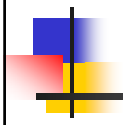
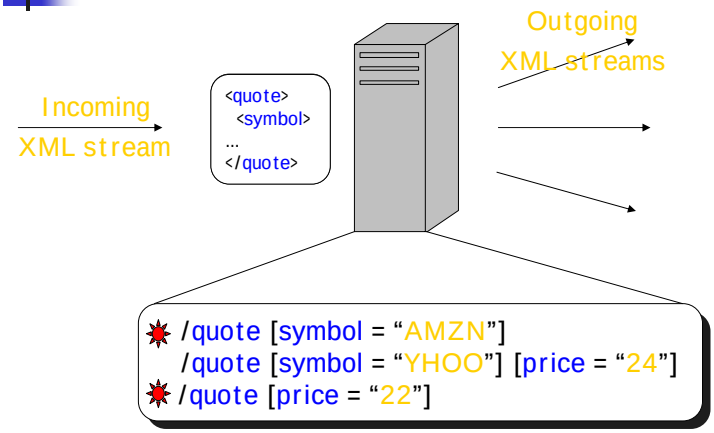


XML Filtering Technologies

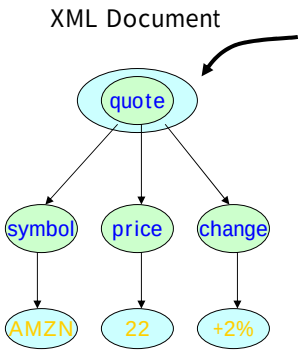
Introduction

- Data exchange between applications:
  - use XML Messages
  - processed by an XML Message Broker
- Examples
  - Publish/subscribe systems [Altinel '00]
  - XML message routing [Snoeren '01]
  - Web services
  - Sensor networks
- Challenge in XML Message brokers: scale

XML Message Broker: The Dispatch Function



XPath Query as a Filter



P_1 /quote [symbol = "AMZN"]

P_1 → TRUE

P_2 /quote [price = "24"]

P_2 → FALSE

The Problem

- Given
  - A set of XPath queries
  - Incoming stream of XML messages
- Compute
  - For each XML message, set of XPath queries it matches
- A Hard Problem
  - Number of XPath queries is large
  - XPath queries are complex
  - Need high throughput of XML messages

Existing Approaches

- XScan - evaluates XPath queries using a DFA
- XFilter, YFilter, XTrie – shared matching of structure
- LazyDFA – complete sharing of structure
- NiagaraCQ – shares the most selective predicate
- Hoffmann and O'Donnell, 1982 - pattern is pre-processed into an exponential size structure
- XPush Machine – shared matching of structure and predicates

Existing Approaches

- In the structure navigation part
  - XFilter – shares tags – [Altinel, Franklin: VLDB'00]
  - XTrie – shares sequences of tags – [Chen, DeWitt, Naughton: ICDE'02]
  - YFilter – shares prefixes – [Diao, Fischer, Franklin, To: ACM TODS'03]
  - LazyDFA – shares everything
  - XPush Machine - [Gupta, Suciu: SIGMOD'03]
- In the predicate evaluation part
  - NiagaraCQ (most selective predicates only) – [Chen, DeWitt, Tian, Wang: SIGMOD'00]
  - YFilter – shares prefixes – [Diao, Fischer, Franklin, To: ACM TODS'03]
  - XPush Machine - [Gupta, Suciu: SIGMOD'03]

Central Dogma of Filtering

- In a traditional *database system*, a large set of data is stored persistently. Queries, coming one at a time, search the data for results.
- In a *filtering system*, a large set of queries is persistently stored. Documents, coming one at a time, drive the matching of the queries.

Selective Dissemination of Information (SDI)

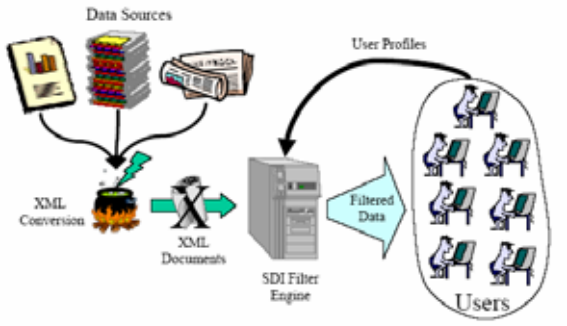


Figure 1: Architecture of an XML Based SDI System

Exploits

- The shared nature of profiles, or standing queries.
- Evaluate Queries simultaneously.
- Perform single evaluations of common structural prefix hierarchies.
- Apply fundamental data structures and methodologies.

Terminology

- Path expression – Query or profile
- Profile – Standing Query
- FSM- Finite State Machine
- NFA – Non Deterministic Finite Automata
- XPath – A query language
- XParser – An event driven parser
- Document Type Definition – general set of rules for a document's elements and attributes.

X-Filter System

[Altinel, Franklin: VLDB'00]

X-Filter: Internal Query Representation

- *Profiles* constitute better half of a filtering system.
- Each XPath query is disassembled into a set of *path nodes* by the XParser.
- Path nodes represent the States of the FSM for the query.
- Path nodes are NOT generated for "*" wildcard nodes.

Path Node Contents

- **Query ID -** unique identifier for the query, arbitrarily assigned by XPath Parser.
- **Position –** A sequence number, relative to the other nodes in a query.
- **RelativePos** – distance in levels between current node and previous path node.
- **Level** – Level in the XML document where current path node should be checked.
- **NextPathNodeSet** – Pointer to next path node of the query to be evaluated.

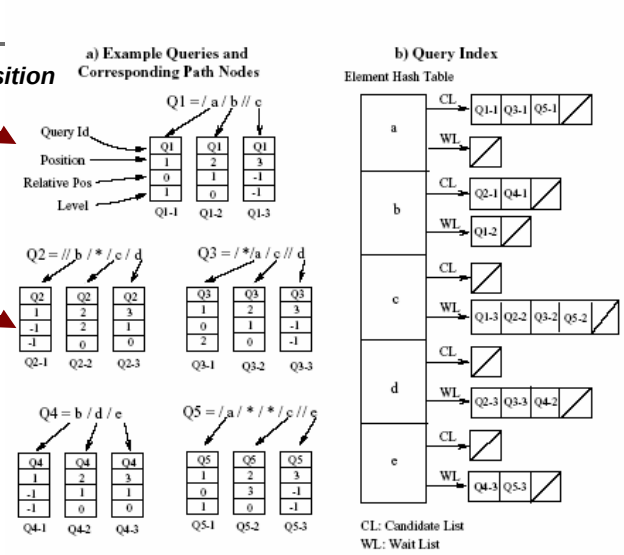
Path Nodes

Query Id & ***Position*** are trivial

RelativePos
-1, if node follows '//'

0, if Not and first node in path

else
1 + number Wildcards (*)

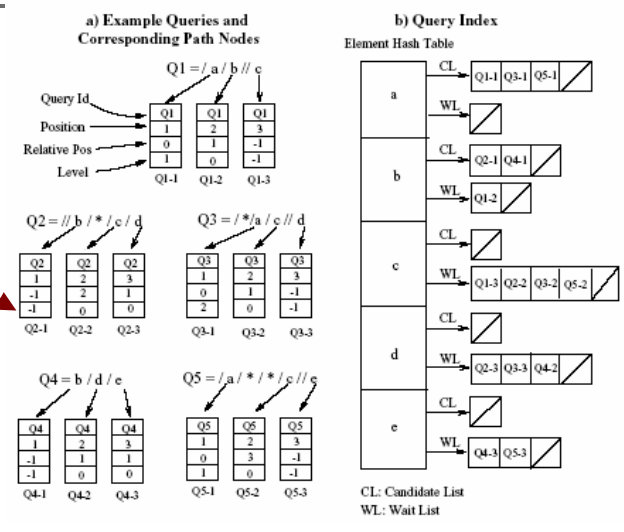


Path Nodes (cont'd)

Level
-1, if RelativePos is

If node is first in query and specifies abs(distance) from root,
1+distance

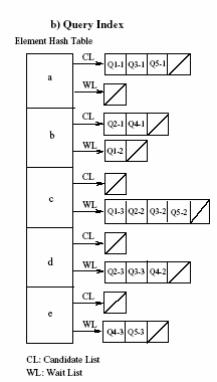
0 otherwise



Path Node Conversion

- XPath Expressions get converted into path nodes by the XPath parser.
- These nodes are then added to the Query Index.
- Query Index organized as a hash table based on the element names that appear in XPath expressions.
- Each unique element has a *Candidate and Waiting* List.

Index Membership



Candidate Lists- correspond to the states of that the FSM is currently attempting to match

Waiting Lists- nodes subsequent to the candidate nodes.

Index Construction

- Performance empirically shown to be dependent on initial distribution of path nodes.
- Naïve approach, initial states are placed into *candidate* list, rest in *waiting*
- *Problem 1-* Poor selectivity due to lack of depth in document, possible element names smaller.
- *Problem 2-* Candidate Lists become highly skewed, reduction of queries considered lost.

List Balance Approach

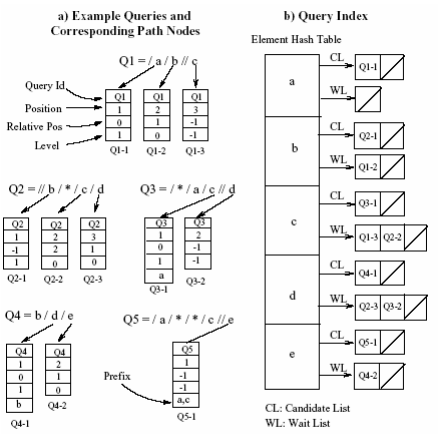
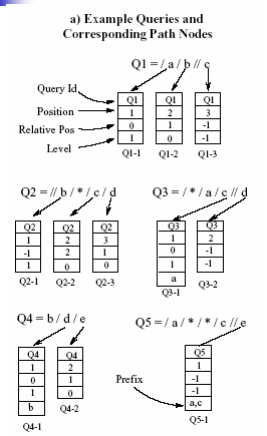
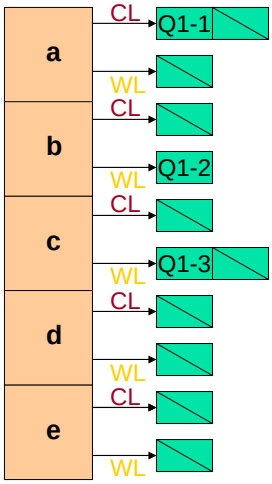


Figure 5: Path Nodes and the content of the Query Index in List Balance

List Balance Algorithm





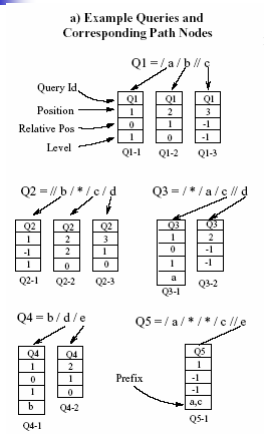
Q1 = **/ a / b // c**

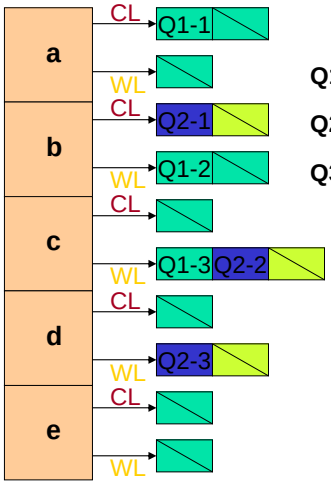
Q2 = **// b / * / c / d**

Select a 'pivot' for the query.

Pivot is the first node with shortest candidate list.

List Balance Algorithm



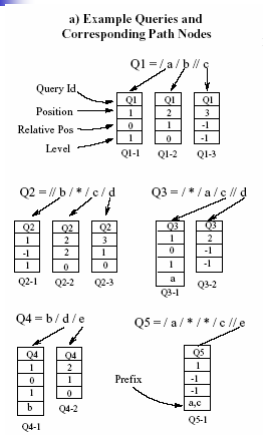


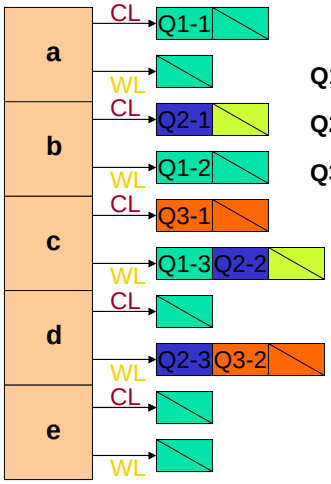
Q1 = **/ a / b // c**

Q2 = **// b / * / c / d**

Q3 = **/ * / a / c // d**

List Balance Algorithm





Q1 = **/ a / b // c**

Q2 = **// b / * / c / d**

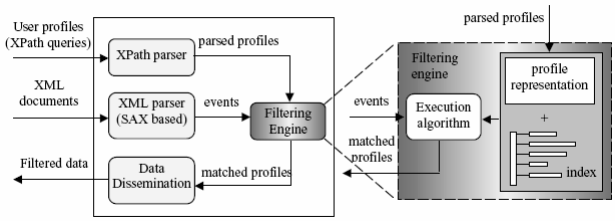
Q3 = **/ * / a / c // d**

'c' is a pivot. **'a'** goes on stack.

Prefix

- FSM of query modified so that its initial state is the **pivot** node.
- Represent the portion that precedes the pivot node as a "prefix"
- Prefix is checked as a pre-condition in the evaluation of a path node.
- List Balance uses a stack that keeps track, fast forward execution of the portion of the FSM.

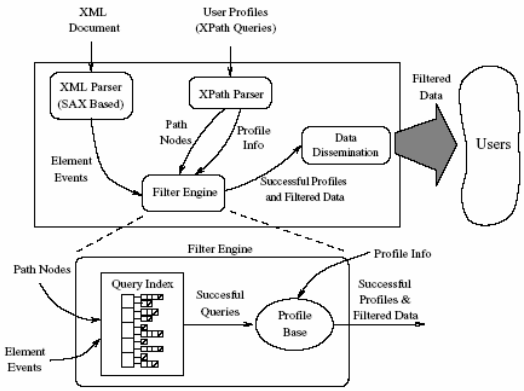
Filter Components



1) XPath Parser
2) Event-based XML parser
3) Filtering Engine
4) Dissemination – via unicast upon a match

NOTE: If a single Query Path (profile) matches any portion of a document, the entire document gets sent.

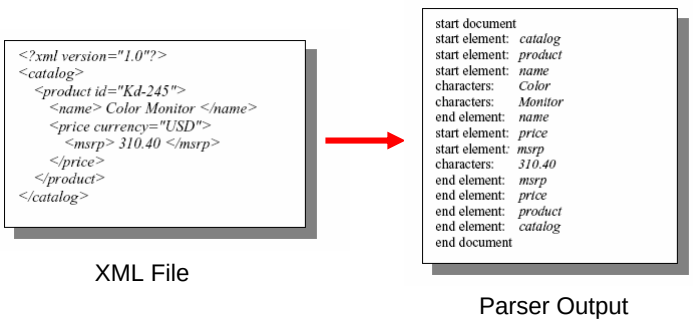
Architecture of the X-Filter Engine



Event Driven X-Filter Execution

- Document arrives at the filtering engine.
- Run thorough an XML Parser, which reports back events that are used in profile matching.
- Callback handles ‘start’ and ‘end’ for events passed name and document level of element for (on in) when event occurred.

Event-based XML parser: Sample SAX API Output



XML File

Parser Output

Execution Algorithm

- Start Element Handler – A start element calls this handler.
- Handler looks up element name in Query Index, and examines all nodes in the candidate list for that element.
- Level is checked, if non-negative, levels must be identical to each other, otherwise level is unrestricted, passes anyway:
  - Match if node is final node in path.
  - Otherwise promote next node from waiting to candidate list.
  - Note: Copy of promoted node remains in the wait list.

Execution Algorithm (cont'd)

- If the *RelativePos* of the copied node is not -1, its level must be updated using current level and *Relative Pos*, to allow correct future checks.
- End Element Handler – end element tag encountered, path nodes promoted to wait list are deleted, restoring those lists to state they were in before reading an element.

Execution Algorithm Wrap-Up

- The restoration process allows for the “backtracking” capacity necessary to handle the case where the same element appears at different levels in the document.
- When the same element appears at nested levels corresponding to a ‘//’ step then multiple copies of the subsequent path node can exist in its corresponding candidate list, reflecting the different levels where it can be matched

Y-Filter System

[Diao, Fischer, Franklin, To: ACM TODS’03]

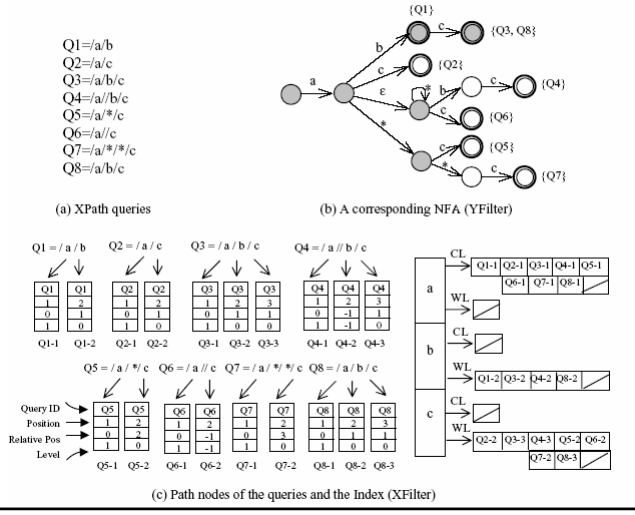
Y-Filter

- An NFA-based approach that attempts to exploit the path sharing of profiles.
- Why? Because people are inherently similar, maybe not at an increasing granularity, but assuredly in a general way.
- Two people read the *Times,* one reads the Sports section, the other the Local News, both read the *Fry's Electronics* add.

NFA Advantages

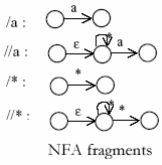
- A relatively small number of machine states required to represent even large numbers of path expressions.
- The ability to support complicated document types
  - Nesting
  - Multiple ancestor/descendant relation
- Incremental Construction & Maintenance, new queries added to an existing system, as they come into existence.

A Comparison X vs. Y



NFA Construction

- Break down the four basic location steps:
  - “ / a ”
  - “ // a ”
  - “ / * ”



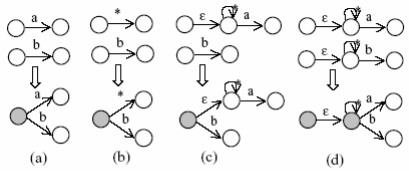


Figure 9: Combining NFA Fragments

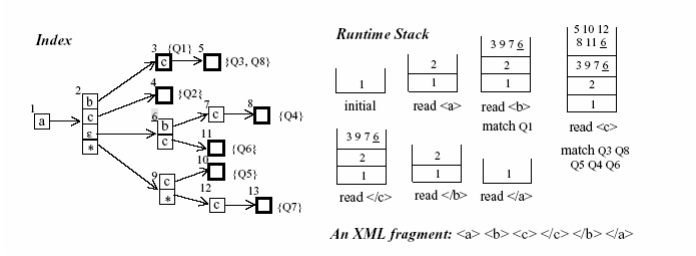
NFA Structure

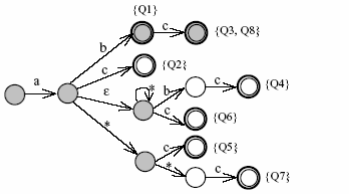
- Each state contains a(n):
  - ID
  - Type (accepting state, or '//-child'
  - Small Hash Table containing all transitions
  - For accepting states, a list of relevant queries Q1, Q2, …Qn

Event Driven Execution

- Once again the events raised by the parser callback the *handlers* that drive transition through NFA.
- A stack mechanism is used to backtrack to the "start-of-element" when "end-of-element" event is raised.
- An example…

Example NFA Execution





Hybrid Approach

- An improved version of X-Filter for path sharing.
- Hybrid decomposes ‘ * ‘ and ‘ // ’ into strictly ‘ / ‘ operators
- Hybrid Path Nodes’ *RelativePos* here specifies distance in document from the previous substring to this substring.

Empirical Results: Query Size Increases

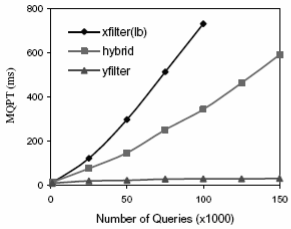


Figure 11: Varying number of distinct queries (NITF, D=6, W=0.2, DS=0.2)

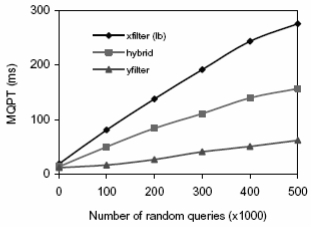


Figure 12: Varying number of queries (with duplicates) (NITF, D=6, W=0.2, DS=0.2)

Metric: **Multi-Query Processing Time (MQPT)** = Wall clock time from start to finish of parsing documents to the end of output minus document parsing time.

NITF – News Industry Text Format

Y-Filter Performance Benefits

- Remember that the NFA exploits shared prefix, not identical queries, these are treated the same as single queries in all three methods.
- The hash based transition table inside of each state in the Y-Filter makes transitioning much faster.
  - Empirically 7.4 times the transitions for X-Filter over Y-Filter took about 25 times longer.

Maintaining the NFA

- Modification of queries are treated as insert/delete operations of the old query and replacement query respectively.
- Inserting obviously gets to be less labor intensive as the number of queries increases and less chance for uniqueness.

Q (x1000)	2	4	6	8	10	10 ~ 50	60 ~ 500
1000 Insertions (ms)	77	57	30	24	9	6	≈ 5

Table 5: Cost of inserting 1000 queries (ms) (NITF, D=6, W=0.2, DS=0.2)

X-Filter v.s. Y-Filter

- X-Filter began the process of evaluating queries in an expedited fashion by evaluating queries in parallel.
- Y-Filter exploited the shared path nature of query processing for structural matching.
- Partial document retrieval and more refined delivery mechanisms are surely on their way, to better hit define and strike their targets.

Value-Based Predicate Evaluation

- *Inline* - Extend the information stored at each state of the NFA to include predicates that are associated with that state.
- While conceptually simple, two caveats
- 1) The predicate failure at a state does not necessarily stop processing, i.e. '//' prior to predicate. Query could stay active.
- 2) Recursively nested 'a'
  - <a a1 = v1><a a2 = v2>

Value-Based Selection Postponed

- Effort spent evaluating predicates with Inline will be wasted if structural based aspects of a query are NOT satisfied.
- SP delays predicate processing until after the structure matching is complete.
- Predicates are stored with each Query in tables.

Selection Postponed (SP)

Index the predicates stored

In a particular query

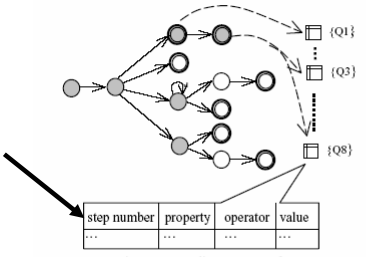


Figure 20: Predicate Storage for SP

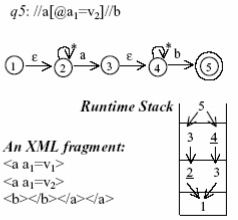


Figure 21: A sample query, its NFA, and the NFA execution

Now need some way of preserving the path, in the run-time stack. This 'backward chaining', a technique similar to PathStack and TwigStack is used.

Differences between SP and Inline

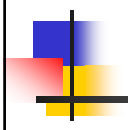
- Structure v. Value Matching
- Inline performs early predicate matching before structure matched, does Not prune future work.
- SP performs structure matching to prune set of queries for which predicate evaluation needs to be performed.

Differences between SP and Inline

- Conjunctive predicates in a query
- Inline, evaluation of predicates in the same query happen independently at different states.
- SP, a failure at any states stops the evaluation of all subsequent predicates.

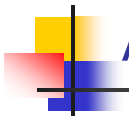
Differences between SP and Inline

- Bookkeeping – Inline requires information bookkeeping information for the final evaluation of the query
- Includes setting information and undoing it during backtracking.
- Memory runs out at 400,000 Q. Does not scale.



Stream Processing of XPath Queries with Predicates

[Gupta, Suciu: SIGMOD 2003]



Approach

- XML message = list of tokens (a.k.a. SAX events)
  - Normally, one token affects several queries
- Our goal: for each token perform a single action !
  - This eliminates all shared computations in queries
- Need to build a deterministic machine
  - With a stack
  - With ability to handle predicates
- XPush Machine = a modified pushdown automata

Approach (continued)

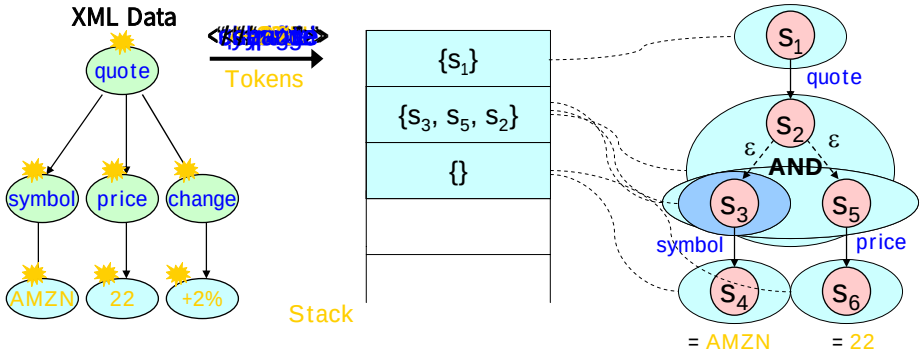
- XPath fragment:

child	/	/quote
descendant-or-self	//	//symbol
wildcard	*	/quote/*
qualifiers	[ ]	/quote [change]
Predicates (path opRel "const")		/quote [symbol = "YHOO" **AND** price = "24"]

- XML tokens:
  - beginElement, endElement, value

Matching One XPath Query

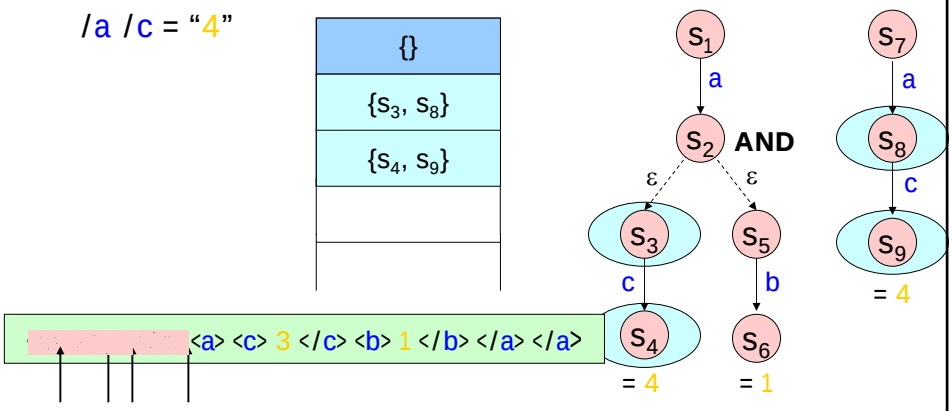
/quote [symbol = "AMZN" **AND** price = "22"]



More than one XPath Query

/a [c = "4" **AND** b = "1"]

/a /c = "4"

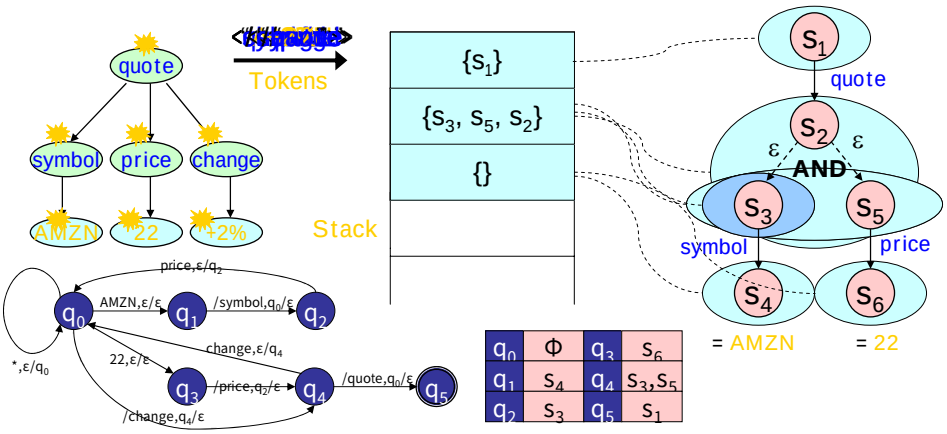


The XPush Machine

- A Modified Pushdown Automata
  - On a beginElement → push (hashtable lookup)
  - On an endElement → pop (hashtable lookup)
  - On a value → "predicate index" lookup
- Deterministic: a single action on each token
- Space
  - Exponential (in worst case)
  - XPush state = a (large?) set of XPath nodes
- Compute the XPush Machine lazily:
  - fill push/pop/predicate tables on a by-need basis
  - run-time penalty

Matching One XPath Query

/quote [symbol = "AMZN" **AND** price = "22"]



Questions & Comments