming language with a functional flavor

- Specifies (stylesheet) transforms from documents to documents
- Can be included in a document (best not to)

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
   href="URL-to-xsl-sheet"?>
<main-element>
  ...
</main-element>
```



XQuery versus XSLT: 1

Competitors in some ways, but

- Share a basis in XPath
- Consequently share the same data model
- Same type systems (in the type-sensitive versions)
- XSLT got out first and has a sizable following, but XQuery has strong backing among vendors and researchers

XQuery versus XSLT: 3

There is a bit of an arms race between them

- Types
  - XSLT 1.0 didn't support types
  - XQuery 1.0 does
  - XSLT 2.0 does too
- XQuery presumably will be enhanced with capabilities to make updates, but XSLT could too

Integrity Constraints in XML

Schema

- Entity: **xsd:unique** and **xsd:key**
- Referential: **xsd:keyref**
- Data type: XML Schema specifications xs:string
- Value: Solve custom queries using XPath or XQuery

Entity and referential constraints are based on XPath

a fragment of

XML Constraints: 1

Keys serve as generalized identifiers, and are captured via XML Schema elements:

- *Unique:* candidate key
  - The selected elements yield unique field tuples
- *Key:* primary key, which means candidate key plus
  - The tuples exist for each selected element
- *Keyref:* foreign key
  - Each tuple of fields of a selected element corresponds to an element in the referenced key

XML Constraints: 2

Two subelements built using restricted application of XPath from within XML Schema

- *Selector:* specify a set of objects: this is the scope over which uniqueness applies
- *Field:* specify what is unique for each member of the above set: this is the identifier within the targeted scope
  - Multiple fields are treated as ordered to produce a tuple of values for each member of the set
  - The order matters for matching **keyref** to **key**

Composite

Selector XPath Expression

A selector finds descendant elements of the context node

- The sublanguage of XPath used *allows*
  - Children via **./child** or **./*** or **child**
  - Descendants via **.//** (not within a path)
  - Choice via **|**
- The subset of XPath used *does not allow*
  - Parents or ancestors
  - **text()**
  - Attributes
  - Fancy axes such as **preceding**, **preceding-sibling**, ...

Field XPath Expression

A field finds a unique descendant element (simple type only) or attribute of the context node

- The subset of XPath used *allows*
  - Children via **./child** or **./***
  - Descendants via **.//** (not within a path)
  - Choice via ~~|~~
  - Attributes via **@attribute** or **@***
- The subset of XPath used *does not allow*
  - Parents or ancestors
  - **text()**
  - Fancy axes such as **preceding**, ...

An element yields its **text()**

XML Foreign Keys

```
<keyref name="..." refer="primary-key-name">
 <selector xpath="..."/>
 <field name="..."/>
</keyref>
```

- Relational requirement: foreign keys don't have to be unique or non-null, but if one component is null, then all components must be null.

- Find innermost

~~= Find its parent~~

func recursively

[output current
embed in func(parent)]

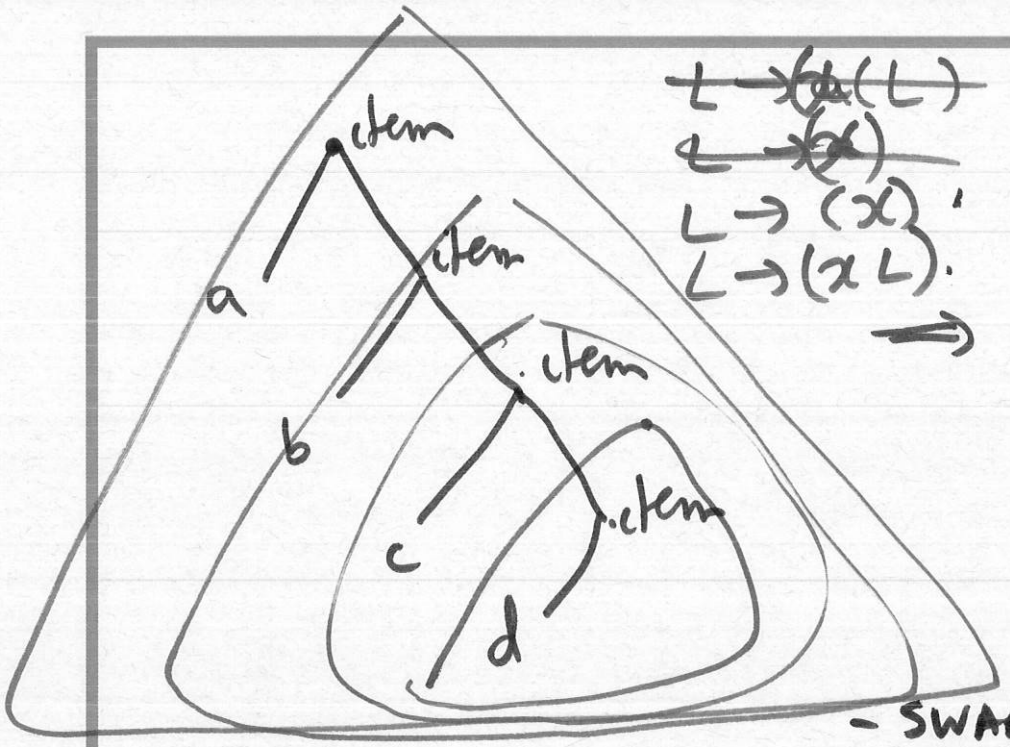
terminate when no parent

from top

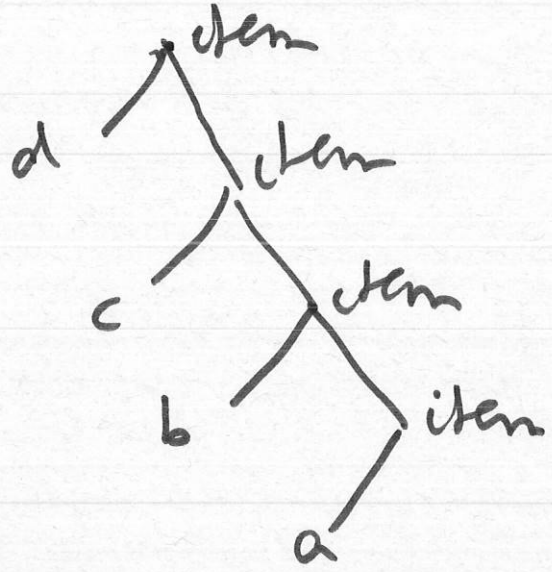
func: if
embed current
inside
func(child)

$$func(x) = x$$

$$func(xL) = embed(func(L), x)$$



~~L → (x (L)~~
~~L → (x)~~
L → (x)
L → (x L).



- SWAP ~~NODE~~ NODE & CHILD ON EACH ITERATION
- n-1 ITERATIONS

ALT: SWAP OUTERMOST & INNERMOST ONE ITERATION

FUNCTIONAL: NO SIDE EFFECTS!

BUILD DESIRED STRUCTURE

RECURSIVE STRUCTURE OF DATA
⇒ RECURSIVE PROGRAM

<item>
a
<item>
b
<item>
c
<item>
</item>
</item>
</item>

⇒

<item>
c
<item>
b
<item>
a
<item>
</item>
</item>
</item>